8*x*931AA, 8*x*931HA Universal Serial Bus Peripheral Controller User's Manual

September 1997

CONTENTS

FIGURES

TABLES

1

Guide to this Manual

CHAPTER 1
GUIDE TO THIS MANUAL

This manual describes the 8*x*931 microcontroller for universal serial bus (USB) applications. This manual is intended for use by both firmware and hardware designers familiar with the principles of microcontroller architecture.

1.1 MANUAL CONTENTS

This chapter provides an overview of the manual with brief summaries of the chapters and appendices. It also explains the terminology and notational conventions used throughout the manual, provides references to related documentation, and tells how to contact Intel for additional information.

Chapter 2, "Architectural Overview"— provides an overview of device hardware. It covers core functions (CPU, clock and reset unit, and interrupts**)**, I/O ports, on-chip memory, the USB module, and on-chip peripherals (timer/counters and serial I/O port).

Chapter 3, "Address Spaces"— describes the three address spaces of the 8*x*931: memory address space, special function register (SFR) space, and the register file. It also provides a map of the SFR space showing the location of the SFRs and their reset values and explains the mapping of the address spaces relative to the MCS® 51 architecture into the address spaces of the 8*x*931.

Chapter 4, "Programming Considerations"— provides an overview of the instruction set. It describes each instruction type (control, arithmetic, logical, etc.) and lists the instructions in tabular form. This chapter also discusses the addressing modes, bit instructions, and the program status words. Appendix A, "Instruction Set Reference" provides a detailed description of each instruction.

Chapter 5, "Interrupt System"— describes the 8*x*931 interrupt circuitry which provides ten maskable interrupts: three external interrupts, three timer interrupts, a serial port interrupt, and three USB interrupts. This chapter also discusses the interrupt priority scheme, interrupt enable, interrupt processing, and interrupt response time.

Chapter 6, "USB Function"— describes the FIFOs and special function registers (SFRs) associated with the USB function interface. This chapter describes the operation of function interface on the 8*x*931 USB microcontrollers.

Chapter 7, "USB Hub"— describes the operation of the Intel Universal Serial Bus (USB) on-chip hub. This chapter introduces on-chip hub operation and includes information on bus enumeration, hub endpoint status and configuration, hub port control, hub suspend and resume, and hub power control.

Chapter 8, "USB Programming Models"— describes the programming models of the 8*x*931 USB function interface. This chapter provides flow charts of suggested firmware routines for using the transmit and receive FIFOs to perform data transfers between the host PC and the embedded function and describes how the firmware interacts with the USB module hardware.

Chapter 9, "Input/Output Ports"— describes the four 8-bit I/O ports (ports 0–3) and discusses their configuration for general-purpose I/O. This chapter also discusses external memory accesses (ports 0, 2) and alternative special functions.

Chapter 10, "Timer/Counters"—describes the three on-chip timer/counters and discusses their application.

Chapter 11, "Serial I/O Port"—describes the full-duplex serial I/O and explains how to program it to communicate with external peripherals. This chapter also discusses baud rate generation, framing error detection, multiprocessor communications, and automatic address recognition.

Chapter 12, "Keyboard Control"— describes the 8*x*931 keyboard control interface, including the keyboard scan output lines, the keyboard scan input lines, and the LED drivers.

Chapter 13, "Minimum Hardware Setup"— describes the basic requirements for operating the 8*x*931 in a system. It also discusses on-chip and external clock sources.

Chapter 14, "Special Operating Modes"— provides an overview of the idle, powerdown, and on-circuit emulation (ONCE) modes and describes how to enter and exit each mode. This chapter also describes the power control (PCON) special function register and lists the status of the device pins during the special modes.

Chapter 15, "External Memory Interface"— describes the external memory signals and bus cycles and provides examples of external memory design. It also provides waveform diagrams for the bus cycles.

Chapter 16, "Verifying Nonvolatile Memory"— provides instructions for verifying on-chip program memory, signature bytes, and lock bits.

Appendix A, "Instruction Set Reference"— provides reference information for the instruction set. It describes each instruction; defines the bits in the program status word register (PSW); shows the relationships between instructions and PSW flags; and lists hexadecimal opcodes, instruction lengths, and execution times.

Appendix B, "Pin Descriptions"— describes the function(s) of each device pin. Descriptions are listed alphabetically by signal name. This appendix also provides a list of the signals grouped by functional category.

Appendix C, "Registers"— accumulates, for convenient reference, copies of the register definition figures that appear throughout the manual.

Appendix D, "Data Flow Model"— describes the data flow model for the 8*x*931 USB transactions.

Appendix E, "8x931AA Design Considerations"—describes the differences between the hubless 8*x*931AA and the 8*x*931HA.

Glossary — a glossary of terms has been provided for reference of technical terms.

Index — an index has been included for your convenience.

1.2 NOTATIONAL CONVENTIONS AND TERMINOLOGY

The following notations and terminology are used in this manual. The Glossary defines other terms with special meanings.

— The pound symbol (#) has either of two meanings, depending on the context. When used with a signal name, the symbol means that the signal is active low. When used with an instruction mnemonic, the symbol prefixes an immediate value in immediate addressing mode.

italics — Italics identify variables and introduce new terminology. The context in which italics are used distinguishes between the two possible meanings.

Variables in registers and signal names are commonly represented by *x* and *y*, where *x* represents the first variable and *y* represents the second variable. For example, in register P*x.y*, *x* represents the variable [1–4] that identifies the specific port, and *y* represents the register bit variable [7:0]. Variables must be replaced with the correct values when configuring or programming registers or identifying signals.

XXXX, xxxx — Uppercase X (no italics) and lowercase x (no italics) represent unknown values or a “don’t care” states or conditions. The value may be either binary or hexadecimal, depending on the context. For example, 2xAFH (hex) indicates that bits 11:8 are unknown; 10xx in binary context indicates that the two LSBs are unknown.

Assert and Deassert — The terms *assert* and *deassert* refer to the act of making a signal active (enabled) and inactive (disabled), respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To assert RD# is to drive it low; to assert ALE is to drive it high; to deassert RD# is to drive it high; to deassert ALE is to drive it low.

Instructions — Instruction mnemonics are shown in upper case to avoid confusion. When writing code, either upper case or lower case may be used.

Logic 0 (Low) — An input voltage level equal to or less than the maximum value of V_{IL} or an output voltage level equal to or less than the maximum value of V_{OL}. See data sheet for values.

Logic 1 (High) — An input voltage level equal to or greater than the minimum value of V_{IH} or an output voltage level equal to or greater than the minimum value of V_{OH}. See data sheet for values.

Numbers Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character *H*. Decimal and binary numbers are represented by their customary notations. That is, 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter *B* is added for clarity.

Register Access All register bits support read/write access unless noted otherwise in the bit description. Other types of access include read-only, write-only, read/conditional-write, etc.

Register Bits Bit locations are indexed by 7:0 for byte registers, 15:0 for word registers, and 31:0 for double-word (dword) registers, where bit 0 is the least-significant bit and 7, 15, or 31 is the most-significant bit. An individual bit is represented by the register name, followed by a period and the bit number. For example, PCON.4 is bit 4 of the power control register. In some discussions, bit names are used. For example, the name of PCON.4 is POF, the power-off flag.

Register Names Register names are shown in upper case. For example, PCON is the power control register. If a register name contains a lowercase character, it represents more than one register. For example, CCAPM*x* represents the five registers: CCAPM0 through CCAPM4.

Reserved Bits Some registers contain reserved bits. These bits are not used in this device, but they may be used in future implementations. Do not write a "1" to a reserved bit. The value read from a reserved bit is indeterminate.

Set and Clear The terms *set* and *clear* refer to the value of a bit or the act of giving it a value. If a bit is *set*, its value is "1"; *setting* a bit gives it a "1" value. If a bit is *clear*, its value is "0"; *clearing* a bit gives it a "0" value.

Signal Names Signal names are shown in upper case. When several signals share a common name, an individual signal is represented by the signal name followed by a number. Port pins are represented by the port abbreviation, a period, and the pin number (e.g., P0.0, P0.1). A pound symbol (#) appended to a signal name identifies an active-low signal.

Units of Measure The following abbreviations are used to represent units of measure:

A	amps, amperes
DCV	direct current volts
Kbyte	kilobytes
KΩ	kilo-ohms

mA	milliamps, milliamperes
Mbyte	megabytes
MHz	megahertz
ms	milliseconds
mW	milliwatts
ns	nanoseconds
pF	picofarads
W	watts
V	volts
µA	microamps, microamperes
µF	microfarads
µs	microseconds
µW	microwatts

1.3 RELATED DOCUMENTS

The following documents contain additional information that is useful in designing systems that incorporate the 8*x*931. To order documents, please call Intel Literature Fulfillment (1-800-548-4725 in the U.S. and Canada; +44(0) 793-431155 in Europe).

Embedded Microcontrollers	Order Number 270646
Embedded Processors	Order Number 272396
Embedded Applications	Order Number 270648
Packaging	Order Number 240800
Universal Serial Bus Specification	Order Number 272904
MCS® 51 Microcontroller Family User's Manual	Order Number 272383

1.3.1 Data Sheet

The data sheet is included in *Embedded Microcontrollers* and is also available individually.

8*x*931AA/8*x*931HA *Universal Serial Bus Microcontroller*	Order Number 273108

1.3.2 Application Notes

The following MCS 51 microcontroller application notes also apply to the 8*x*931.

AP-70, Using the Intel MCS® 51 Boolean Processing Capabilities	Order Number 203830
AP-223, 8051 Based CRT Terminal Controller	Order Number 270032
AP-252, Designing With the 80C51BH	Order Number 270068
AP-425, Small DC Motor Control	Order Number 270622
AP-410, Enhanced Serial Port on the 83C51FA	Order Number 270490
AP-476, How to Implement I^2C Serial Communication Using Intel MCS® 51 Microcontrollers	Order Number 272319

1.4 APPLICATION SUPPORT SERVICES

You can get up-to-date technical information from a variety of electronic support systems: the World Wide Web, the FaxBack* service, and Intel's Brand Products and Applications Support

bulletin board service (BBS). These systems are available 24 hours a day, 7 days a week, providing technical information whenever you need it.

In the U.S. and Canada, technical support representatives are available to answer your questions between 5 a.m. and 5 p.m. Pacific Standard Time (PST). Outside the U.S. and Canada, please contact your local distributor. You can order product literature from Intel literature centers and sales offices.

Table 1-1 lists the information you need to access these services.

Table 1-1. Intel Application Support Services

Service	U.S. and Canada	Asia-Pacific and Japan	Europe
World Wide Web	URL:http://www.intel.com/	URL:http://www.intel.com/	URL:http://www.intel.com/
World Wide Web	URL:http://www.intel.com/design/usb/	URL:http://www.intel.com/design/usb/	URL:http://www.intel.com/design/usb/
FaxBack*	800-525-3019	503-264-6835 916-356-3105	+44(0)1793-496646
BBS	503-264-7999 916-356-3600	503-264-7999 916-356-3600	+44(0)1793-432955
Help Desk	800-628-8686 916-356-7999	Please contact your local distributor.	Please contact your local distributor.
Literature	800-548-4725	708-296-9333 +81(0)120 47 88 32	+44(0)1793-431155 England +44(0)1793-421777 France +44(0)1793-421333 Germany

1.4.1 World Wide Web

We offer a variety of technical and product information through the World Wide Web (URL: http://www.intel.com/design/usb). Also visit Intel's Web site for financials, history, news and USB information at: www.intel.com/design/.

1.4.2 FaxBack Service

The FaxBack service is an on-demand publishing system that sends documents to your fax machine. You can get product announcements, change notifications, product literature, device characteristics, design recommendations, and quality and reliability information from FaxBack 24 hours a day, 7 days a week.

Think of the FaxBack service as a library of technical documents that you can access with your phone. Just dial the telephone number and respond to the system prompts. After you select a document, the system sends a copy to your fax machine.

Each document is assigned an order number and is listed in a subject catalog. The first time you use FaxBack, you should order the appropriate subject catalogs to get a complete listing of document order numbers. Catalogs are updated twice monthly. In addition, daily update catalogs list the title, status, and order number of each document that has been added, revised, or deleted during the past eight weeks. The daily update catalogs are numbered with the subject catalog number followed by a zero. For example, for the complete microcontroller and flash catalog, request document number 2; for the daily update to the microcontroller and flash catalog, request document number 20.

The following catalogs and information are available at the time of publication:

1. *Solutions OEM* subscription form
2. Microcontroller and flash catalog
3. Development tools catalog
4. Systems catalog
5. Multimedia catalog
6. Multibus and iRMX® firmware catalog and BBS file listings
7. Microprocessor, PCI, and peripheral catalog
8. Quality and reliability and change notification catalog
9. iAL (Intel Architecture Labs) technology catalog

1.4.3 Bulletin Board System (BBS)

Intel's Brand Products and Applications Support bulletin board system (BBS) lets you download files to your PC. The BBS has the latest *Ap*BUILDER firmware, hypertext manuals and datasheets, firmware drivers, firmware upgrades, application notes and utilities, and quality and reliability data.

Any customer with a PC and modem can access the BBS. The system provides automatic configuration support for 1200- through 19200-baud modems. Use these modem settings: no parity, 8 data bits, and 1 stop bit (N, 8, 1).

To access the BBS, just dial the telephone number (see Table 1-1 on page 1-7) and respond to the system prompts. During your first session, the system asks you to register with the system operator by entering your name and location. The system operator will set up your access account within 24 hours. At that time, you can access the files on the BBS.

NOTE

In the U.S. and Canada, you can get a BBS user's guide, a master list of BBS files, and lists of FaxBack documents by calling 1-800-525-3019. Use these modem settings: no parity, 8 data bits, and 1 stop bit (N, 8, 1).

2

Architectural Overview

CHAPTER 2
ARCHITECTURAL OVERVIEW

The 8*x*931AA and 8*x*931HA are PC peripheral controllers for Universal Serial Bus (USB) applications. These peripheral controllers provide the means for connecting PC peripherals such as monitors, keyboards, joysticks, telephones, and modems to USB-equipped personal computers. For keyboard applications, both devices include an on-chip keyboard control interface. The USB material in this document relies heavily on the *Universal Serial Bus Specification* which provides a detailed description of the USB system.

In the language of the USB specification, the 8*x*931AA/HA is a *USB device*. A USB device can serve as a *function* by providing an interface for a PC peripheral, and it can serve as a *hub* by providing USB ports for additional PC peripherals.

The 8*x*931AA is a hubless USB peripheral controller which serves as a USB function. The 8*x*931HA serves as both a USB function and as a hub; it supports one embedded function and provides four external downstream ports. Figure 2-1 depicts the 8*x*931 in an example USB system.

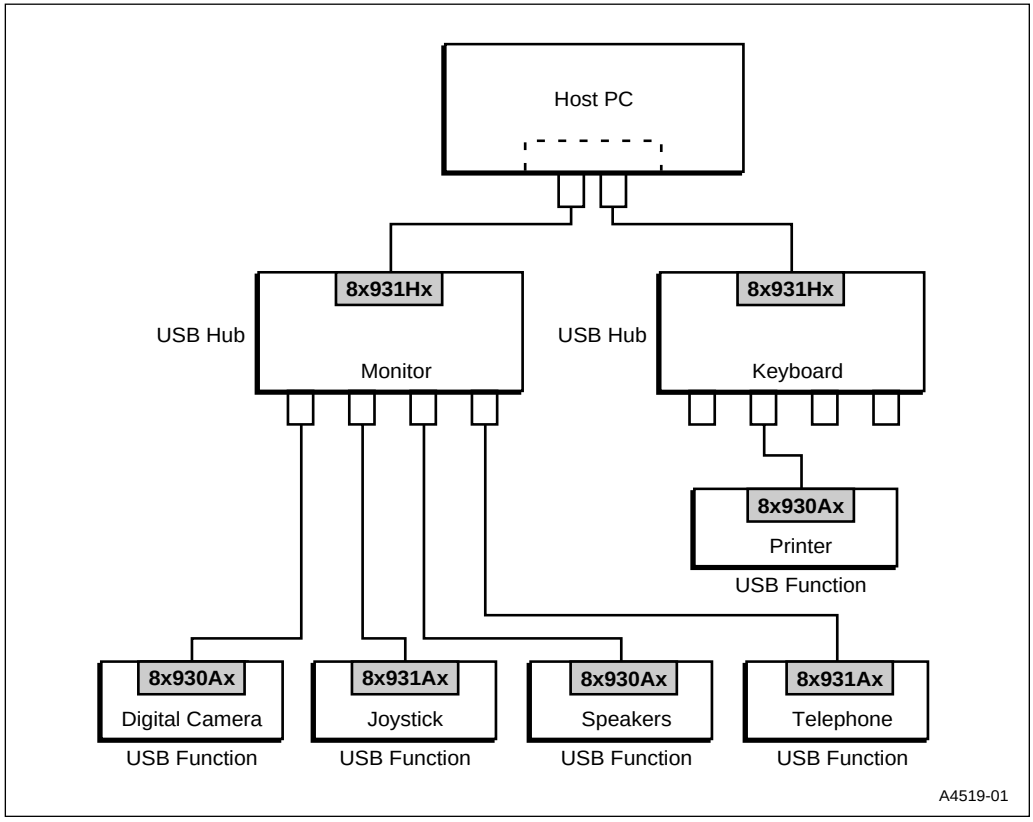


Figure 2-1. 8*x*931 in a USB System

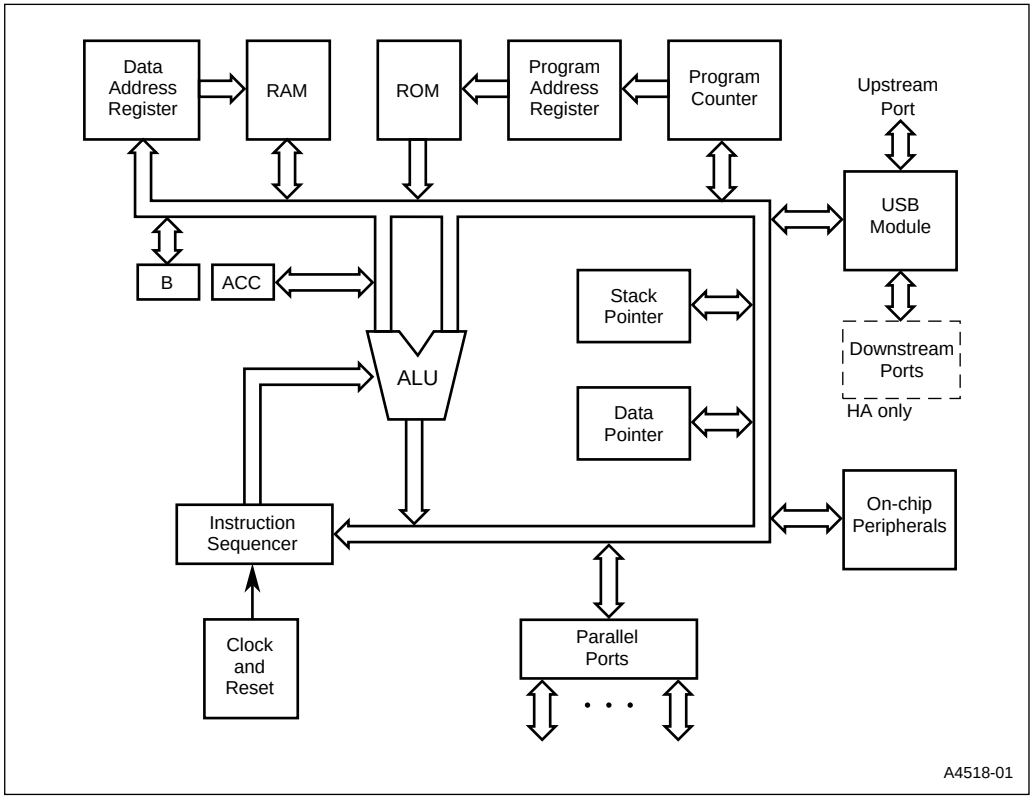


Figure 2-2. Functional Block Diagram of the 8*x*931

2.1 PRODUCT OVERVIEW

The 8*x*931 employs the architecture of the MCS® 51 microcontroller family. Specifically, it is derived from the 8*x*C51F*x* core which is optimized for control operations with extensive boolean processing capabilities. The 8*x*931 executes the standard instruction set of the MCS 51 architecture.

A functional block diagram of the 8*x*931 is shown in Figure 2-2. The 8*x*931 contains a microcontroller core, a USB module, a keyboard control interface, on-chip ROM (optional) and RAM, four 8-bit parallel ports, and on-chip peripherals (timer/counters and serial port). The USB module operates in conjunction with the CPU to provide the capabilities of a USB device. It supports all four types of USB data transfers: control, isochronous, interrupt, and bulk. Dedicated pinouts are provided for USB signals.

The 8*x*931 is available in ROMless and factory-programmed ROM versions in 64-pin S-DIP, 64-pin QFP, and 68-pin PLCC packages. See Appendix B for package diagrams, pin assignments, and signal descriptions. Table 2-1 lists the on-chip RAM and ROM memory options.

Table 2-1. 8*x*931 Memory Options

8*x*931AA (Hubless)	8*x*931HA (Hub)	On-chip Memory	
		ROM (Kbytes)	RAM (Bytes)
80931AA	80931HA	0	256
83931AA	83931HA	8	256

The 8*x*931 provides a rich set of microcontroller features. The following sections describe the major features. Table 2-2 on page 2-4 summarizes these features and provides an item-by-item comparison of the 8*x*930H*x* and 8*x*931H*x* and the 8*x*930A*x* and 8*x*931A*x*. The 8*x*931 is based on the MCS® 51 architecture, whereas the 8*x*930H*x* is based on the MCS® 251 architecture.

For detailed description of the 8*x*C51F*x* hardware, programmer's model, and instruction set, see the *MCS 51 Microcontroller Family User's Manual*, order number 272383.

For further information on the 8*x*931, see "Microcontroller Core" on page 2-6, and "Universal Serial Bus Module" on page 2-11.

2.1.1 8*x*931AA Features

The 8*x*931AA provides a USB interface for one PC peripheral. The 8*x*931AA function interface provides three function endpoint pairs with corresponding transmit/receive FIFO pairs. Function endpoint 0 supports control data transfers only, while function endpoints 1 and 2 support control, interrupt, and bulk data transfers. Function endpoint 1, which has 16-byte FIFOs, also supports isochronous data transfers. See Table 2-5 on page 2-12 for endpoint pair information.

2.1.2 8*x*931HA Features

The 8*x*931HA also provides a USB hub capability, permitting the connection of additional PC peripherals or hubs. In addition to an upstream port to the host PC (USB root port), the 8*x*931HA provides four external downstream ports (with ganged power switching), and an internal downstream port for the embedded function. The 8*x*931HA provides on-chip transceivers for each of the external USB ports.

The 8*x*931HA has two hub endpoint pairs: endpoint 0 which supports 8-byte control data transfers and endpoint 1 which transmits a status change byte to the host PC. See Table 2-2 for a summary of USB features and Table 2-5 on page 2-12 for endpoint pair information. See Figure 2-3 for the 8*x*931HA USB module block diagram.

NOTE

The 8*x*931AA microcontroller does not support hub operations. Specific details of the 8*x*931AA are covered in Appendix E, "8x931AA Design Considerations".

2.1.3 Keyboard Control Interface

The 8*x*931 contains a keyboard control interface with a 20-bit by 8-bit scan capability and four LED drivers. Chapter 12, "Keyboard Control", describes this further.

Table 2-2. USB Peripheral Controller Feature Summary and Comparison

	8*x*931H*x*	**8*x*931A*x***	**8*x*930H*x***	**8*x*930A*x***
General Features				
On-chip ROM	0, 8 Kbytes	0, 8 Kbytes	0, 8 or 16 Kbytes	0, 8 or 16 Kbytes
On-chip RAM	256 bytes	256 bytes	1024 bytes	1024 bytes
On-chip peripherals:				
Timer/counters	3	3	3	3
Serial I/O port	Yes	Yes	Yes	Yes
PCA, Hardware Watchdog Timer	No	No	Yes	Yes
Code compatible with MCS® 51 Microcontrollers	Yes	Yes	Yes	Yes
Code compatible with MCS® 251 Microcontrollers	No	No	Yes	Yes
Keyboard control interface	Yes	Yes	No	No
General USB Features				
Complete ***Universal Serial Bus Specification***, Rev. 1.0 compatibility	Yes	Yes	Yes	Yes
On-chip USB transceivers	Yes	Yes	Yes	Yes
Automatic transmit/receive FIFO management	Yes	Yes	Yes	Yes
Time base (crystal/PLL)	12 MHz	12 MHz	12 MHz	12 MHz
USB rate (full speed)	12 Mbps	12 Mbps	12 Mbps	12 Mbps
Low-clock mode	Yes	Yes	Yes	Yes
Suspend/resume	Yes	Yes	Yes	Yes
USB interrupt vectors (hub, function, and suspend/resume)	Yes	Yes	Yes	Yes
Reset Separation	Yes	Yes	No	Yes
6 Endpoint Pair Option	No	No	No	Yes
USB Function Features				
Function endpoint pairs	3	3	4	4 (or 6)
Transmit/receive FIFO sizes:				
Endpoint 0	8 bytes	8 bytes	16 bytes	16 bytes
Endpoint 1	16 bytes	16 bytes	0-1024 bytes	0-1024 bytes (or 256 bytes)
Endpoint 2	8 bytes	8 bytes	16 bytes	16 (or 32 bytes)
Endpoint 3	—	—	16 bytes	16 (or 32 bytes)
Endpoint 4	—	—	—	(32 bytes)
Endpoint 5	—	—	—	(16 bytes)
USB Hub Features				
Internal downstream port	USB port 1	—	USB port 4	—
USB rate (full speed)	12 Mbps	—	12 Mbps	—

† INT2:0# are external interrupts.T2:0 are timer/counter interrupts.

Table 2-2. USB Peripheral Controller Feature Summary and Comparison (Continued)

	8*x*931H*x*	8*x*931A*x*	8*x*930H*x*	8*x*930A*x*
USB Hub Features (cont.)				
External downstream ports	4 (USB ports 2,3,4,5)	—	4 (USB ports 1,2,3,5)	—
USB rate (full speed/low speed)	12 Mbps/ 1.5 Mbps	—	12Mbps/ 1.5 Mbps	—
Hub endpoint 0: transmit/receive FIFOs	8 bytes	—	16 bytes	—
Hub endpoint 1: one transmit data buffer register	1 byte	—	1 byte	—
Core Microcontroller Features				
Architecture	MCS® 51 (Accumulator-based)	MCS® 51 (Accumulator-based)	MCS® 251 (Register-based)	MCS® 251 (Register-based)
Address spaces:				
Program memory Data memory	64 Kbytes 64 Kbytes	64 Kbytes 64 Kbytes	(Single 256-Kbyte address space)	(Single 256-Kbyte address space)
External bus (multiplexed)				
Address	16 bits	16 bits	16, 17, or 18 bits	16, 17, or 18 bits
Data	8 bits	8 bits	8 bits	8 bits
Number of Registers	8	8	40	40
Core interrupt vectors:				
INT0#, INT1#, T0, T1, T2†, and Serial I/O	Yes	Yes	Yes	Yes
Keyboard (INT2#)†	Yes	Yes	No	No
PCA	No	No	Yes	Yes
Parallel I/O ports	4	4	4	4
Powerdown and idle power-saving modes	Yes	Yes	Yes	Yes

† INT2:0# are external interrupts.T2:0 are timer/counter interrupts.

2.1.4 MCS® 51 Architecture Features

The 8*x*931 retains the basic features of and is code-compatible with the MCS 51 microcontroller. Features of the MCS 51 architecture are discussed in the following paragraphs and summarized in Table 2-2.

The MCS 51 architecture has separate program memory and data memory addresses spaces. A sixteen-bit address bus permits the 8*x*931 to address 64 Kbytes of program memory (up to 8 Kbytes of on-chip ROM and the remainder in external program memory) and 64 Kbytes of data memory (256 bytes of on-chip RAM and the remainder in external data memory). The general purpose registers (four banks of R0–R7) and the special function registers (SFRs) are located in the data memory address space. Refer to Chapter 3, “Address Spaces” for a description of the address modes.

The MCS 51 architecture has four 8-bit parallel I/O ports. The pins of these ports can be individually programmed to provide an external bus, to support special functions (keyboard scanning, timer/counter, interrupts, etc.), or for general I/O use. Ports P0 and P2 comprise a 16-line external

bus, which transmits a 16-bit address multiplexed with eight data bits. Ports P1 and P3 carry bus-control and peripheral signals. See Table B-7, "Signal Description," on page B-12.

The MCS 51 architecture has two power-saving modes. In idle mode, the CPU clock is stopped, while clocks to the peripherals continue to run. In global suspend mode (powerdown), the on-chip oscillator is stopped, and the chip enters a static state. In addition to idle and powerdown, the 8*x*931 has a special power-saving mode, low-clock mode, which it enters following a device reset. Refer to Chapter 14, "Special Operating Modes," for details on power-saving modes.

2.2 MICROCONTROLLER CORE

The microcontroller core contains the central processor unit (CPU), the clock and reset unit, the interrupt handler, the bus interface, and the peripheral interface (Figure 2-2).

2.2.1 CPU

The CPU contains the ALU, program counter, instruction decoder, data memory interface, general-purpose registers R0 – R7, and special function registers ACC, B, stack pointer (SP) and data pointer (DPTR).

The CPU executes the instruction set of the MCS 51 architecture. The instruction set is optimized for control operations. It provides fast addressing modes to facilitate byte operations on small data structures and extensive support for one-bit variables as a separate data type. For information on the instruction set refer to the *MCS® 51 Microcontroller Family User's Manual* and the *MCS® 51 Macro Assembler User's Guide*.

To provide fast context switching, the 8*x*931 implements registers R0 – R7 as four banks of eight registers, with the active bank selected by the program status word (PSW). The register banks occupy the lowest 32 bytes of RAM memory in the data memory address space.

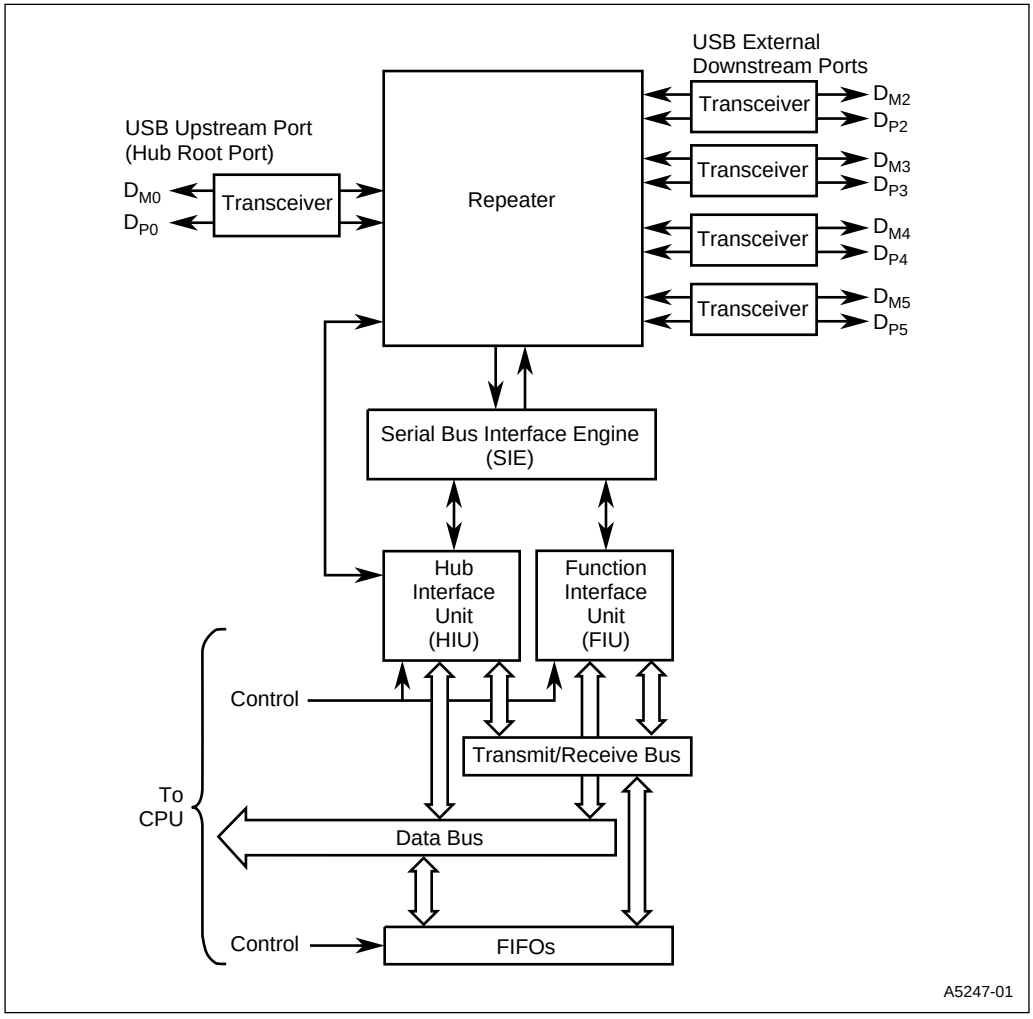


Figure 2-3. 8*x*931HA USB Module Block Diagram

2.2.2 Clock and Reset Unit

The waveform at XTAL1 is the 8*x*931 system clock. It can be supplied by an external source connected to XTAL1 or generated by an on-chip oscillator which has its resonant circuit (crystal or ceramic resonator) connected across XTAL1 and XTAL2. See Figure 2-4 for the 8*x*931 clock circuit.

2.2.2.1 State Time and Machine Cycles

The basic unit of time for 8*x*931 peripheral controller is the *state time* (or *state*). States are divided into two phases identified as *phase 1* and *phase 2*. The 8*x*931 machine cycle equals six states. See Figure 2-5 on page 2-10 for 8*x*931 clocking definitions. A specific time within a machine cycle is denoted by its state and phase. For example, when a timer/counter counts external events, the external input is sampled during state 5, phase 2 (S5P2) of every machine cycle.

The 8*x*931 executes single-cycle instructions in one machine cycle. With a 12 MHz crystal, F_{CLK} = 6 MHz and the duration of a machine cycle is 1 µs. Instruction execution begins in state 1 of a machine cycle when the opcode is latched into the instruction register. Execution is complete at the end of state 6. On-chip peripherals such as the timer/counter also operate on a machine cycle.

2.2.2.2 USB Operating Rate

Because of its hub capability, the 8*x*931HA (including the embedded function) always operates as a full-speed (12 Mbps) USB device. Root port data transfers with the host PC are always full speed. Data transfer rates on the external downstream ports are matched to the type of USB device attached — i.e., full speed or low speed (1.5 Mbps). For full speed operation, the PLL provides the 4X USB sampling rate.

The 8*x*931AA can operate as a full-speed (12Mbps) or low-speed (1.5Mbps) device.

2.2.2.3 Low-clock Mode

Low-clock mode is a special power-reduction mode in which the CPU and the on-chip peripherals operate at 3 MHz following device reset. To exit low-clock mode, clear the LC bit in the PCON register. During low-clock mode, F_{CLK} = 3 MHz, so the timing definitions in Figure 2-5 do not apply to the CPU and on-chip peripherals. Low-clock mode does not affect the USB rate. Also see Chapter 14, "Special Operating Modes".

2.2.2.4 Reset Unit

The reset unit resets the 8*x*931 to a known state. A chip reset is initiated by asserting the RST pin or by a USB-initiated reset. For information on resets refer to Chapter 13, "Minimum Hardware Setup".

NOTE

The 8*x*931 can be programmed so a USB-initiated reset does not cause a chip reset. For additional information, see "USB Reset Separation" on page 5-17.

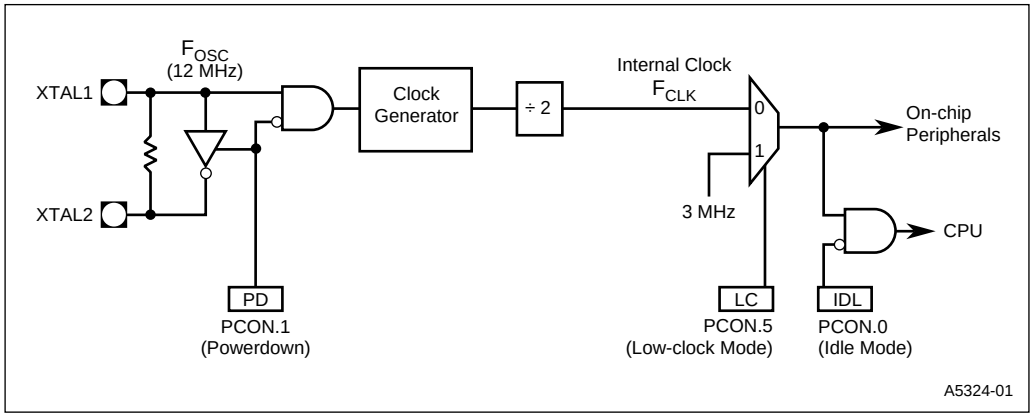


Figure 2-4. 8*x*931 Clock Circuit

Table 2-3. 8*x*931HA Operating Frequency

PLLSEL	XTAL1 Frequency (F_{osc})	USB Rate (1)	Internal Frequency (F_{CLK}) (2)	XTAL1 Clocks per State (T_{osc}/state) (3)	Comments
0 (4)	–	–	–	–	–
1	12 MHz	12 Mbps (Full Speed)	6 MHz (3)	2	PLL On

NOTES:
1. The sampling rate is 4 times the USB rate.
2. The internal frequency, $F_{CLK} = 1/T_{CLK}$, is the clock signal distributed to the CPU and the on-chip peripherals,
3. Following device reset, the CPU and on-chip peripherals operate in low-clock mode (F_{CLK} = 3 MHz) until the LC bit in the PCON register is cleared. In low clock mode, there are four T_{OSC} periods per state. Low-clock mode does not affect the USB rate.
4. PLLSEL = 0 is used during factory test only.

Table 2-4. 8*x*931AA Operating Frequencies

PLLSEL Pin	FSSEL Pin	LC Bit (1)	XTAL1 Frequency (MHz)	USB Rate (FS/LS) (2)	Core Frequency F_{CLK} (Mhz)	Comment
0	0	0	6	LS	3	PLL Off
0	0	1	6	LS	3	PLL Off
1	0	0	12	LS	6	PLL Off
1	0	1	12	LS	3	PLL Off
1	1	0	12	FS	6	PLL On
1	1	1	12	FS	3	PLL On

NOTES:
1. Reset and power up routines set the LC bit in PCON to put the 8*x*931AA in low-clock mode (core frequency = 3 MHz) for lower I_{CC} prior to device enumeration. Following completion of device enumeration, firmware should clear the LC bit to exit the low-clock mode. The user may switch the core frequency back and forth at any time, as needed.
2. USB rates: Low speed = 1.5 Mbps; Full speed = 12 Mbps. The USB sample rate is 4X the USB rate.

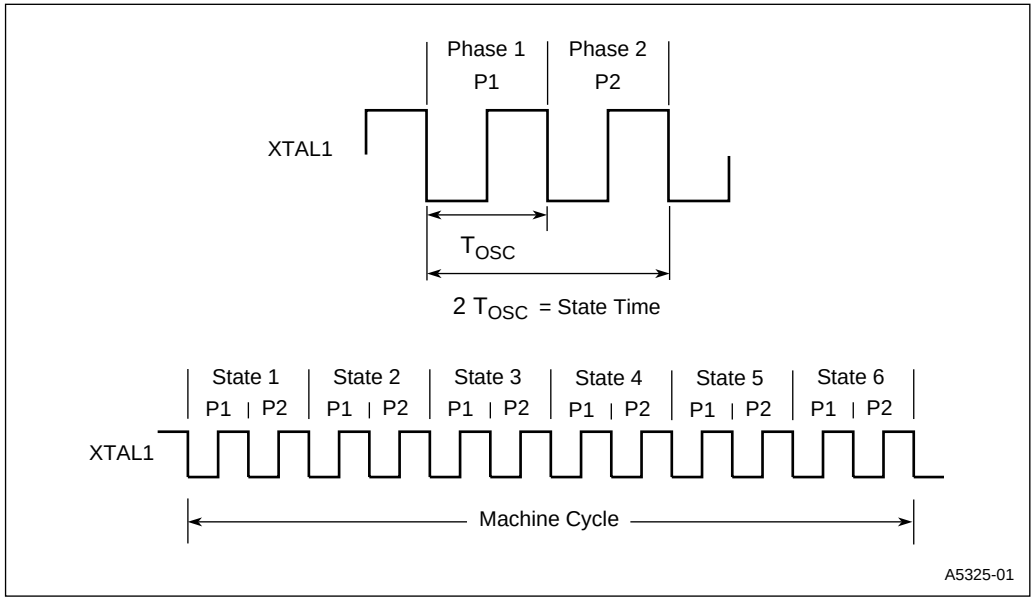


Figure 2-5. 8*x*931 Clocking Definitions

2.2.3 Interrupt Handler

The interrupt handler processes interrupt requests from maskable interrupt sources (USB module, keyboard control interface, timer/counters and external). When the interrupt handler grants an interrupt request, the CPU discontinues the normal sequence of instruction execution and branches to a routine that services the interrupt source. You can enable or disable the interrupts individually and you can assign one of four priority levels to each interrupt. Refer to Chapter 5, “Interrupt System” for a detailed description.

2.3 8*x*931 MEMORY

The 8*x*931 has separate 64-Kbyte program memory and data memory address spaces. A sixteen-bit address bus permits the 8*x*931 to address 64 Kbytes of program memory (up to 8 Kbytes of on-chip ROM and the remainder in external program memory) and 64 Kbytes of data memory (256 bytes of on-chip RAM and the remainder in external data memory). See Table 2-1 for 8*x*931 memory options.

The 8*x*931 is available with 8 Kbytes of on-chip ROM memory located at the lowest addresses of program memory, or without ROM. Program memory is read only. Following chip reset, the first instruction is fetched from location 0000H in program memory. For ROM devices, this will be from on-chip program memory and EA# should be tied to V_{CC}. For devices without on-chip ROM, all instruction fetches are from external memory and EA# should be tied to ground.

The 8*x*931 has 256 bytes of on-chip RAM located at the lowest addresses of the data memory. Data memory locations can be accessed with direct and indirect addressing. Sixteen of these locations (20H–2FH) are bit addressable. The general purpose registers (four banks of R0–R7) reside at data memory locations 00H-1FH.

Special functions registers (SFRs) are also located in the data memory space at locations 80H – FFH. SFRs are accessed by direct addressing, while general purpose RAM in this address range is accessed by indirect addressing.

2.4 UNIVERSAL SERIAL BUS MODULE

The 8*x*931HA USB module operates in conjunction with the CPU to provide both USB function and USB hub capabilities. The block diagram in Figure 2-3 on page 2-7 shows the main components of the 8*x*931HA USB module and how they interface with the CPU.

The hub provides the electrical interface between the host PC and downstream devices connected to the USB. The repeater and the hub interface, which is made up of the Serial Bus Interface Unit (SIE), the Hub Interface Unit (HIU), and the hub FIFOs, provide the hub capability (Figure 2-3). The USB function interface manages communications between the host PC and the embedded function. The function interface is made up of the SIE, the function interface unit (FIU), the function FIFOs (Figure 2-3).

The 8*x*931HA USB module communicates with the host PC by means of an upstream data port (USB port 0). The USB module communicates with devices attached to the USB by means of an internal downstream port (USB port 1) and the four external downstream ports (8*x*931HA only). See Figure 2-3 and Figure 7-1 on page 7-2 for a hub block diagram. For USB port descriptions and pin assignments, see Appendix B. The external USB ports are differential data ports that are

fully compliant with the *Universal Serial Bus Specification*. The 8*x*931HA provides on-chip transceivers for each external USB port and ganged-switched port power on the external downstream ports.

A complete description of the USB can be found in *Universal Serial Bus Specification*. For a description of the transceiver see the "Driver Characteristics" and "Receiver Characteristics" sections of the "Electrical" chapter of the *Universal Serial Bus Specification*. For electrical characteristics and data signal timing, see the "Bus Timing/Electrical Characteristics" and "Timing Diagram" sections of the same chapter.

NOTE

The 8*x*931AA microprocessor does not support a hub interface. Specific details of the 8*x*931AA are covered in Appendix E, "8x931AA Design Considerations".

2.4.1 USB Operation

Operation of the USB module is controlled through the use of special function registers (SFRs). SFRs associated with the USB module are described in Chapter 5, "USB Function," Chapter 6, "USB Hub," and Chapter 4, "Interrupt System." Register definition tables in these chapters describe register usage and define the register bits. The register definition tables also appear in Appendix C in alphabetical order. A memory map of the 8*x*931 SFRs is presented in Chapter 3, "Address Spaces" and Table C-1 on page C-2.

Data transfers with the host are made to/from endpoint pairs (EPPs) on the USB module. The 8*x*931HA provides three function endpoint pairs, a hub control endpoint pair, and a transmit-only hub status change endpoint. Table 2-5 lists the hub and function endpoint pairs available on the 8*x*931 along with the associated transmit and receive FIFO data buffers. For any given data transfer operation, the EPINDEX register specifies the endpoint pair involved and the HPINDEX register specifies the downstream port.

Table 2-5. Endpoint Pairs for 8*x*931

Device	EPINDEX	Endpoint Pair	Transmit FIFOs	Receive FIFOs	USB Data Transfer Types
8*x*931AA/HA	0000 0000	Function Endpoint 0	8 bytes	8 bytes	Control
8*x*931AA/HA	0000 0001	Function Endpoint 1	16 bytes	16 bytes	Control, Interrupt, Bulk, Isochronous
8*x*931AA/HA	0000 0010	Function Endpoint 2	8 bytes	8 bytes	Control, Interrupt, Bulk
8*x*931HA	1000 0000	Hub Endpoint 0	8 bytes	8 bytes	Control
8*x*931HA	1000 0001	Hub Endpoint 1 (1)	N/A	N/A	Interrupt

NOTES:
1. Hub endpoint 1 assembles status-change information in a buffer register (TXDAT) and transmits it to the host PC. Hub endpoint 1 does not require FIFOs.

The CPU runs the firmware associated with the operation of the hub and the function interface. It also reads the receive FIFOs, loads the transmit FIFOs, and decodes and executes USB requests for the hub. Control transaction stages are tracked by firmware.

2.4.2 Hub Interface

Hub operation is implemented by reading and writing the hub SFRs. The repeater, the SIE, the hub interface unit (HIU), and the hub FIFOs provide the hub capability. The hub interface has two endpoint pairs. Hub endpoint 0 supports only control data transfers. Hub endpoint 1 is used to transmit hub status change information to the host PC.

8*x*931HA USB hub operations fall into two categories: hub repeater operations and hub controller operations. The hub controller is split among four modules: the serial bus interface engine, the hub interface unit, the hub endpoint 0 transmit and receive FIFOs, and the 8*x*931HA CPU. See Chapter 7, “USB Hub”. The following subsections discuss the role of each module.

2.4.3 Hub Repeater

The repeater is the connectivity manager for the 8*x*931HA. It detects the connection or disconnection of devices on the external downstream ports and manages the upstream/downstream connectivity for data packets. It keeps track of hub port status, manages connectivity, and performs power management for external downstream ports. The repeater supports both full-speed (12 Mbps) and low-speed (1.5 Mbps) data traffic. The repeater also controls bus fault detection and recovery. Downstream port control is managed primarily by the HIU.

2.4.4 Serial Bus Interface Engine (SIE)

The SIE is the USB communication protocol interpreter. It places data on and accepts data from the bus. On the 8*x*931HA, the hub interface and the function interface share the SIE. This is possible because the host communicates with only one USB device during any given transaction.

The internal downstream port is permanently attached to the SIE. The SIE provides serial-to-parallel conversion for data transfers from the host and parallel-to-serial conversion for data transfers to the host. For additional information on the SIE, see “SIE Details” on page 6-34.

2.4.5 Hub Interface Unit (HIU)

The HIU uses special function registers (SFRs) to control the operation of the hub and to maintain the status of the hub and its downstream ports. Control SFRs are set by firmware in response to USB requests. Status SFRs are set by the repeater hardware. Refer to Chapter 7, “USB Hub” and Chapter 8, “USB Programming Models” for a discussion on the use of the HIU SFRs.

2.4.6 Hub FIFOs

Hub FIFOs operate in the same manner as the function interface FIFOs. See Chapter 6, “USB Function” for a detailed description of their operation. Hub endpoint 0 handles only control data transfers. It is implemented with 8-byte transmit and receive FIFO data buffers. The maximum packet size for hub control data transfers is eight bytes. Data received from the USB for endpoint

0 is stored in the receive FIFO for reading by firmware. Data to be sent to the host from hub endpoint 0 is loaded into the transmit FIFO.

Hub endpoint 1 transmits single-byte interrupt tokens to the host and does not have FIFO data buffers.

2.5 ON-CHIP PERIPHERALS

The on-chip peripherals reside outside the microcontroller core. They perform specialized functions in hardware. Firmware controls the peripherals via their special function registers (SFRs). The 8*x*931 has two peripherals: the timer/counter unit and the serial I/O port.

2.5.1 Timer/Counters

The timer/counter unit has three programmable 16-bit timer/counters. They can be clocked by the divided-down system clock or an external timebase (timer operation) or by external events (counter operation). They can be set up as 8-bit, 13-bit, or 16-bit timer/counters. You can program them for special applications, such as capturing the time of an event on an external pin, outputting a programmable clock signal on an external pin, or generating a baud rate for the serial I/O port. Timer/counter events generate interrupt requests.

2.5.2 Serial I/O Port

The serial I/O port provides one synchronous and three asynchronous communication modes. The synchronous mode (mode 0) is half-duplex: the serial port outputs a clock signal on one pin and transmits or receives data on another pin.

The asynchronous modes (modes 1–3) are full-duplex (i.e., the port can send and receive simultaneously). Mode 1 uses a serial frame of 10 bits: a start bit, 8 data bits, and a stop bit. The baud rate is generated by the overflow of timer 1 or timer 2. Modes 2 and 3 use a serial frame of 11 bits: a start bit, eight data bits, a programmable ninth data bit, and a stop bit. The ninth bit can be used for parity checking or to specify that the frame contains an address and data. In mode 2, you can use a baud rate of 1/32 or 1/64 of the oscillator frequency. In mode 3, you can use the overflow from timer 1 or timer 2 to determine the baud rate.

In its asynchronous modes (modes 1–3) the serial port can operate as a slave in an environment where multiple slaves share a single serial line. It can accept a message intended for itself or a message that is being broadcast to all of the slaves, and it can ignore a message sent to another slave.

2.6 OPERATING CONDITIONS

The 8*x*931 is designed for a commercial operating environment and to accommodate the operating rates of the USB interface. For detailed specifications, refer to the current *8x*931AA/*8x*931HA *Universal Serial Bus Peripheral Controller* datasheet (order number: 273108-001). For USB module operating rates, see "Clock and Reset Unit" on page 2-7.

3

Address Spaces

CHAPTER 3
ADDRESS SPACES

3.1 MEMORY ORGANIZATION IN 8*x*931 DEVICES

3.1.1 Logical Separation of Program and Data Memory

8*x*931 devices have separate address spaces for Program and Data Memory, as shown in Figure 3-2. The logical separation of Program and Data Memory allows the Data Memory to be accessed by 8-bit addresses, which can be more quickly stored and manipulated by an 8-bit CPU. Nevertheless, 16-bit Data Memory addresses can also be generated through the DPTR register.

Program Memory can only be read, not written to. There can be up to 64K bytes of Program Memory. In the ROM version, the lowest 8K bytes of Program Memory are provided on-chip. Refer to Table 2-1 on page 2-3 for the amount of on-chip ROM on each device. In the ROMless versions, all Program Memory is external. The read strobe for external Program Memory is the signal PSEN# (Program Store Enable).

Data Memory occupies a separate address space from Program Memory. Up to 64K bytes of external RAM can be addressed in the external Data Memory space. The CPU generates read and write signals, RD# and WR#, as needed during external Data Memory accesses.

External Program Memory and external Data Memory may be combined if desired by applying the RD# and PSEN# signals to the inputs of an AND gate and using the output of the gate as the read strobe to the external Program/Data memory.

3.1.2 Program Memory

Figure 3-1 shows a map of the lower part of the Program Memory. After reset, the CPU begin execution from location 0000H.

As shown in Figure 3-1, each interrupt is assigned a fixed location in Program Memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose Program Memory.

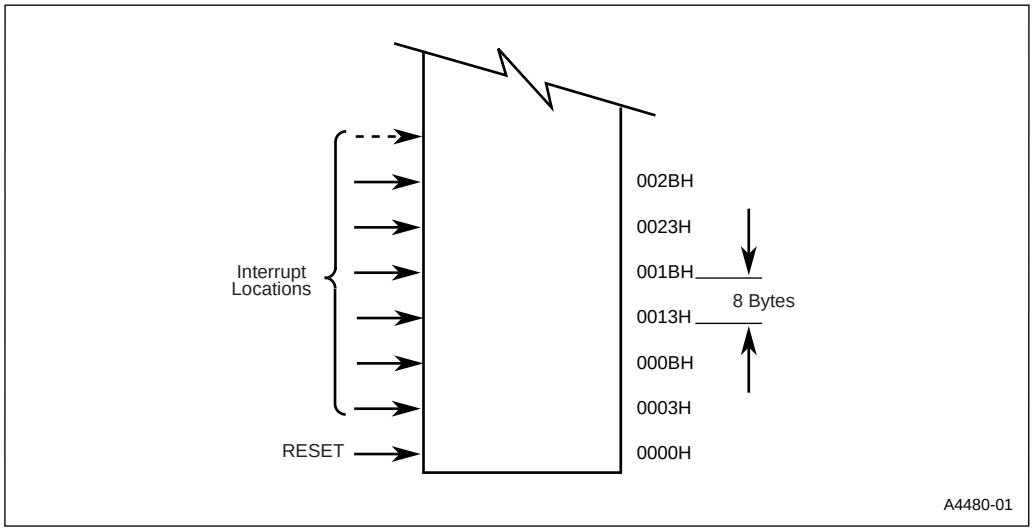


Figure 3-1. MCS® 51 Program Memory

The interrupt service locations are spaced at 8-byte intervals: 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

The lowest 8K bytes of Program Memory can be either in the on-chip ROM or in an external ROM. This selection is made by strapping the EA# (External Access) pin to either V_{CC} or V_{SS}.

In the 8K byte ROM devices, EA# = V_{CC} selects addresses 0000H through 1FFFH to be internal, and addresses 2000H through FFFFH to be external.

If the EA# pin is strapped to V_{SS}, then all program fetches are directed to external ROM. The ROMless parts must have this pin externally strapped to V_{SS} to enable them to execute properly.

The read strobe to external ROM, PSEN#, is used for all external program fetches. PSEN# is not activated for internal program fetches.

Program Memory addresses are always 16 bits wide, even though the actual amount of Program Memory used may be less than 64K bytes. External program execution sacrifices two of the 8-bit ports, P0and P2, to the function of addressing the Program Memory.

3.1.3 Data Memory

Figure 3-2 shows the internal and external Data Memory spaces available to the 8*x*931 user.

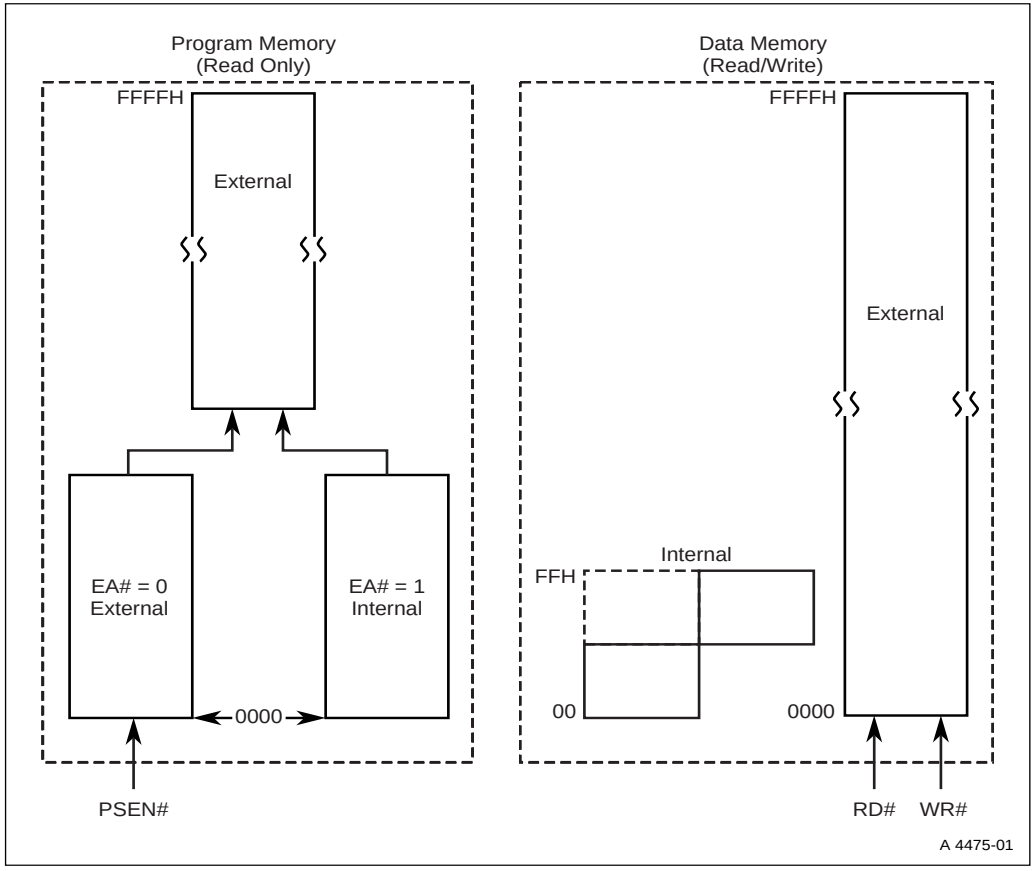


Figure 3-2. 8*x*931 Memory Structure

Internal Data Memory is mapped in Figure 3-3. The memory space is shown divided into three blocks, which are generally referred to as the Lower 128, the Upper 128, and SFR space. Internal Data Memory addresses are always one byte wide, which implies an address space of only 256 bytes. However, the addressing modes for internal RAM can in fact accommodate 384 bytes, using a simple trick. Direct addresses higher than 7FH access one memory space, and indirect addresses higher than 7FH access a different memory space. Thus Figure 3-3 shows the Upper 128 and SFR space occupying the same block of addresses, 80H through FFH, although they are physically separate entities.

The Lower 128 bytes of RAM are present in all MCS® 51 devices as mapped in Figure 3-4. The lowest 32 bytes are grouped into 4 banks of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) select which register bank is

in use. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing.

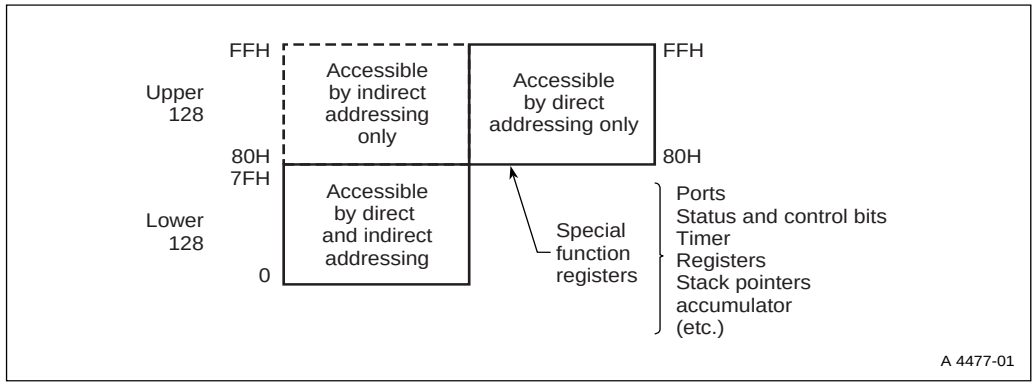


Figure 3-3. Internal Data Memory

The next 16 bytes above the register banks form a block of bit-addressable memory space. The MCS® 51 instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH. All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing. The Upper 128 can only be accessed by indirect addressing.

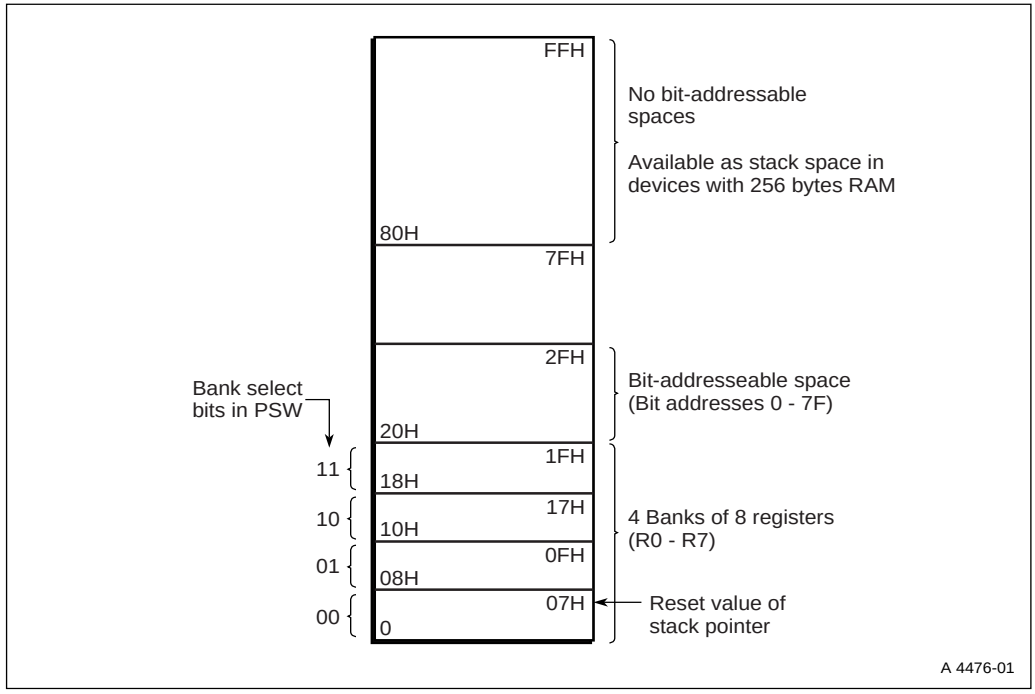


Figure 3-4. Upper and Lower 128 Bytes of Internal RAM

3.2 SPECIAL FUNCTION REGISTERS (SFRS)

The special function registers (SFRs) reside in the microcontroller core, the USB module, and the on-chip peripherals. Memory maps showing the location of all the 8*x*931HA SFRs are presented in Appendix C, “Registers”. 8*x*931AA SFRs are shown in Appendix E, “8x931AA Design Considerations”. The content of each register following device reset is given. An “x” indicates the bit value following reset is indeterminate.

Figure 3-5 gives a brief look at the Special Function Register (SFR) space. SFRs include the Port latches, timers, peripheral controls, etc. These registers can only be accessed by direct addressing.

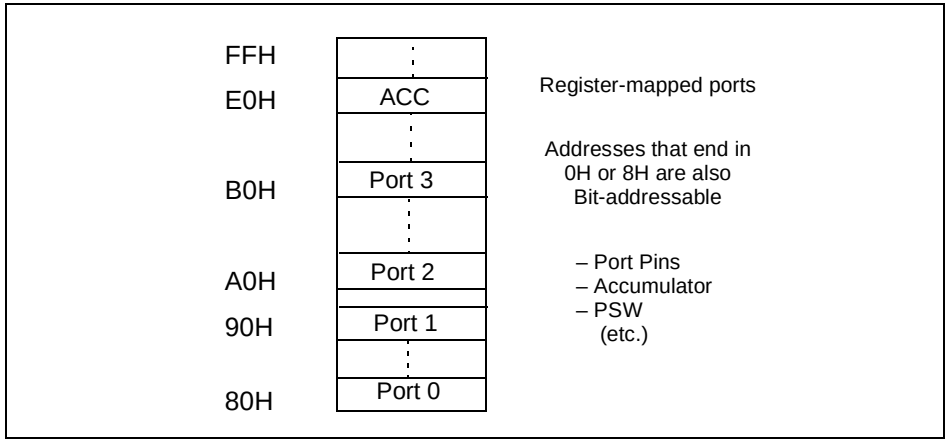


Figure 3-5. SFR Space

Blank locations in the SFR map are unimplemented, i.e., no register exists. If an instruction attempts to write to an unimplemented SFR location, the instruction executes, but nothing is actually written. If an unimplemented SFR location is read, it returns an unspecified value.

Endpoint-indexed SFRs are implemented as banks of registers. There is a set or bank of registers for each endpoint pair. Endpoint-indexed SFRs are accessed by means of the SFR address and an index value. The EPINDEX register specifies hub/function and the endpoint number (which serves as the index value). See “Endpoint-indexed SFRs” on page 6-5 and “Hub Endpoint Indexing Using EPINDEX” on page 7-11.

Port-indexed SFRs (HPCON, HPSC, and HPSTST) are implemented in a similar manner. There is a set or bank of these registers for each USB downstream port. Port-indexed SFRs are accessed by means of the SFR address and an index value. The HPINDEX register contains the port number which serves as the index value. See “Hub Port Indexing Using HPINDEX” on page 7-23.

Individual SFRs are presented in alphabetical order in Appendix C. Tables listing the SFRs by functional category are also given in Appendix C.

4

Programming Considerations

CHAPTER 4
PROGRAMMING CONSIDERATIONS

The instruction set for the 8*x*931 supports the instruction set for the MCS® 51 architecture. This chapter describes the addressing modes and summarizes the instruction set, which is divided into data instructions, bit instructions, and control instructions. The program status word register is also described. Appendix A, "Instruction Set Reference" contains an opcode map and a detailed description of each instruction.

4.1 THE MCS® 51 INSTRUCTION SET

All members of the MCS 51 family execute the same instruction set. The MCS-51 instruction set is optimized for 8-bit control applications. It provides a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. The instruction set provides extensive support for one-bit variables as a separate data type, allowing direct bit manipulation in control and logic systems that require Boolean processing.

An overview of the MCS 51 instruction set is presented below, with a brief description of how certain instructions might be used. References to "the assembler" in this discussion are to Intel's MCS 51 Macro Assembler, ASM51. More detailed information on the instruction set can be found in the MCS 51 Macro Assembler User's Guide (Order No. 9800937 for ISIS Systems, Order No. 122752 for DOS Systems).

4.1.1 Program Status Word

The Program Status Word (PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown in Figure 4-1, resides in SFR space. It contains the Carry bit, the Auxiliary Carry (for BCD operations), the two register bank select bits, the Overflow flag, a Parity bit, and two user-definable status flags.

The Carry bit, other than serving the functions of a Carry bit in arithmetic operations, also serves as the "Accumulator" for a number of Boolean operations.

The bits RS0 and RSl are used to select one of the four register banks shown in Figure 3-4 on page 3-4. A number of instructions refer to these RAM locations as R0 through R7. The selection of which of the four banks is being referred to is made on the basis of the bits RS0 and RS1 at execution time.

The Parity bit reflects the number of 1s in the Accumulator P = 1 if the Accumulator contains an odd number of 1s, and P = 0 if the Accumulator contains an even number of 1s. Thus the number of 1s in the Accumulator plus P is always even.

Two bits in the PSW are uncommitted and may be used as general purpose status flags.

Table 4-1 shows the effects of instructions on the PSW flags.

PSW Address: S:D0H
Reset State: 0000 0000B

7							0
CY	AC	F0	RS1	RS0	OV	UD	P

Bit Number	Bit Mnemonic	Function
7	CY	Carry Flag: The carry flag is set by an addition instruction (ADD, ADDC) if there is a carry out of the MSB. It is set by a subtraction (SUB, SUBB) or compare (CMP) if a borrow is needed for the MSB. The carry flag is also affected by logical bit, bit move, multiply, decimal adjust, and some rotate and shift instructions (see Table 4-1).
6	AC	Auxiliary Carry Flag: The auxiliary carry flag is affected only by instructions that address 8-bit operands. The AC flag is set if an arithmetic instruction with an 8-bit operand produces a carry out of bit 3 (from addition) or a borrow into bit 3 (from subtraction). Otherwise, it is cleared. This flag is useful for BCD arithmetic (see Table 4-1).
5	F0	Flag 0: This general-purpose flag is available to the user.
4:3	RS1:0	Register Bank Select Bits 1 and 0: These bits select the memory locations that comprise the active bank of the register file (registers R0–R7). RS1 RS0 Bank Address 0 0 0 00H–07H 0 1 1 08H–0FH 1 0 2 10H–17H 1 1 3 18H–1FH
2	OV	Overflow Flag: This bit is set if an addition or subtraction of signed variables results in an overflow error (i.e., if the magnitude of the sum or difference is too great for the seven LSBs in 2's-complement representation). The overflow flag is also set if a multiplication product overflows one byte or if a division by zero is attempted.
1	UD	User-definable Flag: This general-purpose flag is available to the user.
0	P	Parity Bit: This bit indicates the parity of the accumulator. It is set if an odd number of bits in the accumulator are set. Otherwise, it is cleared. Not all instructions update the parity bit. The parity bit is set or cleared by instructions that change the contents of the accumulator (ACC).

Figure 4-1. Program Status Word Register

Table 4-1. The Effects of Instructions on the PSW Flags

Instruction Type	Instruction	Flags Affected (1), (4)		
		CY	OV	AC (2)
Arithmetic	ADD, ADDC, SUBB, CMP	X	X	X
	INC, DEC			
	MUL, DIV (3)	0	X	
	DA	X		
Logical	ANL, ORL, XRL, CLR A, CPL A, RL, RR, SWAP			
	RLC, RRC	X		
Program Control	CJNE	X		
	DJNZ			

NOTES:
1. X = the flag can be affected by the instruction.
   0 = the flag is cleared by the instruction.
2. The AC flag is affected only by operations on 8-bit operands.
3. If the divisor is zero, the OV flag is set, and the other bits are meaningless.
4. The parity bit (PSW.0) is set or cleared by instructions that change the contents of the accumulator (ACC).

4.1.2 Addressing Modes

The addressing modes in the MCS 51 instruction set are as follows.

4.1.2.1 DIRECT ADDRESSING

In direct addressing the operand is specified by an 8-bit address field in the instruction. Only internal Data RAM and SFRs can be directly addressed.

4.1.2.2 INDIRECT ADDRESSING

In indirect addressing the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed. The address register for 8-bit addresses can be R0 or R1 of the selected register bank, or the Stack Pointer. The address register for 16-bit addresses can only be the 16-bit "data pointer" register, DPTR.

4.1.2.3 REGISTER INSTRUCTIONS

The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code efficient, since this mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank is accessed. One of four banks is selected at execution time by the two bank select bits in the PSW.

4.1.2.4 REGISTER-SPECIFIC INSTRUCTIONS

Some instructions are specific to a certain register. For example, some instructions always operate on the Accumulator, or Data Pointer, etc., so no address byte is needed to point to it. The opcode itself does this. Instructions that refer to the Accumulator as A, assemble as accumulator-specific opcodes.

4.1.2.5 IMMEDIATE CONSTANTS

The value of a constant can follow the opcode in Program Memory. For example,

```
MOV A, # 100
```

loads the Accumulator with the decimal number 100. The same number could be specified in hex digits as 64H.

4.1.2.6 INDEXED ADDRESSING

Only Program Memory can be accessed with indexed addressing, and it can only be read. This addressing mode is intended for reading look-up tables in Program Memory. A 16-bit base register (either DPTR or the Program Counter) points to the base of the table, and the Accumulator is setup with the table entry number. The address of the table entry in Program Memory is formed by adding the Accumulator data to the base pointer.

Another type of indexed addressing is used in the "case jump" instruction. In this case the destination address of a jump instruction is computed as the sum of the base pointer and the Accumulator data.

Table 4-2. Addressing Modes for Data Instructions in the MCS® 51 Architecture

Mode	Address Range of Operand	Assembly Language Reference	Comments
Register	00H–1FH	R0–R7 (Bank selected by PSW)	
Immediate	Operand in Instruction	#data = #00H–#FFH	
Direct	00H–7FH	dir8 = 00H–7FH	On-chip RAM
	SFRs	dir8 = 80H–FFH or SFR mnemonic.	SFR address
Indirect	00H–FFH	@R0, @R1	Accesses on-chip RAM or the lowest 256 bytes of external data memory (MOVX).
	0000H–FFFFH	@DPTR, @A+DPTR	Accesses external data memory (MOVX).
	0000H–FFFFH	@A+DPTR, @A+PC	Accesses code memory (MOVC).

4.1.3 Arithmetic Instructions

The menu of arithmetic instructions is listed in Table 4-3. The table indicates the addressing modes that can be used with each instruction to access the <byte> operand. For example, the ADD A, <byte> instruction can be written as:

ADD A,7FH (direct addressing)

ADD A,@RO (indirect addressing)

ADD A,R7 (register addressing)

ADD A, # 127 (immediate constant)

The execution times listed in Table 4-3 assume a 12 MHz clock frequency. All of the arithmetic instructions execute in 1µs except the INC DPTR instruction, which takes 2 µs, and the Multiply and Divide instructions, which take 4 µs.

NOTE

Any byte in the internal Data Memory space can be incremented or decremented without going through the Accumulator.

One of the INC instructions operates on the 16-bit Data Pointer. The Data Pointer is used to generate 16-bit addresses for external memory, so being able to increment it in one 16-bit operation is a useful feature.

The MUL AB instruction multiplies the Accumulator by the data in the B register and puts the 16-bit product into the concatenated B and Accumulator registers.

Table 4-3. List of MCS® 51 Arithmetic Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (µs)
		Dir	Ind	Reg	Imm	
ADD A, <byte>	A = A + <byte>	X	X	X	X	1
ADDOA, <byte>	A= A+< byte>+C	X	X	X	X	1
SUBB A, <byte>	A= A–<byte>-C	X	X	X	X	1
INC A	A=A+1	Accumulator only				1
INC . <byte>	<byte> =<byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A= A- 1	Accumulator only				1
DEC <byte>	<byte> = <byte> – 1	X	X	X		1
MUL AB	B:A=Bx A	ACC and B only				4
DIV AB	A = Int [A/B] B = MOd[A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

The DIV AB instruction divides the Accumulator by the data in the B register and leaves the 8-bit quotient in the Accumulator, and the 8-bit remainder in the B register.

Oddly enough, DIV AB finds less use in arithmetic "divide" routines than in radix conversions and programmable shift operations. An example of the use of DIV AB in a radix conversion will be given later. In shift operations, dividing a number by 2^n shifts its n bits to the right. Using DIV AS to perform the division completes the shift in 4 µs and leaves the B register holding the bits that were shifted out.

The DA A instruction is for BCD arithmetic operations. In BCD arithmetic, ADD and ADDC instructions should always be followed by a DA A operation, to ensure that the result is also in BCD.

NOTE

DA A will not convert a binary number to BCD. The DA A operation produces a meaningful result only as the second step in the addition of two BCD bytes.

Table 4-4. List of MCS® 51 Logical Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (µs)
		Dir	Ind	Reg	Imm	
ANL A, <byte>	A = A .AND. <byte>	X	X	X	X	1
ANL <byte>, A	<byte> = <byte> .AND. A	X				1
ANL <byte>, #data	<byte> = <byte> .AND. #data	X				2
ORL A, < byte>	A = A.OR. <byte>	X	X	X	X	1
ORL <bvte>,A	<byte> = <byte> .OR. A	X				1
ORL <byte>, #data	<byte> = <byte> .OR. #data	X				2
XRL A,< byte>	A = A .XOR. <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> .XOR. A	X				1
XRL <byte>, #data	<byte> = <byte> .XOR. #data	X				2
CLR A	A=00H	Accumulator only				1
CPL A	A = .NOT.A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

4.1.4 Logical Instructions

Table 4-4 shows the list of MCS 51 logical instructions. The instructions that perform Boolean operations (AND, OR, Exclusive OR, NOT) on bytes perform the operation on a bit-by-bit basis. That is, if the Accumulator contains 00110101B and <byte> contains 01010011B, then

ANL A, <byte>

will leave the Accumulator holding 00010001B.

The addressing modes that can be used to access the <byte> operand are listed in Table 4-4. Thus, the ANL A, <byte> instruction may take any of the forms:

ANL A,7FH (direct addressing)

ANL A,@Rl (indirect addressing)

ANL A,R6 (register addressing)

ANL A, # 53H (immediate constant)

All of the logical instructions that are Accumulator-specific execute in l µs (using a 12 MHz clock). The others take 2 µs.

Note that Boolean operations can be performed on any byte in the lower 128 internal Data Memory space or the SFR space using direct addressing, without having to use the Accumulator. The XRL <byte >, #data instruction, for example offers a quick and easy way to invert port bits, as in

XRL Pl,#0FFH

If the operation is in response to an interrupt, not using the Accumulator saves the time and effort to stack it in the service routine. The Rotate instructions (RL & RLC A, etc.) shift the Accumulator 1 bit to the left or right. For a left rotation, the MSB rolls into the LSB position. For a right rotation, the LSB rolls into the MSB position.

The SWAP A instruction interchanges the high and low nibbles within the Accumulator. This is a useful operation in BCD manipulations. For example, if the Accumulator contains a binary number which is known to be less than 100, it can be quickly converted to BCD by the following code:

```
MOV B, # 10
DIV AB
SWAP A
ADD A,B
```

Dividing the number by 10 leaves the tens digit in the low nibble of the Accumulator, and the ones digit in the B register. The SWAP and ADD instructions move the tens digit to the high nibble of the Accumulator, and the ones digit to the low nibble.

4.1.5 Data Transfers

4.1.5.1 Internal RAM

Table 4-5 shows the menu of instructions that are available for moving data around within the internal memory spaces, and the addressing modes that can be used with each one. With a 12 MHz clock, all of these instructions execute in either 1 or 2 µs.

The MOV <dest>, <src> instruction allows data to be transferred between any two internal RAM or SFR locations without going through the Accumulator. Remember the Upper 128 bytes of data RAM can be accessed only by indirect addressing, and SFR space only by direct addressing.

Table 4-5. List of MCS® 51 Data Transfer Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (µs)
		Dir	Ind	Reg	Imm	
MOV A, <src>	A = <src>	X	X	X	X	1
MOV <dest> ,A	<dest> = A	X	X	X		1
MOV <dest>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	dptr = 16-bit immediate constant				X	2
PUSH <src>	INC SP:MOV"@SP",<src>	X				2
POP <dest>	MOV <dest>, "@SP"; DEC SP	X				2
XCH A, <byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A, @Ri	ACC and @Ri exchange low nibbles		X			1

NOTE

In all MCS 51 devices, the stack resides in on-chip RAM and grows upwards.

The PUSH instruction first increments the Stack Pointer (SP), then copies the byte into the stack. PUSH and POP use only direct addressing to identify the byte being saved or restored, but the stack itself is accessed by indirect addressing using the SP register. This means the stack can go into the Upper 128, if they are implemented, but not into SFR space.

In devices that do not implement the Upper 128, if the SP points to the Upper 128, PUSHed bytes are lost, and POPped bytes are indeterminate.

The Data Transfer instructions include a 16-bit MOV that can be used to initialize the Data Pointer (DPTR) for look-up tables in Program Memory, or for 16-bit external Data Memory accesses.

The XCH A, <byte> instruction causes the Accumulator and addressed byte to exchange data. The XCHD A, @Ri instruction is similar, but only the low nibbles are involved in the exchange.

To see how XCH and XCHD can be used to facilitate data manipulations, consider first the problem of shifting an 8-digit BCD number two digits to the right. Figure 4-2 shows how this can be done using direct MOVs, and for comparison how it can be done using XCH instructions. To aid in understanding how the code works, the contents of the registers that are holding the BCD number and the content of the Accumulator are shown alongside each instruction to indicate their status after the instruction has been executed.

After the routine has been executed, the Accumulator contains the two digits that were shifted out on the right. Doing the routine with direct MOVs uses 14 code bytes and 9 µs of execution time (assuming a 12 MHz clock). The same operation with XCHs uses less code and executes almost twice as fast.

To right-shift by an odd number of digits, a one-digit shift must be executed. Figure 4-3 shows a sample of code that will right-shift a BCD number one digit, using the XCHD instruction. Again, the contents of the registers holding the number and of the Accumulator are shown alongside each instruction.

		2A	2B	2C	2D	2E	ACC
MOV	A,2EH	00	12	34	56	78	78
MOV	2EH,2DH	00	12	34	56	56	78
MOV	2DH,2CH	00	12	34	34	56	78
MOV	2CH,2BH	00	12	12	34	56	78
MOV	2BH,#0	00	00	12	34	56	78

(a) Using direct MOVs: 14 bytes, 9 µs

		2A	2B	2C	2D	2E	ACC
CLR	A	00	12	34	56	78	00
XCH	A,2BH	00	00	34	56	78	12
XCH	A,2CH	00	00	12	56	78	34
XCH	A,2DH	00	00	12	34	78	56
XCH	A,2EH	00	00	12	34	56	78

(b) Using XCHs: 9 bytes, 5 µs

Figure 4-2. Shifting BCD Number Two Digits Right

				2A	2B	2C	2D	2E	ACC
	MOV	R1,#2EH		00	12	34	56	78	XX
	MOV	R0,#2DH		00	12	34	56	78	XX
	loop for R1 = 2EH								
LOOP:	MOV	A,@R1		00	12	34	56	78	78
	XCHD	A,@R0		00	12	34	58	78	76
	SWAP	A		00	12	34	58	78	67
	MOV	@R1,A		00	12	34	58	67	67
	DEC	R1		00	12	34	58	67	67
	DEC	R0		00	12	34	58	67	67
	CJNE	R1,#2AH,	LOOP						
	loop for R1 =	A,2DH:		00	12	38	45	67	45
	loop for R1 =	A,2CH:		00	18	23	45	67	23
	loop for R1 =	A,2BH:		08	01	23	45	67	01
	CLR	A		08	01	23	45	67	00
	XCH	A,2AH		00	01	23	45	67	08

Figure 4-3. Shifting BCD Number One Digit Right

First, pointers R1 and R0 are set up to point to the two bytes containing the last four BCD digits. Then a loop is executed which leaves the last byte, location 2EH, holding the last two digits of the shifted number. The pointers are decremented, and the loop is repeated for location 2DH. The CJNE instruction (Compare and if Not Equal) is a loop control that will be described later.

The loop is executed from LOOP to CJNE for R1=2EH, 2DH, 2CH and 2BH. At that point the digit that was originally shifted out on the right has propagated to location 2AH. Since that location should be left with 0s, the lost digit is moved to the Accumulator.

4.1.5.2 External RAM

Table 4-6 shows a list of the Data Transfer instructions that access external Data Memory. Only indirect addressing can be used. The choice is whether to use a one-byte address, @Ri, where Ri can be either R0 or R1 of the selected register bank, or a two-byte address, @DPTR. The disadvantage to using 16-bit addressees if only a few K bytes of external RAM are involved is that 16-bit addresses use all 8-bits of Port 2 as address bus. On the other hand, 8-bit addresses allow one to address a few K bytes of RAM, as shown in Figure 5, without having to sacrifice all of Port 2.

All of these instructions execute in 2 µs, with a 12 MHz clock.

Table 4-6. Transfer Instructions for Accessing External Data Memory Space

Address Width	Mnemonic	Operation	Execution Time (µs)
8 bits	MOVX A,@Ri	Read external RAM @Ri	2
8 bits	MOVX@Ri,A	Write external RAM @Ri	2
16 bits	MOVX A,@DPTR	Read external RAM @DPTR	2
16 bits	MOVX@DPTR,A	Write external RAM @DPTR	2

NOTE

In all external Data RAM accesses, the Accumulator is always either the destination or source of the data.

The read and write strobes to external RAM are activated only during the execution of a MOVX instruction. Normally these signals are inactive, and in fact if they're not going to be used at all, their pins are available as extra I/O lines.

4.1.5.3 Lookup Tables

Table 4-7 shows the two instructions that are available for lookup tables in Program Memory. Since these instructions access only Program Memory, the lookup tables can only be read, not updated The mnemonic is MOVC for "move constant".

If the table access is to external Program Memory, then the read strobe is PSEN#.

Table 4-7. MCS® 51 Read Instructions

Mnemonic	Operation	Execution Time (µs)
MOVC A,@A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A,@A+PC	Read Pgm Memory at (A+PC)	2

The first MOVC instruction in Table 4-7 can accommodate a table of up to 256 entries, numbered 0 through 255. The number of the desired entry is loaded into the Accumulator, and the Data Pointer is set up to point to beginning of the table. Then

```
MOVC A, @A+DPTR
```

copies the desired table entry into the Accumulator.

The other MOVC instruction works the same way, except the Program Counter (PC) is used as the table base, and the table is accessed through a subroutine. First the number of the desired entry is loaded into the Accumulator, and the subroutine is called:

```
MOV A, ENTRY_NUMBER
```

```
CALL TABLE
```

The subroutine "TABLE" would look like this:

```
TABLE: MOVC A,@A+PC
       RET
```

The table itself immediately follows the RET (return) instruction in Program Memory. This type of table can have up to 255 entries, numbered 1 through 255. Number 0 can not be used, because at the time the MOVC instruction is executed, the PC contains the address of the RET instruction. An entry numbered 0 would be the RET opcode itself.

4.1.6 Boolean Instructions

MCS® 51 devices contain a complete Boolean (single-bit) processor. The internal RAM contains 128 addressable bits, and the SFR space can support up to 128 other addressable bits. All of the port lines are bit-addressable, and each one can be treated as a separate single-bit port. The instructions that access these bits are not just conditional branches, but a complete menu of move, set, clear, complement, OR, and AND instructions. These kinds of bit operations are not easily obtained in other architectures with any amount of byte-oriented software.

Table 4-8. MCS® 51Boolean Instructions

Mnemonic	Operation	Execution Time (µs)
ANL C, bit	C = C . AND.bit	2
ANL C,/bit	C = C . AND..NOT.bit	2
ORL C,bit	C = C .OR. bit	2
ORL C,/bit	C = C. OR. .NOT. bit	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C=1	1
SETB bit	bit = 0	1
CPL C	C = .NOT.C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1; CLR bit	2

The instruction set for the Boolean processor is shown in Table 4-8. All bit accesses are by direct addressing. Bit addresses 00H through 7FH are in the Lower 128, and bit addresses 80H through FFH are in SFR space.

Note how easily an internal flag can be moved to a port pin:

```
MOV C ,FLAG
MOV P1.0 ,C
```

In this example, FLAG is the name of any addressable bit in the Lower 128 or SFR space. An I/O line (the LSB of Port 1, in this case) is set or cleared depending on whether the flag bit is 1 or 0.

The Carry bit in the PSW is used as the single -bit Accumulator of the Boolean processor. Bit instructions that refer to the Carry bit as C assemble as Carry-specific instructions (CLR C, etc). The Carry bit also has a direct address, since it resides in the PSW register, which is bit-addressable.

Note that the Boolean instruction set includes ANL and ORL operation, but not the XRL (Exclusive OR) operation. An XRL operation is simple to implement in software. Suppose, for example, it is required to form the Exclusive OR of two bits:

```
C = bit1 .XRL. bit2
```

The software to do that could be as follows:

```
MOV     C, bit1
JNB     bit2, OVER
CPL      C

OVER: (continue)
```

First, bit1 is moved to the Carry. If bit2 = 0, then C now contains the correct result. That is, bit1 .XRL. bit2 = bit1 if bit2 = 0. On the other hand, if bit2 = 1, C now contains the complement of the correct result. It need only be inverted (CPL C) to complete the operation.

This code uses the JNB instruction, one of a series of bit-test instructions which execute a jump if the addressed bit is set (JC, JB, JBC) or if the addressed bit is not set (JNC, JNB). In the above case, bit2 is being tested, and if bit2 = 0, the CPL C instruction is jumped over.

JBC executes the jump if the addressed bit is set, and also clears the bit. Thus a flag can be tested and cleared in one operation.

All the PSW bits are directly addressable, so the Parity bit, or the general purpose flags, for example, are also available to the bit-test instructions.

4.1.6.1 Relative Offset

The destination address for these jumps is specified to the assembler by a label or by an actual address in Program Memory. However, the destination address assembles to a relative offset byte. This is a signed (two’s complement) offset byte which is added to the PC in two’s complement arithmetic if the jump is executed.

The range of the jump is therefore –128 to +127 Program Memory bytes relative to the first byte following the instruction.

4.1.7 Jump Instructions

Table 4-9 shows the list of unconditional jumps.

Table 4-9. Unconditional Jumps in MCS® 51 Devices

Mnemonic	Operation	Execution Time (µs)
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

The table lists a single "JMP addr" instruction, but in fact there are three—SJMP, LJMP and AJMP—which differ in the format of the destination address. JMP is a generic mnemonic which can be used if the programmer does not care which way the jump is encoded.

The SJMP instruction encodes the destination address as a relative offset, as described above. The instruction is 2 bytes long, consisting of the opcode and the relative offset byte. The jump distance is limited to a range of –128 to +127 bytes relative to the instruction following the SJMP.

The LJMP instruction encodes the destination address as a 16-bit constant. The instruction is 3 bytes long, consisting of the opcode and two address bytes. The destination address can be anywhere in the 64K Program Memory space.

The AJMP instruction encodes the destination address as an 11-bit constant. The instruction is 2 bytes long, consisting of the opcode, which itself contains 3 of the 11 address bits, followed by another byte containing the low 8 bits of the destination address. When the instruction is executed, these 11 bits are simply substituted for the low 11 bits in the PC. The high 5 bits stay the same. Hence, the destination has to be within the same 2K block as the instruction following the AJMP.

In all cases the programmer specifies the destination address to the assembler in the same way: as a label or as a 16-bit constant. The assembler will put the destination address into the correct format for the given instruction. If the format required by the instruction will not support the distance to the specified destination address, a "Destination out of range" message is written into the List file.

The JMP @A+DPTR instruction supports case jumps. The destination address is computed at execution time as the sum of the 16-bit DPTR register and the Accumulator. Typically, DPTR is set up with the address of a jump table, and the Accumulator is given an index to the table. In a 5-way branch, for example, an integer 0 through 4 is loaded into the Accumulator. The code to be executed might be as follows:

```
MOV     DPTR,#JUMP_TABLE
MOV     A,INDEX_NUMBER
```

```
RL      A
JMP     @A+DPTR
```

The RL A instruction converts the index number (0 through 4) to an even number on the range 0 through 8, because each entry in the jump table is 2 bytes long:

```
JUMP_TABLE:
    AJMP    CASE_0
    AJMP    CASE_1
    AJMP    CASE_2
    AJMP    CASE_3
    AJMP    CASE_4
```

Table 4-9 shows a single “CALL addr” instruction, but there are two of them—LCALL and ACALL—which differ in the format in which the subroutine address is given to the CPU. CALL is a generic mnemonic which can be used if the programmer does not care which way the address is encoded.

The LCALL instruction uses the 16-bit address format, and the subroutine can be anywhere in the 64K Program Memory space. The ACALL instruction uses the 11-bit format, and the subroutine can be anywhere in the 64K Program Memory space. The ACALL instruction uses the 11-bit format, and the subroutine must be in the same 2K block as the instruction following the ACALL.

In any case, the programmer specifies the subroutine address to the assembler in the same way: as a label or as a 16-bit constant. The assembler will put the address into the correct format for the given instructions.

Subroutines should end with a RET instruction, which returns execution to the instruction following the CALL.

RETI is used to return from an interrupt service routine. The only difference between RET and RETI is that RETI tells the interrupt control system that the interrupt in progress is done. If there is no interrupt in progress at the time RETI is executed, then the RETI is functionally identical to RET.

Table 4-10 shows the list of conditional jumps available to the MCS 51 user. All of these jumps specify the destination address by the relative offset method, and so are limited to a jump distance of –128 to +127 bytes from the instruction following the conditional jump instruction. Important to note, however, the user specifies to the assembler the actual destination address the same way as the other jumps: as a label or a 16-bit constant.

Table 4-10. Conditional Jumps in MCS® 51 Devices

Mnemonic	Operation	Addressing Modes				Execution Time (µs)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A not equal to 0	Accumulator only				2
DJNZ <byte> , rel	Decrement and jump if not zero	X		X		2
CJNE A, <byte> , rel	Jump if A not equal to <byte>	X			X	2
CJNE <byte> , #data,rel	Jump if <byte> not equal to #data		X	X		2

There is no Zero bit in the PSW. The JZ and JNZ instructions test the Accumulator data for that condition.

The DJNZ instruction (Decrement and Jump if Not Zero) is for loop control. To execute a loop N times, load a counter byte with N and terminate the loop with a DJNZ to the beginning of the loop, as shown below for N = 10:

```
        MOV     COUNTER,#10
LOOP:   (begin loop)
          *
          *
          *
        (end loop)
        DJNZ    COUNTER,LOOP
        (continue)
```

The CJNE instruction (Compare and Jump if Not Equal) can also be used for loop control as in Figure 4-3. Two bytes are specified in the operand field of the instruction. The jump is executed only if the two bytes are not equal. In the example of Figure 4-3, the two bytes are the data in R1 and the constant 2AH. The initial data in R1 is 2EH. Every time the loop is executed, R1 is decremented, and the looping is to continue until the R1 data reaches 2AH.

Another application of this instruction is in "greater than, less than" comparisons. The two bytes in the operand field are taken as unsigned integers. If the first is greater than or equal to the second, then the Carry bit is cleared.

5

Interrupt System

CHAPTER 5
INTERRUPT SYSTEM

5.1 OVERVIEW

The 8*x*931, like other control-oriented microcontroller architectures, employs a program interrupt method. This operation branches to a subroutine and performs some service in response to the interrupt. When the subroutine completes, execution resumes at the point where the interrupt occurred. Interrupts may occur as a result of internal 8*x*931 activity (e.g., timer overflow) or at the initiation of electrical signals external to the microcontroller(e.g., keyboard scan). In all cases, interrupt operation is programmed by the system designer, who determines priority of interrupt service relative to normal code execution and other interrupt service routines. All of the interrupts are enabled or disabled by the system designer and may be manipulated dynamically.

A typical interrupt event chain occurs as follows:

1. An internal or external device initiates an interrupt-request signal. This signal, connected to an input pin (see Table 5-1) and periodically sampled by the 8*x*931, latches the event into a flag buffer.
2. The interrupt handler compares the priority of the flag to the priority of other interrupts. A high priority causes the handler to set an interrupt flag. This signals the instruction execution unit to execute a context switch.
3. This context switch breaks the flow of the instruction sequence. The execution unit completes the current instruction prior to a save of the program counter (PC) and reloads the PC with the start address of a firmware service routine.
4. The firmware service routine executes assigned tasks and as a final activity performs a RETI (return from interrupt instruction).
5. The RETI instruction signals completion of the interrupt, resets the interrupt-in-progress priority, and reloads the program counter.
6. Program operation then continues from the original point of interruption.

Table 5-1. Interrupt System Input Signals

Signal Name	Type	Description	Multiplexed With
INT1:0#	I	**External Interrupts 0 and 1.** These inputs set bits IE1:0 in the TCON register. If bits IT1:0 in the TCON register are set, bits IE1:0 are controlled by a negative-edge trigger on INT1#/INT0#. If bits IT1:0 are clear, bits IE1:0 are controlled by a low-level trigger on INT1:0#.	P3.3:2
KSI7:0	I	**Keyboard Scan Input.** Schmitt-trigger inputs with firmware-enabled internal pullup resistors used for the input side of the keyboard scan matrix.	AD7:0/ P0.7:0

NOTE: Other signals are defined in their respective chapters and in Appendix B, “Pin Descriptions”.

Figure 5-1 illustrates the interrupt control system. The 8*x*931 has ten maskable interrupt sources. These include two external interrupts (INT0# and INT1#), three timer interrupts (timers 0, 1 and 2), a serial port interrupt, a keyboard scan interrupt, and three USB interrupts (one of which doubles as a hub interrupt). Each interrupt has an interrupt request flag, which can be set by firmware as well as by hardware (see Table 5-6 on page 5-26). For some interrupts, hardware clears the request flag when it grants the interrupt. Firmware can clear any request flag to cancel an impending interrupt.

There are three types of USB interrupts, as shown in Figure 5-1. The USB function interrupt, used to control the flow of non-isochronous data; the hub/any start-of-frame interrupt (SOF), used to signal a hub interrupt or to monitor the transfer of isochronous data; and the global suspend/resume interrupt, used to allow USB power control and to provide a USB reset separation interrupt. These interrupts are enabled using the IEN1 register. See Table 5-6 on page 5-26 and Figure 5-11.

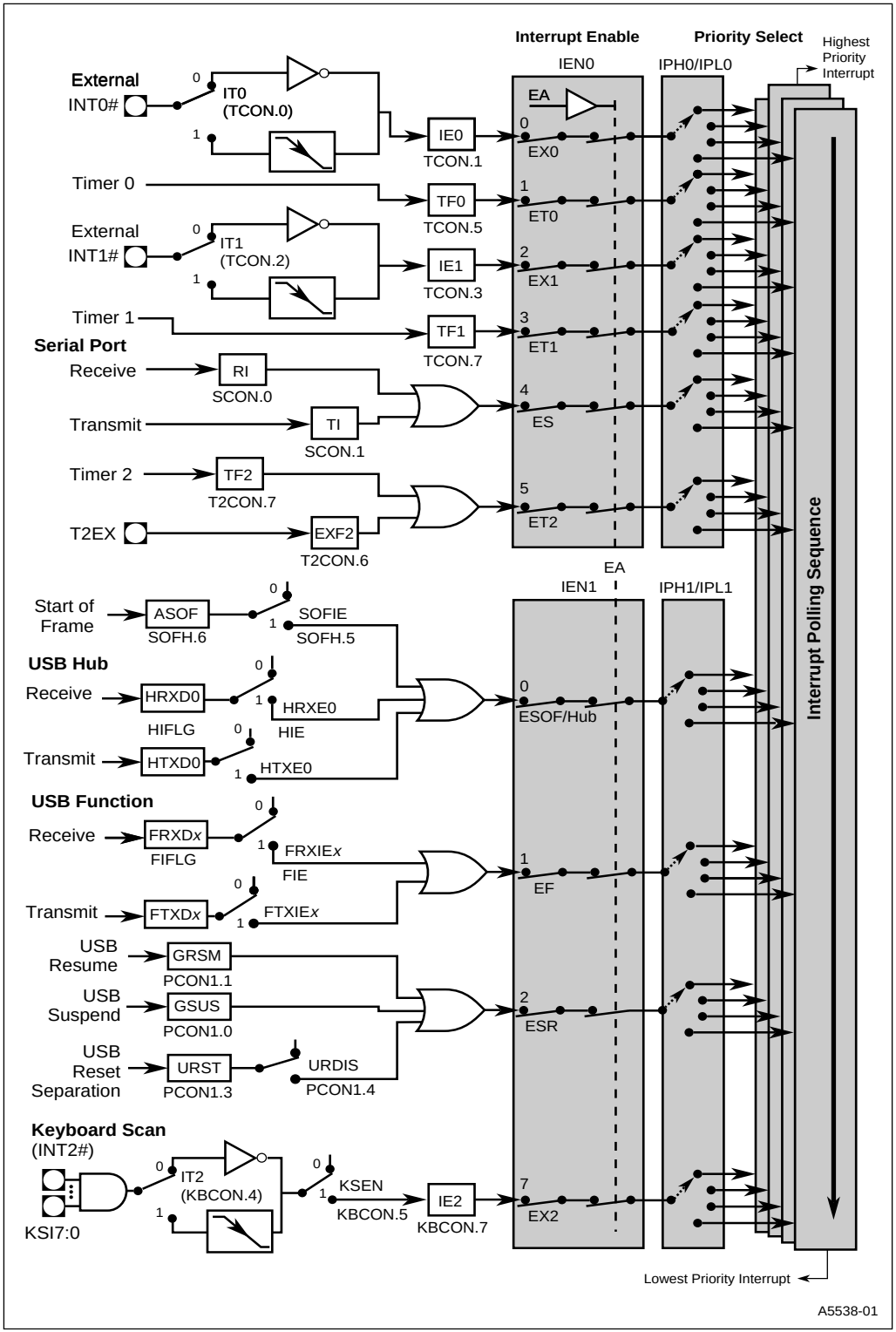


Figure 5-1. Interrupt Control System

SFRs used by the interrupt system are listed in Table 5-2. Figure 5-2 shows the bits contained in the interrupt SFRs.

Table 5-2. Interrupt System Special Function Registers

Mnemonic	Description	Address	Page
FIE	**USB Function Interrupt Enable Register.** Enables and disables the receive and transmit done interrupts for the function endpoints.	A2H	page 5-9
FIFLG	**USB Function Interrupt Flag Register.** Contains the USB function's transmit and receive done interrupt flags for non-isochronous endpoints.	C0H	page 5-11
HIE	**Hub Interrupt Enable Register.** Contains the hub interrupt enable bits.	A1H	page 5-15
HIFLG	**Hub Interrupt Flag Register.** Contains the hub interrupt status flags.	E8H	page 5-16
IEN0	**Interrupt Enable Register 0.** Enables individual programmable interrupts. Also provides a global enable for the programmable interrupts. The reset value for this register is zero (interrupts disabled).	A8H	page 5-24
IEN1	**Interrupt Enable Register1**. Enables individual programmable interrupts for the USB interrupts. The reset value of this register is zero (interrupts disabled).	B1H	page 5-25
IPL0	**Interrupt Priority Low Register 0**. Establishes relative priority for programmable interrupts. Used in conjunction with IPH0.	B8H	page 5-28
IPH0	**Interrupt Priority High Register 0**. Establishes relative priority for programmable interrupts. Used in conjunction with IPL0.	B7H	page 5-27
IPL1	**Interrupt Priority Low Register 1.** Establishes relative priority for programmable interrupts. Used in conjunction with IPH1.	B2H	page 5-30
IPH1	**Interrupt Priority High Register 1.** Establishes relative priority for programmable interrupts. Used in conjunction with IPL1.	B3H	page 5-29
KBCON	**Keyboard Control Register.** This register controls the keyboard scan input and output activity, enables and configures the keyboard scan interrupt, and drives the keyboard LEDs.	F8H	page 12-1
PCON1	**USB Power Control.** Contains USB global suspend and resume interrupt bits. Also contains the USB reset separation enable and interrupt bits.	DFH	page 14-4
SOFH	**Start of Frame High Register.** Contains isochronous data transfer enable and interrupt bits and the upper-three bits of the 11-bit time stamp received from the host.	D3H	page 5-12
SOFL	**Start of Frame Low Register.** Contains the lower-eight bits of the 11-bit time stamp received from the host.	D2H	page 5-13

NOTE: Other SFRs are described in their respective chapters and in Appendix C, "Registers".

	7							0
FIE	—	—	FRXIE2	FTXIE2	FRXIE1	FTXIE1	FRXIE0	FTXIE0
FIFLG	—	—	FRXD2	FTXD2	FRXD1	FTXD1	FRXD0	FTXD0
HIE	—	—	—	—	—	—	HRXE0	HTXE0
HIFLG	—	—	—	—	—	—	HRXD0	HTXD0
IEN0	EA	—	ET2	ES	ET1	EX1	ET0	EX0
IEN1	EX2	—	—	—	—	ESR	EF	ESOF
IPL0	—	—	IPL0.5	IPL0.4	IPL0.3	IPL0.2	IPL0.1	IPL0.0
IPH0	—	—	IPH0.5	IPH0.4	IPH0.3	IPH0.2	IPH0.1	IPH0.0
IPL1	IPL1.7	—	—	—	—	IPL1.2	IPL1.1	IPL1.0
IPH1	IPH1.7	—	—	—	—	IPH1.2	IPH1.1	IPH1.0
KBCON	IE2	—	KSEN	IT2	LED3	LED2	LED1	LED0
PCON1	—	—	—	URDIS	URST	RWU	GRSM	GSUS
SOFH	SOFACK	ASOF	SOFIE	FTLOCK	SOFODIS	TS10	TS9	TS8
SOFL	TS7:0							

Figure 5-2. Bits of the Interrupt SFRs

Many 8*x*931 interrupts are similar to the interrupts of other MCS® 51 microprocessors. These interrupts are shown in Table 5-4. Particulars of the USB and hub interrupts are given in Table 5-6.

5.2 INTERRUPT SOURCES

Interrupt sources for the 8*x*931 include external interrupts, timer interrupts, a keyboard scan interrupt, USB function transmit and receive interrupts, a USB start-of-frame interrupt, a USB global suspend and resume interrupt, and a separate USB-only reset interrupt.

These interrupts are described in the following subsections.

5.2.1 External Interrupts

External interrupts INT0# and INT1# (INT*x*#) pins may each be programmed to be level-triggered or edge-triggered, dependent upon bits IT0 and IT1 in the TCON register (see Figure 10-6 on page 10-8). If IT*x* = 0, INT*x*# is triggered by a detected low at the pin. If IT*x* = 1, INT*x*# is negative-edge triggered. External interrupts are enabled with bits EX0 and EX1 (EX*x*) in the IEN0 register (see Figure 5-10 on page 5-24). Events on the external interrupt pins set the interrupt request flags IE*x* in TCON. These request bits are cleared during the hardware vector to the service routine only if the interrupt is negative-edge triggered. If the interrupt is level-triggered, the interrupt service routine must clear the request bit. External hardware must deassert INTx# before the service routine completes or an additional interrupt is requested. External interrupt pins must be deasserted for at least four state times prior to a request.

External interrupt pins are sampled once every six state times (a frame length of 1 µs at 6 MHz). A level-triggered interrupt pin held low or high for any five-state time period guarantees detection. Edge-triggered external interrupts must hold the request pin low for at least seven state times. This ensures edge recognition and sets interrupt request bit EX*x*. The CPU clears EX*x* automatically during service routine fetch cycles for edge-triggered interrupts.

Table 5-3. 8*x*931AA/HA Interrupt Control Matrix

	Interrupt Name†						
	Keyboard Scan [INT2#]	**Timer 2**	**Serial Port**	**Timer 1**	**INT1#**	**Timer 0**	**INT0#**
Bit Name in IEN0 Register	—	ET2	ES	ET1	EX1	ET0	EX0
Bit Name in IEN1 Register	EX2	—	—	—	—	—	—
Interrupt Priority-Within-Level (11 = Low Priority, 1 = High Priority)	7	6	5	4	3	2	1
Bit Names in: IPH0 IPL0	 IPH1.7 IPL1.7	 IPH0.5 IPL0.5	 IPH0.4 IPL0.4	 IPH0.3 IPL0.3	 IPH0.2 IPL0.2	 IPH0.1 IPL0.1	 IPH0.0 IPL0.0
Programmable for Negative-edge triggered or Level-triggered Detect?	Yes	No	No	No	Yes	No	Yes
Interrupt Request Flag in TCON or KBCON Register	KBCON: IE2	TCON: TF2 EXF2	SCON: RI TI	TCON: TF1	TCON: IE1	TCON: TF0	TCON: IE0
Interrupt Request Flag Cleared by Hardware?	No	No	No	Yes	Edge Yes, Level No	Yes	Edge Yes, Level No
ISR Vector Address	003BH	002BH	0023H	001BH	0013H	000BH	0003H

† Additional interrupts specific to USB and USB hub operation appear in Table 5-11.

Table 5-4. 8*x*931 USB/Hub Interrupt Control Matrix

	Interrupt Name		
	USB Global Suspend/Resume and USB Reset	**USB Function [Non-Isochronous Endpoint]**	**USB Hub/SOF [Isochronous Endpoint]**
Bit Name in IEN1 Register	ESR	EF	ESOF
Interrupt Priority-Within-Level (11 = Low Priority, 1 = High Priority)	11	10	9
Bit Names in: IPH1 IPL1	IPH1.2 IPL1.2	IPH1.1 IPL1.1	IPH1.0 IPL1.0
Interrupt Request Flag	PCON1: GSUS GRSM URST	FIFLG: FTXD*x*, FRXD*x* *x*=0,1,2	SOFH:ASOF, HIFLG: HTXD0, HRXD0
Interrupt Request Flag Cleared by Hardware?	No	No	No
ISR Vector Address	0053H	004BH	0043H

5.2.2 Timer Interrupts

Two timer-interrupt request bits TF0 and TF1 (see TCON register, Figure 10-6 on page 10-8) are set by timer overflow. The exception is Timer 0 in Mode 3, see Figure 10-4 on page 10-6. When a timer interrupt is generated, the bit is cleared during the hardware vector to the interrupt service routine. Timer interrupts are enabled by bits ET0 and ET1 in the IEN0 register (see Figure 5-10 on page 5-24).

Timer 2 interrupts are generated by a logical OR of bits TF2 and EXF2 in register T2CON. Neither flag is cleared by a hardware vector to a service routine. In fact, the interrupt service routine must determine if TF2 or EXF2 generated the interrupt, and then clear the bit. Timer 2 interrupt is enabled by ET2 in register IEN0 (Figure 5-10).

5.2.3 Keyboard Scan Interrupt

The keyboard scan interrupt (INT2#) is actually an external interrupt similar to INT0# and INT1#, except that it is based on the ANDed inputs KSI7:0. When any one of the KSI7:0 signals drops to 0, the keyboard scan interrupt is triggered. This can happen on either a level 0 or the negative edge of a KSI7:0 signal, depending on the value of IT2 in KBCON (Figure 12-1 on page 12-1).

If the keyboard scan enable bit is set (KSEN in KBCON), the keyboard scan interrupt flag (called the Interrupt 2 Flag and represented by the IE2 bit in KBCON) is set when one of the KSI7:0 sig-

nals becomes zero. If external interrupt 2 is enabled (by setting EX2 in IEN1), a hardware interrupt is triggered and program control vectors to 003BH.

See "Keyboard Interrupt Logic" on page 12-3 for additional information.

5.2.4 Serial Port Interrupt

Serial port interrupts are generated by the logical OR of bits RI and TI in the SCON register. Neither flag is cleared by a hardware vector to the service routine. The service routine resolves RI or TI interrupt generation and clears the serial port request flag. The serial port interrupt is enabled by bit ES in the IEN0 register (see Figure 5-10).

5.2.5 USB Function Interrupt

The USB function generates two types of interrupts to control the transfer of non-isochronous data: the receive done interrupt and the transmit done interrupt. Individual USB function interrupts in the 8*x*931 are enabled by setting the corresponding bits in the FIE register (Figure 5-3).

NOTE

To use any of the USB function interrupts, the EF bit in the IEN1 register must be enabled.

The USB Function Interrupt Flag register (FIFLG, as shown in Figure 5-4) is used to indicate pending function interrupts for a given endpoint. For all bits in FIFLG, a '1' indicates that an interrupt is actively pending for that endpoint; a '0' indicates that the interrupt is not active. The interrupt status is shown in the FIFLG register regardless of the state of the corresponding interrupt enable bit in the FIE register (Figures 5-3).

FIE Address: A2H
Reset State: xx00 0000B

Function Interrupt Enable Register. Enables and disables the receive and transmit done interrupts for the three function endpoints.

7							0
—	—	FRXIE2	FTXIE2	FRXIE1	FTXIE1	FRXIE0	FTXIE0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	FRXIE2	Function Receive Interrupt Enable 2: Enables the receive done interrupt for endpoint 2 (FRXD2).
4	FTXIE2	Function Transmit Interrupt Enable 2: Enables the transmit done interrupt for endpoint 2 (FTXD2).
3	FRXIE1	Function Receive Interrupt Enable 1: Enables the receive done interrupt for endpoint 1 (FRXD1).
2	FTXIE1	Function Transmit Interrupt Enable 1: Enables the transmit done interrupt for endpoint 1 (FTXD1).
1	FRXIE0	Function Receive Interrupt Enable 0: Enables the receive done interrupt for endpoint 0 (FRXD0).
0	FTXIE0	Function Transmit Interrupt Enable 0: Enables the transmit done interrupt for endpoint 0 (FTXD0).

NOTE: For all bits, a '1' means the interrupt is enabled and will cause an interrupt to be signaled to the microcontroller. A '0' means the associated interrupt source is disabled and cannot cause an interrupt, even though the interrupt bit's value is still reflected in the FIFLG register.

Figure 5-3. FIE: USB Function Interrupt Enable Register

The USB function generates a receive done interrupt for an endpoint *x* (*x* = 0–2) by setting the FRXD*x* bit in the FIFLG register (Figure 5-4). Only a non-isochronous transfer can cause a receive done interrupt. Receive done interrupts are generated only when *all of the following are true*:

- A valid SETUP or OUT token is received to function endpoint *x*.
- Endpoint *x* is enabled for reception (RXEPEN in EPCON = '1').
- Receive is enabled (RXIE = '1') and STALL is disabled (RXSTL = '0') for OUT tokens (*or* the token received is a SETUP token).
- A data packet is received with no time-out — *regardless* of transmission errors (CRC, bit-stuffing) or FIFO errors (overrun, underrun).

- There is no data sequence PID error.

Because the FRXD*x* bit is set (and a receive done interrupt is generated) regardless of transmission errors, this condition means *either*:

- Valid data is waiting to be serviced in the receive FIFO for function endpoint *x* and that the data was received without error and has been acknowledged.
- Data was received with a receive data error and requires firmware intervention to be cleared. This could be either a transmission error or a FIFO-related error. You must check for these conditions and respond accordingly in the interrupt service routine (ISR).

The USB function generates a transmit done interrupt for an endpoint *x* (*x* = 0–2) by setting the FTXD*x* bit in the FIFLG register (Figure 5-4). Only a non-isochronous transfer can cause a transmit done interrupt. Transmit done interrupts are generated only when *all of the following are true*:

- A valid IN token is received to function endpoint *x*.
- Endpoint *x* is enabled for transmission (TXEPEN = '1').
- Transmit is enabled (TXIE = '1') and STALL is disabled (TXSTL = '0').
- A data packet/byte count has been loaded in the transmit FIFO and was transmitted in response to the IN token — *regardless* of whether or not a FIFO error occurs.
- An ACK is received from the host or there was a time-out in the SIE.

Because the FTXD*x* bit is set (and a transmit done interrupt is generated) regardless of transmission errors, this condition means either:

- The transmit data has been transmitted and the host has sent an acknowledgment to indicate that is was successfully received.
- A transmit data error occurred during transmission of the data packet, which requires servicing by firmware to be cleared. You must check for these conditions and respond accordingly in the ISR.

NOTE

Setting an endpoint interrupt's bit in the Function Interrupt Enable register (FIE, as shown in Figure 5-3) means that the interrupt is enabled and will cause an interrupt to be signaled to the microcontroller. Clearing a bit in the FIE register disables the associated interrupt source, which can no longer cause an interrupt even though its value will still be reflected in the FIFLG register (Figure 5-4).

FIFLG Address: C0H
Reset State: xx00 0000B

Function Interrupt Flag Register. Contains the USB function's transmit and receive done interrupt flags for non-isochronous endpoints.

7							0
—	—	FRXD2	FTXD2	FRXD1	FTXD1	FRXD0	FTXD0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	FRXD2	Function Receive Done Flag, Endpoint 2
4	FTXD2	Function Transmit Done Flag, Endpoint 2
3	FRXD1	Function Receive Done Flag, Endpoint 1
2	FTXD1	Function Transmit Done Flag, Endpoint 1
1	FRXD0	Function Receive Done Flag, Endpoint 0
0	FTXD0	Function Transmit Done Flag, Endpoint 0

NOTES:
1. For all bits in the Interrupt Flag Register, a '1' indicates that an interrupt is actively pending; a '0' indicates that the interrupt is not active. The interrupt status is shown regardless of the state of the corresponding interrupt enable bit in the FIE.
2. Bits are set-only by hardware and clearable in firmware. Firmware can also set the bits for test purposes, allowing the interrupt to be generated in firmware. This SFR is bit-addressable.
3. A set bit indicates either:
   Valid data waiting to be serviced in the RX FIFO for the indicated endpoint and that the data was received without error and has been acknowledged; or
   Data was received with a Receive Data Error requiring firmware intervention to be cleared.

Figure 5-4. FIFLG: USB Function Interrupt Flag Register

5.2.6 USB Start-of-frame Interrupt

The USB start-of-frame interrupt (SOF) is used to control the transfer of isochronous data. The 8*x*931 frame timer attempts to synchronize with the host frame time automatically. When the frame timer is locked to the USB frame time, hardware sets the FTLOCK bit in SOFH (Figure 5-5). To enable the SOF interrupt, set the SOFIE bit in SOFH.

The 8*x*931 generates an SOF interrupt whenever a start-of-frame packet is received from the USB lines, or whenever a start-of-frame packet should have been received (i.e., an artificial SOF). The 8*x*931 generates an SOF interrupt by setting the ASOF bit in the SOFH SFR. When an SOF interrupt occurs, the 8*x*931 loads the current value of the 11-bit frame number issued with an SOF token into the SOFH/SOFL registers (Figures 5-5 and 5-6). If an artificial SOF is generated, the time stamp remains at its previous value — leaving it up to the firmware for updating.

NOTE

For the 8*x*931HA, the start-of-frame interrupt shares an interrupt vector with the hub interrupt. When this interrupt is triggered, firmware must examine the ASOF bit in the SOFH SFR to determine that it was the start-of-frame interrupt that was triggered, and not the hub interrupt.

SOFH Address: D3H
Reset State: 0000 1000B

Start-of-frame High Register. Contains isochronous data transfer enable and interrupt bits and the upper three bits of the 11-bit time stamp received from the host.

7							0
SOFACK	ASOF	SOFIE	FTLOCK	SOFODIS	TS10	TS9	TS8

Bit Number	Bit Mnemonic	Function
7	SOFACK	SOF Token Received without Error (read-only): When set, this bit indicates that the 11-bit time stamp stored in SOFL and SOFH is valid. This bit is updated every time an SOF token is received from the USB bus, and it is cleared when an artificial SOF is generated by the frame timer. This bit is set and cleared by hardware.
6	ASOF	Any Start of Frame: This bit is set by hardware to indicate that a new frame has started. The interrupt can result either from reception of an actual SOF packet or from an artificially-generated SOF from the frame timer. This interrupt is asserted in hardware even if the frame timer is not locked to the USB bus frame timing. When set, this bit is an indication that either an actual SOF packet was received or an artificial SOF was generated by the frame timer. This bit must be cleared by firmware or inverted and driven to the SOF# pin. The effect of setting this bit by firmware is the same as hardware: the external pin will be driven with an inverted ASOF value for eight T_{CLK}s. This bit also serves as the SOF interrupt flag. This interrupt is only asserted in hardware if the SOF interrupt is enabled (SOFIE set) and the interrupt channel is enabled.
5	SOFIE	SOF Interrupt Enable: When this bit is set, setting the ASOF bit causes an interrupt request to be generated if the interrupt channel is enabled. Hardware reads but does not write this bit.
4	FTLOCK	Frame Timer Locked (read-only): When set, this bit indicates that the frame timer is presently locked to the USB bus' frame time. When cleared, this bit indicates that the frame timer is attempting to synchronize to the frame time.

Figure 5-5. SOFH: Start-of-frame High Register

SOFH (Continued) Address: D3H
Reset State: 0000 1000B

Start-of-frame High Register. Contains isochronous data transfer enable and interrupt bits and the upper three bits of the 11-bit time stamp received from the host.

7							0
SOFACK	ASOF	SOFIE	FTLOCK	SOFODIS	TS10	TS9	TS8

Bit Number	Bit Mnemonic	Function
3	SOFODIS	SOF# Pin Output Disable: When set, the SOF# pin will be disabled and will respond like a port pin. The SOF# pin will be driven to '1' when SOFODIS is set. When this bit is clear, setting the ASOF bit causes the SOF# pin to be toggled with a low pulse for eight T_{CLK}s.
2:0	TS10:8	Time stamp received from host: TS10:8 are the upper three bits of the 11-bit frame number issued with an SOF token. This time stamp is valid only if the SOFACK bit is set. If an artificial SOF is generated, the time stamp remains at its previous value and it is up to firmware to update it. These bits are set and cleared by hardware.

Figure 5-5. SOFH: Start-of-frame High Register (Continued)

SOFL Address: D2H
Reset State: 0000 0000B

Start-of-frame Low Register. Contains the lower eight bits of the 11-bit time stamp received from the host.

7	0
TS7:0	

Bit Number	Bit Mnemonic	Function
7:0	TS7:0	Time stamp received from host: This time stamp is valid only if the SOFACK bit in the SOFH register is set. TS7:0 are the lower eight bits of the 11-bit frame number issued with an SOF token. If an artificial SOF is generated, the time stamp remains at its previous value and it is up to firmware to update it. These bits are set and cleared by hardware.

Figure 5-6. SOFL: Start-of-frame Low Register

The 8*x*931 uses the SOF interrupt to signal either of two complementary events:

1. When transmitting: The next isochronous data packet needs to be retrieved from memory and loaded into the transmit FIFO in preparation for transmission in the next frame; or
2. When receiving: An isochronous packet has been received in the previous frame and needs to be retrieved from the receive FIFO.

Since the SOF packet could be corrupted, there is a chance that a new frame could be started without successful reception of the SOF packet. For this reason, an artificial SOF is provided. The frame timer signals a time-out when an SOF packet has not been received within the allotted amount of time. In this fashion, the 8*x*931 generates an SOF interrupt reliably once each frame within 1 µs of accuracy, except when this interrupt is suspended or when the frame timer gets out-of-sync with the USB bus frame time.

In summary, to use the USB start-of-frame functionality for isochronous data transfer, *the following must all be true*:

- The global enable bit must be set. That is, the EA bit must be set in the IEN0 register (Figure 5-10).
- The isochronous endpoint Any SOF interrupt must be enabled. That is, the ESOF bit must be set in the IEN1 register (Figure 5-11).
- The SOF interrupt must be enabled. That is, the SOFIE bit must be set in the SOFH Register (Figure 5-5).

NOTE

The SOF interrupt is brought out to an external pin (SOF#) in order to provide a 1 ms pulse, subject to the accuracy of the USB start-of-frame. This pin is enabled by clearing the SOFODIS bit in the SOFH register.

5.2.7 USB Hub Interrupt

The USB hub interrupt is used to signal a receive done or transmit done for hub endpoint 0.

To enable the hub interrupt:

1. Set the global enable bit (EA) in the IEN0 register (Figure 5-10)
2. Enable the hub endpoint 0 transmit done and receive done interrupts individually:
   a. To enable the receive done interrupt, set the HRXE0 bit in the Hub Interrupt Enable SFR (HIE, as shown in Figure 5-7)
   b. To enable the transmit done interrupt, set the HTXE0 bit in HIE

NOTE

The 8*x*931A*x* microprocessor does not support hub operations or a hub interrupt. Specific details of the 8*x*931A*x* are covered in Appendix E, "8x931AA Design Considerations".

HIE Address: A1H
Reset State: xxxx xx00B

Hub Interrupt Enable Register. Enables and disables the receive and transmit done interrupts for hub endpoint 0.

7							0
—	—	—	—	—	—	HRXE0	HTXE0

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved: Write zeros to these bits.
1	HRXE0	HRXE0: Enable the hub endpoint 0 receive done interrupt (HRXD0).†
0	HTXE0	HTXE0: Enable the hub endpoint 0 transmit done interrupt (HTXD0).†

† A '1' means the interrupt is enabled, and will cause an interrupt to be signaled to the microcontroller. A '0' means the associated interrupt source is disabled and cannot cause an interrupt, even though its value is still reflected in the HIFLG register.

Figure 5-7. HIE: Hub Interrupt Enable Register

The USB Hub Interrupt Flag Register (HIFLG) is shown in Figure 5-8. This register is used to indicate pending hub interrupts. For all bits in HIFLG, a '1' indicates that an interrupt is actively pending; a '0' indicates that the interrupt is not active. The interrupt status is shown in the HIFLG register regardless of the state of the corresponding interrupt enable bit in the HIE register (Figure 5-7).

NOTE

For the 8*x*931HA, the hub interrupt shares an interrupt vector with the SOF interrupt. When this interrupt is triggered, firmware must examine the HIFLG SFR to determine that it was the hub interrupt that was triggered and not the SOF interrupt.

HIFLG

Address: E8H
Reset State: xxxx xx00B

Hub Interrupt Flag Register. Contains the hub's transmit and receive done interrupt flags for hub endpoint 0.

7							0
—	—	—	—	—	—	HRXD0	HTXD0

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved: Write zeros to these bits.
1	HRXD0	Hub Receive Done, Endpoint 0.: Hardware sets this bit to indicate that there is either: (1) valid data waiting to be serviced in the receive data buffer for hub endpoint 0 and that the data was received without error and has been acknowledged; or (2) that data was received with a FIFO error requiring firmware intervention to be cleared.
0	HTXD0	Hub Transmit Done, Endpoint 0: Hardware sets this bit to indicate that one of two conditions exists in the transmit data buffer for hub endpoint 0: (1) the transmit data has been transmitted and the host has sent acknowledgment which was successfully received; or (2) a FIFO-related error occurred during transmission of the data packet, which requires servicing by firmware to be cleared.

NOTES:
1. Note that because the HIFLG appears in the first SFR column, it is a bit-addressable SFR. All bits are set in hardware and cleared by firmware. Firmware can also set these bits for test purposes, allowing the interrupt to be generated by firmware.
2. For both HRXD0 and HTXD0, a '1' indicates that an interrupt is actively pending; a '0' indicates that the interrupt is not active. The interrupt status is shown regardless of the state of the corresponding interrupt enable bit in the HIE.

Figure 5-8. HIFLG: Hub Interrupt Status Register

5.2.8 USB Global Suspend/Resume Interrupt

The 8*x*931 supports USB power control through firmware. The USB power control register (PCON1, as shown in Figure 9-2 on page 9-3) facilitates USB power control of the 8*x*931, including global suspend/resume and USB function resume.

5.2.8.1 Global Suspend

When a global suspend is detected by the 8*x*931, the global suspend bit (GSUS of PCON1) is set, and the global suspend/resume interrupt is generated if IEN0.7 (EA) and IEN1.2 are set. Global suspend is defined as bus inactivity for more than 3 ms on the USB lines. For additional information, see "Global Suspend Mode" on page 14-7.

5.2.8.2 Global Resume

When a global resume is detected by the 8*x*931, the global resume bit (GRSM of PCON1) is set, and the global suspend/resume interrupt is generated if IEN0.7 (EA) and IEN1.2 are set. As soon as resume signaling is detected on the USB lines, the oscillator is restarted. After executing the resume interrupt service routine, the 8*x*931 resumes operation from where it was when it was interrupted by the suspend interrupt. For additional information, see "Global Resume Mode" on page 14-9.

5.2.8.3 USB Remote Wake-up

The 8*x*931 can also initiate resume signaling to the USB lines through remote wake-up of the USB function while it is in powerdown/idle mode. While in powerdown mode, remote wake-up has to be initiated through assertion of an enabled external interrupt. The external interrupt has to be enabled and it must be configured with level trigger and with higher priority than a suspend/resume interrupt. An external interrupt restarts the clocks to the 8*x*931 and program execution branches to the external interrupt service routine.

Within this external interrupt service routine, you must set the remote wakeup bit (RWU in PCON1) to drive resume signaling on the USB lines to the host or upstream hub. After executing the external ISR, the program continues execution from where it was put into powerdown mode and the 8*x*931 resumes normal operation. For additional information, see "USB Remote Wake-up" on page 14-10.

5.2.9 USB Reset Separation

The 8*x*931 features an optional USB reset that functions independently from the chip reset. When the PCON1 SFR's URDIS bit is set, the 8*x*931 core and peripherals will not reset when a USB reset signal is detected. After an 8*x*931 with URDIS set detects a USB reset signal, it resets all the USB blocks (including the USB SFRs), sets the URST bit in PCON1, and generates a USB reset interrupt. USB reset signals can originate only from the host PC or upstream hub.

NOTE

The use of a separate USB reset is recommended only for applications where the device is required to be operated continually, even when the PC is powered-off, as is the case with Computer Telephony Integration (CTI). All

other applications are advised against using the separate USB reset. Leaving the URDIS bit clear will ensure a robust, chip-level reset.

The USB reset must be implemented partially in firmware, including an initialization routine as part of chip start-up. To ensure compliance with USB-specified timing constraints and minimize the potential for data corruption, you must implement flag checking as part of your main routine, subroutines, and ISRs. These requirements increase the complexity of your firmware code.

If the 8*x*931 is in powerdown or suspend mode when the separate USB reset interrupt is generated, the device will wake up from powerdown or suspend mode upon receiving the USB reset signal. The ISR of a bus-powered device must set the LC bit of PCON (Figure 14-1 on page 14-3) in order to operate at 3 MHz. This ensures that the device meets the 100 mA current limit during enumeration, as required by the *Universal Serial Bus Specification*. Self-powered devices (i.e., devices drawing less than 100mA from the USB wires) may choose not to switch to Low Clock mode after detecting the USB reset.

NOTE

If desired, your firmware can handle the separate USB reset without using an ISR. To do this, you must clear the ESR bit in the IEN1 SFR. The USB reset hardware operations will still take place, but the ISR will not be called. That is, step 1 and step 2 under "USB Reset Hardware Operations" on page 5-21 will still occur, but step 3 will not. Your firmware must poll the URST flag periodically to detect the USB reset and take the appropriate action.

Since the global suspend and global resume interrupts share the same interrupt vector as USB reset, your firmware must also poll the GRSM and GSUS bits in PCON1 to detect global suspend or resume.

If instead you choose to implement a separate USB reset using an ISR, follow the procedure outlined in the following subsections and displayed in Figure 5-9.

5.2.9.1 Initialization Required for USB Reset

Because USB reset implementation depends heavily on firmware, your code must perform the following initialization prior to execution of the main routine (See Figure 5-9):

1. To enable the USB reset interrupt on the 8*x*931, your initialization routine must set the following bits to '1':
   a. the EA bit of IEN0 (Figure 5-10)
   b. the ESR bit of IEN1 (Figure 5-11)
   c. the URDIS bit of PCON1 (Figure 14-2 on page 14-4)
2. Use bit 2 of IPH1/IPL1 to set the priority of the USB reset interrupt (See "Interrupt Priorities" on page 5-26).

NOTE

It is recommended that you set the USB reset interrupt to the highest priority.

3. After enabling the USB reset interrupt and assigning it a priority, your initialization routine should clear the USB_RST_FLG flag. This flag is a global variable declared in your firmware, not a bit in an SFR.
   This flag, an indicator that a USB reset has occurred, will be examined at various points in your main routine, subroutines, and ISRs.

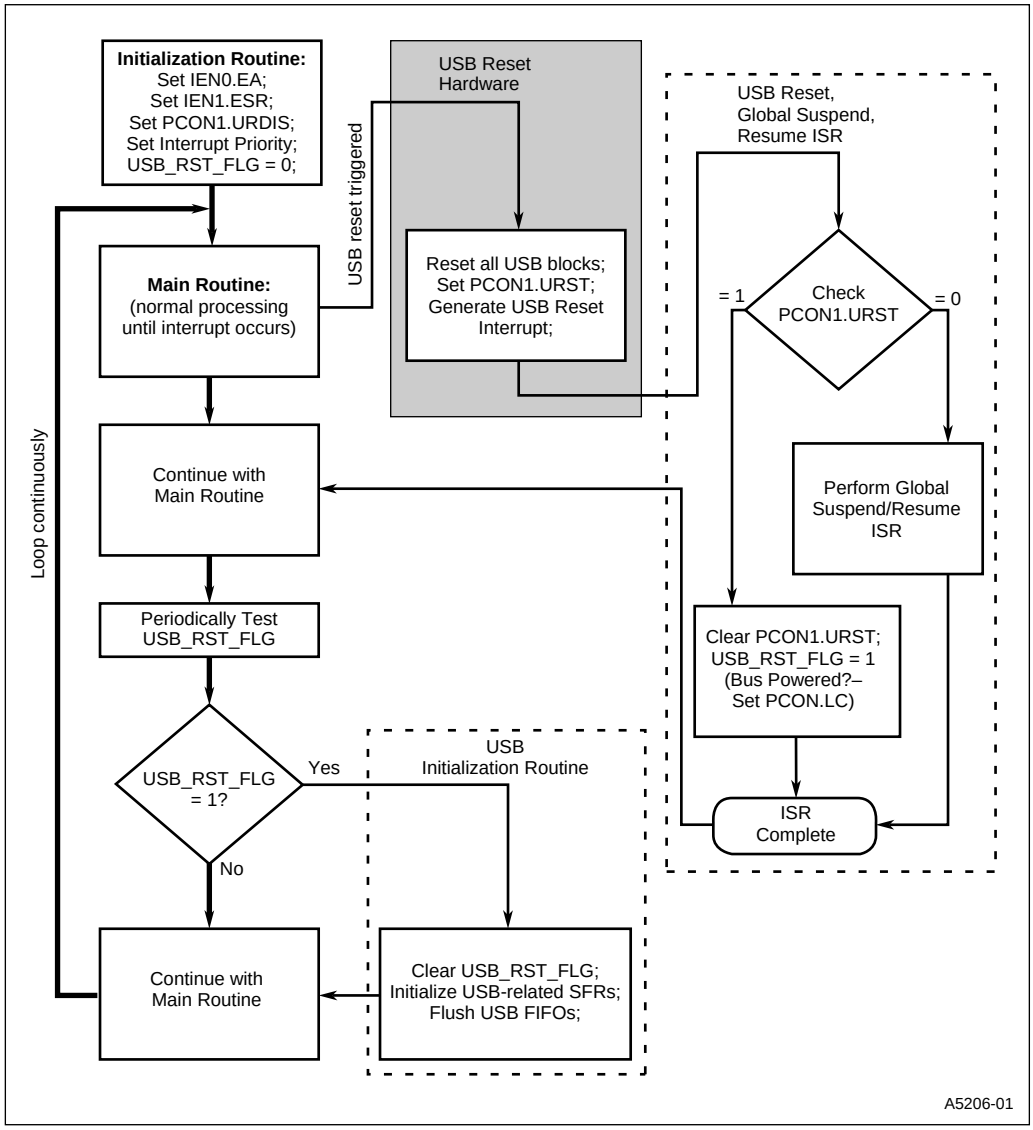


Figure 5-9. USB Reset Separation Operating Model

5.2.9.2 USB Reset Hardware Operations

When the host initiates a USB reset signal, the following series of events is performed by the 8*x*931 hardware (See Figure 5-9):

1. Upon detecting a USB reset signal, the 8*x*931 hardware resets all the USB blocks (i.e., the FIFOs, the SIU, the SIE, and the USB transceiver).

   As a result of this process, all USB-related SFRs are reset to their default reset states. This includes EPINDEX, EPCON, SOFL, SOFH, FIE, FIFLG, FADDR, TXSTAT, TXDAT, TXCON, TXFLG, TXCNTL, TXCNTH, RXSTAT, RXDAT, RXCON, RXFLG, RXCNTL, RXCNTH, and PCON1. Note that PCON1 is only partially reset — the URDIS and URST bits retain their original values.

   Because of this hardware reset, any USB-related operations (e.g., MOV TXDAT,A) will not provide valid data.

2. The 8*x*931 sets the PCON1.URST bit to indicate a USB reset to the ISR.
3. If the ESR bit in IEN1 is set, the 8*x*931 generates a USB reset interrupt, which causes a branch to the interrupt service routine whose vector is located at FF:0053H. This ISR services both the USB reset interrupt and the global suspend/resume interrupt.

5.2.9.3 USB Reset ISR

Because the USB reset interrupt shares an interrupt vector with the USB global suspend/resume interrupt, the interrupt service routine must play a dual role. The ISR must first check PCON1's URST bit to ensure that this interrupt is indeed a USB reset interrupt.

If URST = '0', then this interrupt must be a global suspend/resume interrupt and the ISR must branch to service that type of interrupt. See "USB Global Suspend/Resume Interrupt" on page 5-17 for a description of this portion of the ISR.

If the URST bit is set to '1', then this interrupt is a USB reset interrupt. The ISR must perform the following procedure (See Figure 5-9):

1. Clear PCON1's URST bit — to indicate that the USB reset interrupt has been serviced.
2. Set the user flag USB_RST_FLG that was cleared as part of your initialization routine.

   This flag is discussed in "Initialization Required for USB Reset" on page 5-18. Setting this flag is necessary to inform your firmware routines that a USB reset has occurred and that USB initialization must be performed.

3. Bus-powered devices must set the LC bit of PCON (Figure 14-1 on page 14-3) in order to operate at 3 MHz. This ensures that the device meets the *Universal Serial Bus Specification*'s 100 mA current limit during enumeration.
4. Restore any register values and return from interrupt.

The rest of the USB reset procedure will be initiated by a USB initialization routine that can be called from the main routine, subroutines, or other ISRs.

5.2.9.4 Main Routine Considerations

Although the USB-related SFRs were reset by the USB reset ISR, they must also be initialized by a special USB initialization routine called by the main routine. Since the USB reset interrupt can occur at any time, the only way the main routine will know that a USB reset occurred is to periodically check the USB reset flag (USB_RST_FLG). This is the firmware flag that was set in Step 2 of the "USB Reset ISR" on page 5-21.

When a set reset flag is detected, the main routine branches to a USB initialization routine, which performs the following tasks (See Figure 5-9):

1. Clear the user flag USB_RST_FLG.

   Clearing this flag indicates that USB initialization is not required. Clear this flag first in case a second USB reset occurs during this initialization routine, rendering this initialization invalid.

2. Initialize the USB-related SFRs to the values required by your program.

   If your application requires any other SFRs to be initialized upon USB reset (e.g., SCON), now is the time to do so.

3. Restore any USB-related user flags specific to your application.
4. Flush all USB FIFOs. This is done by setting RXCLR in RXCON and TXCLR in TXCON. This must be done for each function endpoint.
5. Return to the calling routine.

At this point, the main routine can resume normal processing. Eventually, the host PC will transmit a SETUP token. This will trigger an interrupt that will perform USB enumeration.

NOTE

USB specifications require that all devices must be able to accept a device address via a SET_ADDRESS command no later than 10 ms after the reset is removed.

It is recommended that you ensure that the total time required for the following is less than 10ms:

1. The time to complete and exit from the USB reset ISR (accounting for latency — see "Response Time" on page 5-32)
2. The time for the maximum number of instructions that could execute before your code recognizes that a USB reset has occurred (by checking USB_RST_FLG) and calls your USB initialization routine
3. The time to execute your USB initialization routine

This time constraint may require you to check USB_RST_FLG at multiple points in your code (and within any ISRs that may take longer than 10ms to perform). By inserting this checkpoint, your program can branch from a routine (or ISR) after the USB reset without having to complete the routine (or ISR). Your program can continue the interrupted routine after ensuring that the device is ready for USB enumeration.

CAUTION

If a USB reset interrupt occurs during execution of a USB receive ISR (e.g., receive done or start-of-frame), the 8*x*931 will reset the USB hardware. This will render invalid any data received during the USB transfer. If this is not detected by your firmware, misprocessing can occur.

The risk of USB reset-related misprocessing can be reduced if your USB receive/transmit ISRs check USB_RST_FLG before returning. If this flag is set, your code should branch to the USB initialization routine to initialize the USB-related SFRs and flush the FIFOs. If this is done, the only potential opportunity for misprocessing would be if the USB reset interrupt occurs between the test of USB_RST_FLG and the branch to the USB initialization routine.

NOTE

Because of the risk of misprocessing, however slight, it is recommended that applications that will not substantially benefit from a separate USB reset disable this option (by leaving the URDIS bit in PCON1 cleared) to simplify firmware coding and ensure a robust, chip-level reset.

5.3 INTERRUPT ENABLE

Each interrupt source may be individually enabled or disabled by the appropriate interrupt enable bit in the IEN0 register at A8H (see Figure 5-10) or the IEN1 register at B1H (see Figure 5-11). Note IEN0 also contains a global disable bit (EA). If EA is set, interrupts are individually enabled or disabled by bits in IEN0 and IEN1. If EA is clear, all interrupts are disabled.

IEN0 Address: A8H
Reset State: 0000 0000B

Interrupt Enable Register 0. IEN0 contains two types of interrupt enable bits. The global enable bit (EA) enables/disables all of the interrupts (including those in IEN1). The remaining bits enable/disable the other individual interrupts.

7							0
EA	—	ET2	ES	ET1	EX1	ET0	EX0

Bit Number	Bit Mnemonic	Function
7	EA	Global Interrupt Enable: Setting this bit enables all interrupts that are individually enabled by the other bits of this register, as well as the interrupts enabled by the bits in the IEN1 SFR. Clearing this bit disables all interrupts, except the TRAP interrupt, which are always enabled.
6	—	Reserved: Write a zero to this bit.
5	ET2	Timer 2 Overflow Interrupt Enable: Setting this bit enables the timer 2 overflow interrupt.
4	ES	Serial I/O Port Interrupt Enable: Setting this bit enables the serial I/O port interrupt.
3	ET1	Timer 1 Overflow Interrupt Enable: Setting this bit enables the timer 1 overflow interrupt.
2	EX1	External Interrupt 1 Enable: Setting this bit enables external interrupt 1.
1	ET0	Timer 0 Overflow Interrupt Enable: Setting this bit enables the timer 0 overflow interrupt.
0	EX0	External Interrupt 0 Enable: Setting this bit enables external interrupt 0.

NOTE: Note that because IEN0 appears in the first SFR column, it is a bit-addressable SFR.

Figure 5-10. IEN0: USB Interrupt Enable Register 0

IEN1 Address: B1H
Reset State: xxxx x000B

Interrupt Enable Register 1. Contains the enable bits for the USB interrupts.

7							0
EX2	—	—	—	—	ESR	EF	ESOF

Bit Number	Bit Mnemonic	Function
7	EX2	External Interrupt 2 Enable (Keyboard Scan): Setting this bit enables the external interrupt used for the keyboard scan. **NOTE:** Setting this bit causes the 8*x*931 to trigger a hardware interrupt when a keyboard scan interrupt occurs, but only if the KSEN bit in the KBCON register is also set.
6:3	—	Reserved: Write zeros to these bits.
2	ESR	Enable Suspend/Resume/Reset: USB global suspend/resume/reset interrupt enable bit.
1	EF	Enable Function: Transmit/Receive Done interrupt enable bit for non-isochronous USB function endpoints.
0	ESOF	Enable USB Hub/Start of Frame: Any start-of-frame interrupt enable for isochronous endpoints, or USB hub interrupt enable.

Figure 5-11. IEN1: USB Interrupt Enable Register

5.4 INTERRUPT PRIORITIES

The 8*x*931 interrupt sources may be individually programmed to one of four priority levels. This is accomplished with the IPH*X*.*x*/IPL*X*.*x* bit pairs in the interrupt priority high (IPH1/IPH0 in Figure 5-12 and 5-14) and interrupt priority low (IPL1/IPL0) registers (Figures 5-13 and 5-15). Specify the priority level as shown in Table 5-5 using IPH0.*x* (or IPH1.*x*) as the MSB and IPL0.*x* (or IPL1.*x*) as the LSB.

Table 5-5. Level of Priority

Priority Level	IPH1.*x*, IPL1.*x*	IPH0.*x*, IPL0.*x*
0 (Lowest Priority)	00	00
1	01	01
2	10	10
3 (Highest Priority)	11	11

A low-priority interrupt is always interrupted by a higher priority interrupt but not by another interrupt of equal or lower priority. The highest priority interrupt is not interrupted by any other interrupt source. Higher priority interrupts are serviced before lower priority interrupts. The response to simultaneous occurrence of equal priority interrupts (i.e., sampled within the same four-state interrupt cycle) is determined by a hardware priority-within-level resolver (see Table 5-6).

Table 5-6. Interrupt Priority Within Level

Priority Number	Interrupt Name
1 (Highest Priority)	INT0#
2	Timer 0
3	INT1#
4	Timer 1
5	Serial Port
6	Timer 2
7	Keyboard Scan (INT2#)
8	—
9	USB Hub / SOF
10	USB Function
11 (Lowest Priority)	USB Global Suspend/Resume

IPH0

Address: B7H
Reset State: x000 0000B

Interrupt Priority High Control Register 0. IPH0, together with IPL0, assigns each interrupt in IEN0 a priority level from 0 (lowest) to 3 (highest):

IPH0*x*	IPL0*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
—	—	IPH0.5	IPH0.4	IPH0.3	IPH0.2	IPH0.1	IPH0.0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	IPH0.5	Timer 2 Overflow Interrupt Priority Bit High.
4	IPH0.4	Serial I/O Port Interrupt Priority Bit High.
3	IPH0.3	Timer 1 Overflow Interrupt Priority Bit High.
2	IPH0.2	External Interrupt 1 Priority Bit High.
1	IPH0.1	Timer 0 Overflow Interrupt Priority Bit High.
0	IPH0.0	External Interrupt 0 Priority Bit High.

Figure 5-12. IPH0: Interrupt Priority High Register 0

IPL0

Address: B8H
Reset State: x000 0000B

Interrupt Priority Low Control Register 0. IPL0, together with IPH0, assigns each interrupt in IEN0 a priority level from 0 (lowest) to 3 (highest):

IPH0*x*	IPL0*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
—	—	IPL0.5	IPL0.4	IPL0.3	IPL0.2	IPL0.1	IPL0.0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	IPL0.5	Timer 2 Overflow Interrupt Priority Bit Low.
4	IPL0.4	Serial I/O Port Interrupt Priority Bit Low.
3	IPL0.3	Timer 1 Overflow Interrupt Priority Bit Low.
2	IPL0.2	External Interrupt 1 Priority Bit Low.
1	IPL0.1	Timer 0 Overflow Interrupt Priority Bit Low.
0	IPL0.0	External Interrupt 0 Priority Bit Low.

Figure 5-13. IPL0: Interrupt Priority Low Register 0

IPH1

Address: B3H
Reset State: xxxx x000B

Interrupt Priority High Control Register 1. IPH1, together with IPL1, assigns each interrupt in IEN1 a priority level from 0 (lowest) to 3 (highest):

IPH1.*x*	IPL1.*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
IPH1.7	—	—	—	IPH1.3	IPH1.2	IPH1.1	IPH1.0

Bit Number	Bit Mnemonic	Function
7	IPH1.7	Keyboard Scan Interrupt Priority Bit High.
6:3	—	Reserved: Write zeros to these bits.
2	IPH1.2	Global Suspend/Resume/Reset Interrupt Priority Bit High.
1	IPH1.1	USB Function Interrupt Priority Bit High.
0	IPH1.0	USB Hub/SOF Interrupt Priority Bit High.

Figure 5-14. IPH1: Interrupt Priority High Register 1

IPL1

Address: B2H
Reset State: xxxx x000B

Interrupt Priority Low Control Register 1. IPL1, together with IPH1, assigns each interrupt in IEN1 a priority level from 0 (lowest) to 3 (highest):

IPH1.*x*	IPL1.*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
IPL1.7	—	—	—	—	IPL1.2	IPL1.1	IPL1.0

Bit Number	Bit Mnemonic	Function
7	IPL1.7	Keyboard Scan Interrupt Priority Bit Low.
6:3	—	Reserved: Write zeros to these bits.
2	IPL1.2	Global Suspend/Resume/Reset Interrupt Priority Bit Low.
1	IPL1.1	USB Function Interrupt Priority Bit Low.
0	IPL1.0	USB Hub/SOF Interrupt Priority Bit Low.

Figure 5-15. IPL1: Interrupt Priority Low Register 1

5.5 INTERRUPT HANDLING

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. The Timer 2 interrupt cycle is slightly different, as described in the Response Time section. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate-service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.
2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any write to the IENx or IPx registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any write to IENx or IPx, then at least one more instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle and the values polled are the values that were present at S5P2 of the previous machine cycle. If the interrupt flag for a level-sensitive external interrupt is active but not being responded to for one of the above conditions and is not still active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new. The polling cycle/LCALL sequence is illustrated in Figure 5-17.

Note that if an interrupt of a higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in Figure 5-17, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to.

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking interrupt was still in progress.

Note that the starting addresses of consecutive interrupt service routines are only 8 bytes apart.That means if consecutive interrupts are being used (IE0 and TF0 for example, or TF0 and IEl), and if the first interrupt routine is more than 7 bytes long, then that routine will have to execute a jump to some other memory location where the service routine can be completed without overlapping the starting address of the next interrupt routine.

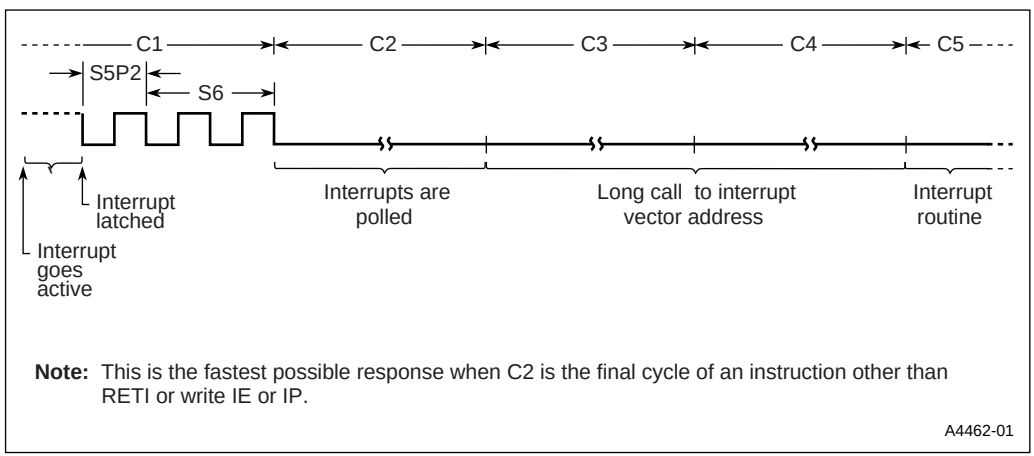


Figure 5-16. Interrupt Response Timing Diagram

5.6 RESPONSE TIME

The INT0# and INT1# levels are inverted and latched into the Interrupt Flags IE0 and IE1 at S5P2 of every machine cycle. Similarly, the Timer 2 flag EXF2 and the serial Port flags RI and TI are set at S5P2. The values are not actually polled by the circuitry until the next machine cycle.

The Timer 0 and Timer 1 flags, TF0 and TFl, are set at S5P2 of the cycle in which the timers overflow. The values are then polled by the circuitry in the next cycle. However, the Timer 2 flag TF2 is set at S2P2 and is polled in the same cycle in which the timer overflows.

If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapses between activation of an external interrupt request and the beginning of execution of the service routine's first instruction. Figure 5-16 shows interrupt response timing.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or write to IENx or IPx, the additional wait time cannot be more than 5 cycles (a maximum of one or more cycles to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV). Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 9 cycles.

CAUTION

It is recommended that programmers set the contents of EPINDEX and/or HPINDEX once at the start of each routine, instead of writing to the EPINDEX register prior to each access of an endpoint-indexed SFR (and to HPINDEX prior to each access of a port-indexed SFR). This means that interrupt service routines must save the contents of the EPINDEX and HPINDEX registers at the start of the routine and restore the contents at the end of the routine to prevent the EPINDEX and HPINDEX registers from being corrupted.

6

USB Function

CHAPTER 6
USB FUNCTION

This chapter describes the FIFOs and special function registers (SFRs) associated with the USB function interface. This chapter (along with Chapter 2, “Architectural Overview” and Chapter 8, “USB Programming Models”) describes the operation of function interface on the 8*x*931 USB microcontroller.

A data flow model for USB transactions, intended to bridge the hardware and firmware layers of the 8*x*931, is presented in truth table form in Appendix D, “Data Flow Model”. The model describes 8*x*931 behavior in response to a particular USB event, given a known state/configuration.

The SFRs described in this chapter are listed in Table 6-3. The SFR definition tables that appear in this chapter also appear in alphabetical order in Appendix C, “Registers”.

6.1 FUNCTION INTERFACE

The function interface provides a USB interface capability for one USB function. The main components of the function interface are the serial bus interface engine (SIE) and the function interface unit (FIU). Refer to the block diagram in Figure 2-3 on page 2-7. The operation of the function interface is discussed in section 2.4, “Universal Serial Bus Module” (pg. 2-11). On the 8*x*931HA, the hub accesses the function interface through the internal downstream port.

6.1.1 Function Endpoint Pairs

The endpoint pairs implemented on the 8*x*931 are listed in Table 2-5 on page 2-12. The EPINDEX register selects the endpoint pair for any given data transaction. The 8*x*931HA supports three function endpoint pairs and two hub endpoint pairs. See “USB Hub Endpoints” on page 7-10.

6.1.2 Function FIFOs

The 8*x*931 provides a transmit/receive FIFO pair for each endpoint pair. Transmit FIFOs are written by the CPU and then read by the FIU for transmission on the USB. Receive FIFOs are written by the FIU following reception from the host PC, then read by the CPU. All transmit FIFOs have the same architecture, and all receive FIFOs have the same architecture. Table 6-1 shows the FIFO size and configuration for the hub and function endpoint pairs.

Table 6-1. Function and Hub FIFO Configurations

Endpoint	Control	Bulk	Interrupt	Isochronous	Dual-packet	RX FIFO Size	TX FIFO Size
Hub EP0	Yes	No	No	No	No	8 bytes	8 bytes
Hub EP1	No	No	Yes	No	No	N/A	1 byte
Function EP0	Yes	No	No	No	No	8 bytes	8 bytes
Function EP1	Yes	Yes	Yes	Yes	Yes	16 bytes	16 bytes
Function EP2	Yes	Yes	Yes	No	No	8 bytes	8 bytes

The USB signals discussed in this chapter are described in Table 6-2. The pinout diagrams for the 8*x*931 appear in Appendix B, "Pin Descriptions".

Table 6-2. Non-hub USB Signal Descriptions

Signal Name	Type	Description	Alternate Function
PLLSEL	I	**Phase-locked Loop Select.** For normal operation, connect PLLSEL to logic high. PLLSEL = 0 is used for factory test (see Table 2-3 on page 2-9).	—
SOF#	O	**Start of Frame.** The SOF# pin is asserted for eight states when an SOF token is received.	—
D_{P0}, D_{M0}	I/O	**USB Port 0**. D_{P0} and D_{M0} are the data plus and data minus lines of differential USB upstream port 0. These lines do not have internal pullup resistors. For low-speed devices, provide an external 1.5 KΩ pullup resistor at D_{M0}. For full-speed devices, provide an external 1.5 KΩ pullup resistor at D_{P0}. **NOTE:** Provide an external 1.5 KΩ pullup resistor at D_{P0} so the device indicates to the host that it is a full-speed device.	—
ECAP	I	**External Capacitor**. Must be connected to a 1 µF capacitor (or larger) to ensure proper operation of the differential line driver. The other lead of the capacitor must be connected to V_{SS}.	—

The FIU controls operations through the use of four sets of special functions registers (SFRs): the FIU SFRs, the transmit FIFO SFRs, the receive FIFO SFRs, and the USB interrupt SFRs. Table 6-3 lists the SFRs described in this chapter. Figure 6-1 provides a quick reference to the bit assignments in the SFRs.

USB interrupt SFRs are described in Chapter 5, "Interrupt System". Table C-1 on page C-2 shows a memory map of all the 8*x*931HA SFRs. Table E-4 on page E-10 shows a memory map of all the 8*x*931AA SFRs.

Table 6-3. USB Function SFRs

Mnemonic	Description	Address	Page
EPCON	**Endpoint Control Register.** Configures the operation of the endpoint specified by EPINDEX.	E1H	page 6-7
EPINDEX	**Endpoint Index Register.** Selects the appropriate endpoint pair.	F1H	page 6-6
FADDR	**Function Address Register.** Stores the USB function address for the device. The host PC assigns the address and informs the device via endpoint 0.	8FH	page 6-14
FIE	**USB Function Interrupt Enable Register.** Enables and disables the receive and transmit done interrupts for the function endpoints.	A2H	page 5-9
FIFLG	**USB Function Interrupt Flag Register.** Contains the USB function's transmit and receive done interrupt flags for non-isochronous endpoints.	C0H	page 5-11
RXCNTL	**Receive FIFO Byte-Count Low Register.** Stores the byte count for the data packets received in the receive FIFO specified by EPINDEX.	E6H	page 6-26
RXCON	**Receive FIFO Control Register.** Controls the receive FIFO specified by EPINDEX.	E4H	page 6-29
RXDAT	**Receive FIFO Data Register.** Receive FIFO data is read from this register (specified by EPINDEX).	E3H	page 6-26
RXFLG	**Receive FIFO Flag Register.** These flags indicate the status of data packets in the receive FIFO specified by EPINDEX.	E5H	page 6-31
RXSTAT	**Endpoint Receive Status Register.** Contains the endpoint status of the receive FIFO specified by EPINDEX.	E2H	page 6-11
TXCNTL	**Transmit Count Low Register.** Stores the byte count for the data packets in the transmit FIFO specified by EPINDEX.	F6H	page 6-16
TXCON	**Transmit FIFO Control Register.** Controls the transmit FIFO specified by EPINDEX.	F4H	page 6-19
TXDAT	**Transmit FIFO Data Register.** Transmit FIFO data is written to this register (specified by EPINDEX).	F3H	page 6-16
TXFLG	**Transmit Flag Register.** These flags indicate the status of data packets in the transmit FIFO specified by EPINDEX.	F5H	page 6-21
TXSTAT	**Endpoint Transmit Status Register.** Contains the endpoint status of the transmit FIFO specified by EPINDEX.	FAH	page 6-9

	7							0
EPCON	RXSTL	TXSTL	CTLEP	RXSPM	RXIE	RXEPEN	TXOE	TXEPEN
EPINDEX	HORF	—	—	—	—	—	EPINX1	EPINX0
FADDR	—	Function Address						
FIE	—	—	FRXIE2	FTXIE2	FRXIE1	FTXIE1	FRXIE0	FTXIE0
FIFLG	—	—	FRXD2	FTXD2	FRXD1	FTXD1	FRXD0	FTXD0
RXCNTL	—	—	—	BC4	BC3	BC2	BC1	BC0
RXCON	RXCLR	—	—	RXFFRC	RXISO	ARM	ADVWM	REVWP
RXDAT	Receive Data Byte							
RXFLG	RXFIF1	RXFIF0	—	—	RXEMP	RXFULL	RXURF	RXOVF
RXSTAT	RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK
TXCNTL	—	—	—	BC4	BC3	BC2	BC1	BC0
TXCON	TXCLR	—	—	—	TXIS0	ATM	ADVRM	REVRP
TXDAT	Transmit Data Byte							
TXFLG	TXFIF1	TXFIF0	—	—	TXEMP	TXFULL	TXURF	TXOVF
TXSTAT	TXSEQ	—	—	TXFLUSH	TXSOVW	TXVOID	TXERR	TXACK

Figure 6-1. Bits of the USB Function SFRs

The registers in the FIU SFR set are: EPINDEX, EPCON, TXSTAT, RXSTAT, SOFL, SOFH, and FADDR. The SOFH and SOFL SFRs are defined in Figure 5-5 on page 5-12 and Figure 5-6 on page 5-13. The remaining registers are defined in Figures 6-2 through 6-6.

The registers in the transmit FIFO SFR set are TXDAT, TXCON, TXFLG, and TXCNTL. These registers are defined in Figures 6-8 through 6-11 beginning on page 6-16.

The registers in the receive FIFO SFR set are RXDAT, RXCON, RXFLG, and RXCNTL. These registers are defined in Figures 6-13 through 6-16 beginning on page 6-26.

The transmit SFR set, the receive SFR set, EPCON, TXSTAT, and RXSTAT are endpoint-indexed

CAUTION

Unless otherwise noted in the bit definition, SFRs can be read and written by firmware. All SFRs should be written using *read-modify-write instructions only*, due to the possibility of simultaneous writes by hardware and firmware. These instructions are listed in “Read-Modify-Write Instructions” on page 9-5.

6.1.3 Endpoint-indexed SFRs

As indicated in the SFR memory maps in Table C-1 on page C-2, certain USB SFRs are endpoint-indexed. These SFRs are implemented as banks of registers. Endpoint-indexed SFRs are accessed by means of the SFR address and the current contents of the EPINDEX register (which selects the appropriate bank).

With the exception of hub endpoint 1, there is a bank of SFRs (TXDAT, TXCON, TXFLG, etc.) for each hub and function endpoint pair. Thus the 8*x*931, with three function endpoint pairs, plus hub endpoint 0, has four TXCON registers. When EPINDEX = 0000 0001, the function endpoint 1 TXCON is accessed. When EPINDEX = 0000 0010, the function endpoint 2 TXCON is accessed. The contents of a given SFR are retained when other endpoints are selected.

Only SFRs necessary for device operation are implemented. For example, since hub endpoint 1 is transmit only, RXDAT for that endpoint is not implemented.

6.1.4 Endpoint Selection

The most significant bit of the endpoint index register (EPINDEX, Figure 6-2) selects hub or function.The low-order bits (EPINX1:0) indicate the endpoint and serve as an index value for selecting the SFR bank. To specify the endpoint pair, write a value of the form Zxxx xYYYB or Zxxx xxYYB to EPINDEX, where Z specifies hub or function and YYY and YY specify the endpoint number.

It is recommended that programmers set the contents of EPINDEX once, at the start of each routine, instead of writing the EPINDEX register prior to each access of an endpoint-indexed SFR. This means that interrupt service routines must save the contents of the EPINDEX register at the start of the routine and restore the contents at the end of the routine to prevent the EPINDEX register from being corrupted.

EPINDEX Address: F1H
Reset State: 1xxx xx00B

Endpoint Index Register. This register identifies the endpoint pair. Its contents select the transmit and receive FIFO pair and serve as an index to endpoint-specific SFRs.

7							0
HORF	—	—	—	—	—	EPINX1	EPINX0

Bit Number	Bit Mnemonic	Function
7	HORF	Hub/Function Bit: 1 = Hub. Selects USB hub FIFOs and SFRs. 0 = Function. Selects USB function FIFOs and SFRs.
6:2	—	Reserved: Write zeros to these bits.
1:0	EPINX1:0	Endpoint Index: **EPINDEX**† 0xxx xx00 Function Endpoint 0 0xxx xx01 Function Endpoint 1 0xxx xx10 Function Endpoint 2 **EPINDEX**† 1xxx xx00 Hub Endpoint 0 1xxx xx01 Hub Endpoint 1

† The EPINDEX register identifies the endpoint pair and selects the associated transmit and receive FIFO pair. The value in this register plus SFR addresses select the associated bank of endpoint-indexed SFRs (TXDAT, TXCON, TXFLG, TXCNTL, RXDAT, RXCON, RXFLG, RXCNTL, EPCON, TXSTAT, and RXSTAT).

Figure 6-2. EPINDEX: Endpoint Index Register

6.2 USB FUNCTION SFRS

This section contains the special function registers (SFRs) used by the 8*x*931 USB function.

EPCON (Endpoint-indexed) Address: E1H
Reset State: Endpoint 0 0011 0101B; Function Endpoints 1, 2 0001 0000B

Endpoint Control Register. This SFR configures the operation of the endpoint specified by EPINDEX.

7							0
RXSTL	TXSTL	CTLEP	RXSPM	RXIE	RXEPEN	TXOE	TXEPEN

Bit Number	Bit Mnemonic	Function
7	RXSTL	Stall Receive Endpoint: Set this bit to stall the receive endpoint. Clear this bit only when the host has intervened through commands sent down endpoint 0. When this bit is set and RXSETUP is clear, the receive endpoint will respond with a STALL handshake to a valid OUT token. When this bit is set and RXSETUP is set, the receive endpoint will NAK. This bit does not affect the reception of SETUP tokens by a control endpoint.
6	TXSTL	Stall Transmit Endpoint: Set this bit to stall the transmit endpoint. This bit should be cleared only when the host has intervened through commands sent down endpoint 0. When this bit is set and RXSETUP is clear, the receive endpoint will respond with a STALL handshake to a valid IN token. When this bit is set and RXSETUP is set, the receive endpoint will NAK.
5	CTLEP	Control Endpoint:† Set this bit to configure the endpoint as a control endpoint. Only control endpoints are capable of receiving SETUP tokens.
4	RXSPM	Receive Single Packet Mode:† Set this bit to configure the receive endpoint for single data packet operation. When enabled, only a single data packet is allowed to reside in the receive FIFO. **NOTE:** For control endpoints (CTLEP=1), this bit should be set for single packet mode operation as the recommended firmware model. However, it is possible to have a control endpoint configured in dual packet mode as long as the firmware handles the endpoint correctly.
3	RXIE	Receive Input Enable: Set this bit to enable data from the USB to be written into the receive FIFO. If cleared, the endpoint will not write the received data into the receive FIFO and at the end of reception, but will return a NAK handshake on a valid OUT token if the RXSTL bit is not set.This bit does not affect a valid SETUP token. A valid SETUP token and packet override this bit if it is cleared, and place the receive data in the FIFO.

† For hub endpoint 0 (EPINDEX = 1000 0000), bits 5 and 4 are hard-wired to '1' since hub endpoint 0 is always a control endpoint.

Figure 6-3. EPCON: Endpoint Control Register

EPCON (Continued) Address: E1H
(Endpoint-indexed) Reset State: Endpoint 0 0011 0101B
Function Endpoints 1, 2 0001 0000B

Endpoint Control Register. This SFR configures the operation of the endpoint specified by EPINDEX.

7							0
RXSTL	TXSTL	CTLEP	RXSPM	RXIE	RXEPEN	TXOE	TXEPEN

Bit Number	Bit Mnemonic	Function
2	RXEPEN	Receive Endpoint Enable: Set this bit to enable the receive endpoint. When disabled, the endpoint does not respond to a valid OUT or SETUP token. This bit is hardware read-only and has the highest priority among RXIE and RXSTL. Note that endpoint 0 is enabled for reception upon reset.
1	TXOE	Transmit Output Enable: This bit is used to enable the data in TXDAT to be transmitted. If cleared, the endpoint returns a NAK handshake to a valid IN token if the TXSTL bit is not set.
0	TXEPEN	Transmit Endpoint Enable: This bit is used to enable the transmit endpoint. When disabled, the endpoint does not respond to a valid IN token. This bit is hardware read only. Note that endpoint 0 is enabled for transmission upon reset.

† For hub endpoint 0 (EPINDEX = 1000 0000), bits 5 and 4 are hard-wired to '1' since hub endpoint 0 is always a control endpoint.

Figure 6-3. EPCON: Endpoint Control Register (Continued)

TXSTAT Address: F2H
(Endpoint-indexed) Reset State: 0xx0 0000B

Endpoint Transmit Status Register. Contains the current endpoint status of the transmit FIFO specified by EPINDEX.

7							0
TXSEQ	—	—	TXFLUSH	TXSOVW	TXVOID	TXERR	TXACK

Bit Number	Bit Mnemonic	Function
7	TXSEQ	Transmitter's Current Sequence Bit (read, conditional write): † This bit will be transmitted in the next PID and toggled on a valid ACK handshake. This bit is toggled by hardware on a valid SETUP token. This bit can be written by firmware if the TXSOVW bit is set when written together with the new TXSEQ value.
6:5	—	Reserved: Write zeros to these bits.
4	TXFLUSH	Transmit FIFO Packet Flushed (read-only): When set, this bit indicates that hardware flushed a stale ISO data packet from the transmit FIFO due to a TXFIF1:0 = 11 at SOF. To guard against a missed IN token in ISO mode, if, with TXFIF1:0 = 11, no IN token is received for the current endpoint, hardware automatically flushes the oldest packet and decrements the TXFIF1:0 value.
3	TXSOVW	Transmit Data Sequence Overwrite Bit: † Write a '1' to this bit to allow the value of the TXSEQ bit to be overwritten. Writing a '0' to this bit has no effect on TXSEQ. This bit always returns '0' when read. ††
2	TXVOID	Transmit Void (read-only): ††† A void condition has occurred in response to a valid IN token. Transmit void is closely associated with the NAK/STALL handshake returned by the function after a valid IN token, due to the conditions that cause the transmit FIFO to be unenabled or not ready to transmit. Use this bit to check any NAK/STALL handshake returned by the function. This bit does not affect the FTXD***x***, TXERR or TXACK bits. This bit is updated by hardware at the end of a non-isochronous transaction in response to a valid IN token. For isochronous transactions, this bit is not updated until the next SOF.

† Under normal operation, this bit should not be modified by the user.

†† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

††† For additional information on the operation of these bits see Appendix D, "Data Flow Model."

Figure 6-4. TXSTAT: Transmit FIFO Status Register

TXSTAT (Continued) Address: F2H
(Endpoint-indexed) Reset State: 0xx0 0000B

Endpoint Transmit Status Register. Contains the current endpoint status of the transmit FIFO specified by EPINDEX.

7							0
TXSEQ	—	—	TXFLUSH	TXSOVW	TXVOID	TXERR	TXACK

Bit Number	Bit Mnemonic	Function
1	TXERR	Transmit Error (read-only):†† An error condition has occurred with the transmission. Complete or partial data has been transmitted. The error can be one of the following: 1. Data transmitted successfully but no handshake received 2. Transmit FIFO goes into underrun condition while transmitting The corresponding transmit done bit, FTXD***x*** in FIFLG, is set when active. For non-isochronous transactions, this bit is updated by hardware along with the TXACK bit at the end of the data transmission (this bit is mutually exclusive with TXACK). For isochronous transactions, this bit is not updated until the next SOF.
0	TXACK	Transmit Acknowledge (read-only):†† Data transmission completed and acknowledged successfully. The corresponding transmit done bit, FTXD***x*** in FIFLG, is set when active. For non-isochronous transactions, this bit is updated by hardware at the end of data transmission (along with the TXERR bit — this bit is mutually exclusive with TXERR). For isochronous transactions, this bit is not updated until the next SOF.

† Under normal operation, this bit should not be modified by the user.

†† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

††† For additional information on the operation of these bits see Appendix D, "Data Flow Model."

Figure 6-4. TXSTAT: Transmit FIFO Status Register (Continued)

RXSTAT Address: E2H
(Endpoint-indexed) Reset State: 0000 0000B

Endpoint Receive Status Register. Contains the current endpoint status of the receive FIFO specified by EPINDEX.

7							0
RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK

Bit Number	Bit Mnemonic	Function
7	RXSEQ	Receiver Endpoint Sequence Bit (read, conditional write): This bit will be toggled on completion of an ACK handshake in response to an OUT token. This bit will be set (or cleared) by hardware after reception of a SETUP token. This bit can be written by firmware if the RXSOVW bit is set when written along with the new RXSEQ value. **NOTE:** Always verify this bit after writing to ensure that there is no conflict with hardware, which could occur if a new SETUP token is received.
6	RXSETUP	Received Setup Token: This bit is set by hardware when a valid SETUP token has been received. When set, this bit causes received IN or OUT tokens to be NAKed until the bit is cleared to allow proper data management for the transmit and receive FIFOs from the previous transaction. IN or OUT tokens are NAKed even if the endpoint is stalled (RXSTL or TXSTL) to allow a control transaction to clear a stalled endpoint. Clear this bit upon detection of a SETUP token after the firmware is ready to complete the status stage of a control transaction.
5	STOVW	Start Overwrite Flag (read-only): Set by hardware upon receipt of a SETUP token for any control endpoint to indicate that the receive FIFO is being overwritten with new SETUP data. When set, the FIFO state (FIF and read pointer) resets and is locked for this endpoint until EDOVW is set. This prevents a prior, ongoing firmware read from corrupting the read pointer as the receive FIFO is being cleared and new data is being written into it. This bit is cleared by hardware at the end of handshake phase transmission of the setup stage. This bit is used only for control endpoints.

† Under normal operation, this bit should not be modified by the user.

†† For additional information on the operation of these bits see Appendix D, “Data Flow Model.”

††† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

Figure 6-5. RXSTAT: Receive FIFO Status Register

RXSTAT (Continued) Address: E2H
(Endpoint-indexed) Reset State: 0000 0000B

Endpoint Receive Status Register. Contains the current endpoint status of the receive FIFO specified by EPINDEX.

7							0
RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK

Bit Number	Bit Mnemonic	Function
4	EDOVW	End Overwrite Flag: This flag is set by hardware during the handshake phase of a SETUP stage. It is set after every SETUP packet is received and *must* be cleared prior to reading the contents of the FIFO. When set, the FIFO state (FIF and read pointer) remains locked for this endpoint until this bit is cleared. This prevents a prior, ongoing firmware read from corrupting the read pointer after the new data has been written into the receive FIFO. This bit is only used for control endpoints. **NOTE:** Make sure the EDOVW bit is cleared prior to reading the contents of the FIFO.
3	RXSOVW	Receive Data Sequence Overwrite Bit: † Write a '1' to this bit to allow the value of the RXSEQ bit to be overwritten. Writing a '0' to this bit has no effect on RXSEQ. This bit always returns '0' when read. †††
2	RXVOID	Receive Void Condition (read-only):†† This bit is set when no valid data is received in response to a SETUP or OUT token due to one of the following conditions: 1. The receive FIFO is still locked. 2. The EPCON register's RXSTL bit is set. This bit is set and cleared by hardware. For non-isochronous transactions, this bit is updated by hardware at the end of the transaction in response to a valid OUT token. For isochronous transactions, it is not updated until the next SOF.

† Under normal operation, this bit should not be modified by the user.

†† For additional information on the operation of these bits see Appendix D, "Data Flow Model."

††† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

Figure 6-5. RXSTAT: Receive FIFO Status Register (Continued)

RXSTAT (Continued) Address: E2H
(Endpoint-indexed) Reset State: 0000 0000B

Endpoint Receive Status Register. Contains the current endpoint status of the receive FIFO specified by EPINDEX.

7							0
RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK

Bit Number	Bit Mnemonic	Function
1	RXERR	Receive Error (read-only):†† Set when an error condition has occurred with the reception. Complete or partial data has been written into the receive FIFO. No handshake is returned. The error can be one of the following conditions: 1. Data failed CRC check. 2. Bit stuffing error. 3. A receive FIFO goes into overrun or underrun condition while receiving. This bit is updated by hardware at the end of a valid SETUP or OUT token transaction (non-isochronous) or at the next SOF on each valid OUT token transaction (isochronous). The corresponding FRXD*x* bit of FIFLG is set when active. This bit is updated with the RXACK bit at the end of data reception and is mutually exclusive with RXACK.
0	RXACK	Receive Acknowledged (read-only):†† This bit is set when data is received completely into a receive FIFO and an ACK handshake is sent. This read-only bit is updated by hardware at the end of a valid SETUP or OUT token transaction (non-isochronous) or at the next SOF on each valid OUT token transaction (isochronous). The corresponding FRXD*x* bit of FIFLG is set when active. This bit is updated with the RXERR bit at the end of data reception and is mutually exclusive with RXERR.

† Under normal operation, this bit should not be modified by the user.

†† For additional information on the operation of these bits see Appendix D, “Data Flow Model.”

††† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

Figure 6-5. RXSTAT: Receive FIFO Status Register (Continued)

FADDR Address: 8FH
Reset State: 0000 0000B

Function Address Register. This SFR holds the address for the USB function. During bus enumeration it is written with a unique value assigned by the host.

7							0
—	Function Address						

Bit Number	Bit Mnemonic	Function
7	—	Reserved: Write a zero to this bit.
6:0	A6:0	7-bit Programmable Function Address: This register is programmed through the commands received via endpoint 0 on configuration, which should be the only time the firmware should change the value of this register. This register is hardware read-only.

Figure 6-6. FADDR: Function Address Register

6.3 TRANSMIT FIFOS

The 8*x*931 has a transmit FIFO for each function endpoint pair. In this manual, the term "transmit FIFO" refers to the transmit FIFO associated with the current endpoint pair specified by the EPINDEX register. 8*x*931 FIFOs are listed in Table 2-4.

The transmit FIFOs are circulating data buffers with the following features:

- endpoint 1 supports up to two separate data sets of variable sizes†
- a byte count register to store the number of bytes in the data sets
- protection against overwriting data in a full FIFO
- capability to retransmit the current data set

All transmit FIFOs have the same architecture (Figure 6-7). The transmit FIFO and its associated logic can manage up to two data sets, data set 0 (ds0) and data set 1 (ds1). The ability to have two data sets in the FIFO supports back-to-back transmissions.

† When operating in dual packet mode, the maximum packet size should be, at a maximum, half the FIFO size to ensure both packets will simultaneously fit in the FIFO (see the Endpoint description in the *Universal Serial Bus Specification*).

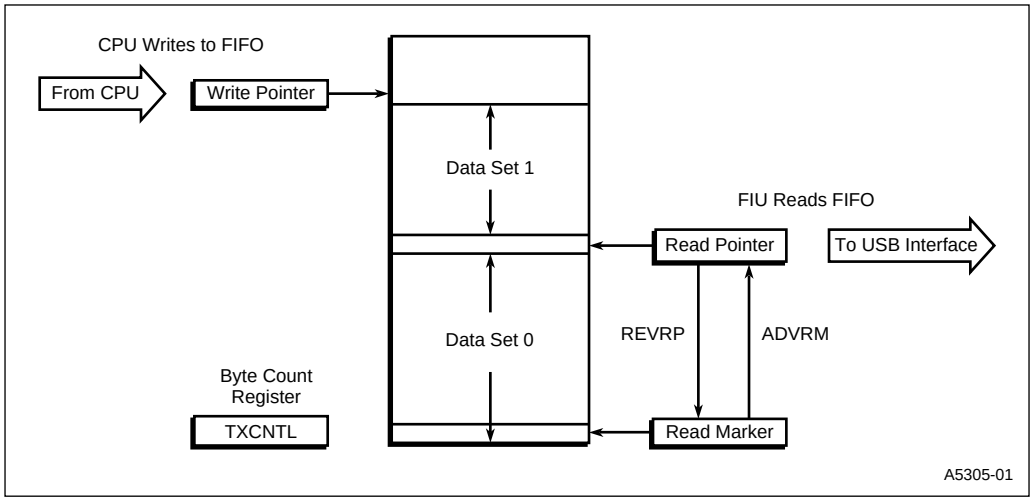


Figure 6-7. Transmit FIFO Outline

The transmit process uses a write pointer, as well as a read pointer and a read marker. The CPU writes to the FIFO location specified by the *write pointer*, which automatically increments by one after a write. The *read marker* points to the first byte of data written to a data set, and the *read pointer* points to the next FIFO location to be read by the function interface. The read pointer automatically increments by one after a read.

When a good transmission is completed, the read marker can be advanced to the position of the read pointer to set up for reading the next data set. When a bad transmission is completed, the read pointer can be reversed to the position of the read marker to enable the function interface to re-read the last data set for retransmission. The read marker advance and read pointer reversal can be accomplished two ways: explicitly by firmware or automatically by hardware, as specified by bits in the transmit FIFO control register (TXCON).

6.3.1 Transmit FIFO Registers

There are five registers directly involved in the operation of the transmit FIFOs:

- TXDAT, the transmit FIFO data register
- TXCNTL, the transmit FIFO byte count register
- TXCON, the transmit FIFO control register
- TXFLG, the transmit FIFO flag register

These registers are endpoint indexed. This means they are used as a set to control the operation of the transmit FIFO associated with the current endpoint, as specified by the EPINDEX register. Figures 6-8 through 6-11 beginning on page 6-16 describe the transmit FIFO registers and provide bit definitions.

6.3.2 Transmit FIFO Data Register (TXDAT)

Bytes are written to the transmit FIFO via TXDAT, the transmit FIFO data register (Figure 6-8).

TXDAT Address: F3H
(Endpoint-indexed)† Reset State: xxxx xxxxB

USB Transmit FIFO Data Register. Data to be transmitted by the FIFO specified by EPINDEX is first written to this register.

7							0
Transmit Data Byte							

Bit Number	Bit Mnemonic	Function
7:0	TXDAT7:0	Transmit Data Byte (write-only): To write data to the transmit FIFO, write to this register. The write pointer is incremented automatically after a write.

† For hub endpoint 1, TXDAT is used in a different manner. See Figure 7-7 on page 7-12.

Figure 6-8. TXDAT: Transmit FIFO Data Register

6.3.3 Transmit FIFO Byte Count Register (TXCNTL)

The transmit FIFO byte count register is used as a five-bit ring buffer, as shown in Figure 6-9.

TXCNTL† Address: F6H
(Endpoint-indexed) Reset States: xxxx xxxxB

Transmit FIFO Byte-count Register. Ring buffer used to store the byte count for the data packets in the transmit FIFO specified by EPINDEX.

7							**0**
—	—	—	BC4	BC3	BC2	BC1	BC0

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved. Write zeros to these bits.
4:0	BC4:0	Transmit Byte Count. (write, conditional read††) Five-bit, ring buffer. Stores transmit byte count for endpoints 0 and 2.

† Byte count registers are not implemented for hub endpoint 1.

†† Read these bits only if TXFIF1:0 = 0; otherwise underrun errors may occur.

Figure 6-9. TXCNTL: Transmit FIFO Byte Count Register

TXCNTL stores the number of bytes in either of the two data sets, data set 0 (ds0) and data set 1 (ds1). The FIFO logic for maintaining the data sets assumes that data is written to the FIFO in the following sequence:

1. The CPU writes data bytes to TXDAT.
2. The CPU writes the number of bytes that were written to TXDAT to the byte count register TXCNTL. TXCNTL must be written after the write to TXDAT to guarantee data integrity.

The function interface reads the byte count register to determine the number of bytes in the set.

6.3.4 Transmit Data Set Management

Two read-only data set index bits, FIF1:0 in the TXFLG register, indicate which data sets (ds0 and/or ds1) have been written into the FIFO and are armed (ready for transmission). See the left side of Table 6-4. FIF*x* = 1 indicates that data set *x* has been written and is armed. Following reset, FIF1:0 = 00, signifying an empty FIFO. FIF1:0 also determine which data set is written next. Note that FIF0 specifies the next data set to be written, except for the case of FIF1:0 = 11. In this case further writes to TXDAT or TXCNTL are ignored.

NOTE

To simplify firmware development, it is recommended that you utilize control endpoints in single-packet mode only.

Two events cause the data set index bits to be updated:

- A new data set is written to the FIFO: The 8*x*931 writes bytes to the FIFO via TXDAT and writes the number of bytes to TXCNTL. The data set index bits are updated after the write to TXCNTL. This process is illustrated in Table 6-4.
- A data set in the FIFO is successfully transmitted: The function interface reads a data set from the FIFO, and when a good transmission is acknowledged, the read marker is advanced to the read pointer. The data set index bits are updated after the read marker is advanced. Note that in ISO mode, this happens at the next start-of-frame (SOF).

Table 6-4. Writing to the Byte Count Register

FIF1:0		Data Sets Written			Set for Next Write to TXCNTL	Write bytes to TXDAT.		FIF1:0	
		ds1	ds0						
0	0	No	No	(Empty)	ds0	—> Write byte count to TXCNTL.	—>	0	1
0	1	No	Yes	(1 set)	ds1			1	1
1	0	Yes	No	(1 set)	ds0			1	1
1	1	Yes	Yes	(2 sets)	Write ignored			1	1

Table 6-5 summarizes how the actions following a transmission depend on the TXISO, ATM, TXACK, and TXERR bits.

Table 6-5. Truth Table for Transmit FIFO Management

TXISO (TXCON.3)	ATM (TXCON.2)	TXERR (TXSTAT.1)	TXACK (TXSTAT.0)	Action at End of Transfer Cycle
X	X	0	0	No operation.
X	0	0	1	Read marker, read pointer, and TXFIF bits remain unchanged. Managed by firmware.
X	0	1	0	Read marker, read pointer, and TXFIF bits remain unchanged. Managed by firmware.
0	1	0	1	Read marker advanced automatically.The TXFIF bit for the corresponding data set is cleared.
0	1	1	0	Read pointer reversed automatically. The TXFIF bit for the corresponding data set remains unchanged.
1	1	X	X	Read marker advanced automatically.The TXFIF bit for the corresponding data set is cleared at the SOF.

NOTE

For normal operation, set the ATM bit in TXCON. Hardware will automatically control the read pointer and read marker, and track the TXFIF bits.

NOTE

To send a status stage after a control write or no data control command or a null packet, write 0 to TXCNTL.

TXCON
(Endpoint-indexed)

Address: F4H
Reset State: 0xxx 0100B

USB Transmit FIFO Control Register. Controls the transmit FIFO specified by EPINDEX.

7							0
TXCLR	—	—	—	TXISO	ATM	ADVRM	REVRP

Bit Number	Bit Mnemonic	Function
7	TXCLR	Transmit Clear: Setting this bit flushes the transmit FIFO, resets all the read/write pointers and markers, sets the EMPTY bit in TXFLG, and clears all other bits in TXFLG. After the flush, hardware clears this bit. Setting this bit does not affect the ATM, TXISO, and FFSZ bits, or the TXSEQ bit in the TXSTAT register.
6:4	—	Reserved: Values read from this bit are indeterminate. Write zeros to these bits.
3	TXISO	Transmit Isochronous Data: Firmware sets this bit to indicate that the transmit FIFO contains isochronous data. The FIU uses this bit to set up the handshake protocol at the end of a transmission. This bit is not reset when TXCLR is set and must be cleared by firmware.
2	ATM	Automatic Transmit Management: Setting this bit (the default value) causes the read pointer and read marker to be adjusted automatically as indicated: **TXISO TX Status Read Pointer Read Marker** X ACK Unchanged Advanced (1) 0 NAK Reversed (2) Unchanged 1 NAK Unchanged Advanced (1) (1) to origin of next data set (2) to origin of the data set last read This bit should always be set, except for test purposes. Setting this bit disables ADVRM and REVRP. This is a sticky bit that is not reset when TXCLR is set, but can be set and cleared by firmware. Hardware neither clears nor sets this bit.

† ATM mode is recommended. ADVRM and REVRP, which control the read marker and read pointer when ATM = 0, are used for test purposes.

Figure 6-10. TXCON: Transmit FIFO Control Register

TXCON (Continued) Address: F4H
(Endpoint-indexed) Reset State: 0xxx 0100B

USB Transmit FIFO Control Register. Controls the transmit FIFO specified by EPINDEX.

7							0
TXCLR	—	—	—	TXISO	ATM	ADVRM	REVRP

Bit Number	Bit Mnemonic	Function
1	ADVRM	Advance Read Marker Control (non-ATM mode only)†: Setting this bit prepares for the next packet transmission by advancing the read marker to the origin of the next data packet (the position of the read pointer). Hardware clears this bit after the read marker is advanced. This bit is effective only when the REVRP, ATM, and TXCLR bits are all clear.
0	REVRP	Reverse Read Pointer Control (non-ATM mode only)†: In the case of a bad transmission, the same data stack may need to be available for retransmit. Setting this bit reverses the read pointer to point to the origin of the last data set (the position of the read marker) so that the FIU can reread the last set for retransmission. Hardware clears this bit after the read pointer is reversed. This bit is effective only when the ADVRM, ATM, and TXCLR bits are all clear.

† ATM mode is recommended. ADVRM and REVRP, which control the read marker and read pointer when ATM = 0, are used for test purposes.

Figure 6-10. TXCON: Transmit FIFO Control Register (Continued)

TXFLG Address: F5H
(Endpoint-indexed) Reset State: 00xx 1000B

Transmit FIFO Flag Register. These flags indicate the status of data packets in the transmit FIFO specified by EPINDEX.

7							0
TXFIF1	TXFIF0	—	—	TXEMP	TXFULL	TXURF	TXOVF

Bit Number	Bit Mnemonic	Function
7:6	TXFIF1:0	FIFO Index Flags (read-only): These flags indicate which data sets are present in the transmit FIFO. The FIF bits are set in sequence after each write to TXCNTL to reflect the addition of a data set. Likewise, TXFIF1 and TXFIF0 are cleared in sequence after each advance of the read marker to indicate that the set is effectively discarded. The bit is cleared whether the read marker is advanced by firmware (setting ADVRM) or automatically by hardware (ATM = 1). The next-state table for the TXFIF bits is shown below:

TXFIF1:0	Operation	Flag	Next TXFIF1:0	Next Flag
00	Wr TXCNTL	X	01	Unchanged
01	Wr TXCNTL	X	11	Unchanged
10	Wr TXCNTL	X	11	Unchanged
11	Wr TXCNTL	X	11	TXOVF = 1
00	Adv RM	X	00	Unchanged
01	Adv RM	X	00	Unchanged
11	Adv RM	X	10/01	Unchanged
10	Adv RM	X	00	Unchanged
XX	Rev RP	X	Unchanged	Unchanged

In ISO mode, TXOVF, TXURF, and TXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. TXFIF is “incremented” by firmware and “decremented” by the USB. Therefore, writes to TXCNTL “increment” TXFIF immediately. However, a successful USB transaction any time within a frame “decrements” TXFIF only at SOF. You must check the TXFIF flags before and after writes to the transmit FIFO and TXCNTL for traceability. See the TXFLUSH bit in TXSTST.

NOTE: To simplify firmware development, configure control endpoints in single-packet mode.

5:4	—	Reserved: Values read from these bits are indeterminate. Write zeros to these bits.

† When set, all transmissions are NAKed.

Figure 6-11. TXFLG: Transmit FIFO Flag Register

TXFLG (Continued) Address: F5H
(Endpoint-indexed) Reset State: 00xx 1000B

Transmit FIFO Flag Register. These flags indicate the status of data packets in the transmit FIFO specified by EPINDEX.

7							0
TXFIF1	TXFIF0	—	—	TXEMP	TXFULL	TXURF	TXOVF

Bit Number	Bit Mnemonic	Function
3	TXEMP	Transmit FIFO Empty Flag (read-only): Hardware sets this bit when the write pointer has not rolled over and is at the same location as the read pointer. Hardware clears this bit when the pointers are at different locations. Regardless of ISO or non-ISO mode, this bit always tracks the current transmit FIFO status.
2	TXFULL	Transmit FIFO Full Flag (read-only): Hardware sets this bit when the write pointer has rolled over and equals the read marker. Hardware clears this bit when the full condition no longer exists. Regardless of ISO or non-ISO mode, this bit always tracks the current transmit FIFO status. Check this bit to avoid causing a TXOVF condition.

† When set, all transmissions are NAKed.

Figure 6-11. TXFLG: Transmit FIFO Flag Register (Continued)

TXFLG (Continued) Address: F5H
(Endpoint-indexed) Reset State: 00xx 1000B

Transmit FIFO Flag Register. These flags indicate the status of data packets in the transmit FIFO specified by EPINDEX.

7							0
TXFIF1	TXFIF0	—	—	TXEMP	TXFULL	TXURF	TXOVF

Bit Number	Bit Mnemonic	Function
1	TXURF	Transmit FIFO Underrun Flag (read, clear only)†: Hardware sets this flag when an additional byte is read from an empty transmit FIFO or TXCNTL. This is caused when the value written to TXCNTL is greater than the number of bytes written to TXDAT. This is a sticky bit that must be cleared through firmware. When this flag is set, the FIFO is in an unknown state, thus it is recommended that you reset the FIFO in your error management routine using the TXCLR bit in TXCON. When the transmit FIFO underruns, the read pointer will not advance — it remains locked in the empty position. If the byte count in TXCNTL does not agree with the data, hardware sets TXURF. This indicates that the transmitted data was corrupted by a bit-stuffing or CRC error. In ISO mode, TXOVF, TXURF, and TXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since underrun can only be caused by USB, TXURF is updated at the next SOF regardless of where the underrun occurs in the frame.
0	TXOVF	Transmit FIFO Overrun Flag†: This bit is set when an additional byte is written to a full FIFO or full TXCNTL with TXFIF1:0 = 11. This is a sticky bit that must be cleared through firmware. When this bit is set, the FIFO is in an unknown state, thus it is recommended that you reset the FIFO in your error management routine using the TXCLR bit in TXCON. When the receive FIFO overruns, the write pointer will not advance — it remains locked in the full position. Check this bit after loading the FIFO prior to writing the byte count register. In ISO mode, TXOVF, TXURF, and TXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since overrun can only be caused by firmware, TXOVF is updated immediately. Check the TXOVF flag after writing to the transmit FIFO before writing to TXCNTL.

† When set, all transmissions are NAKed.

Figure 6-11. TXFLG: Transmit FIFO Flag Register (Continued)

6.4 RECEIVE FIFOs

The 8*x*931 has a receive FIFO for each function endpoint pair. In this manual, the term "receive FIFO" refers to the receive FIFO associated with the current endpoint pair specified by the EPINDEX register. 8*x*931 FIFOs are listed in Table 2-4.

The receive FIFOs are circulating data buffers with the following features:

- endpoint 1 supports up to two separate data sets of variable sizes†
- a byte count register that accesses the number of bytes in the data sets
- flags to signal a full FIFO and an empty FIFO
- capability to re-receive the last data set

Figure 6-12 illustrates a receive FIFO. A receive FIFO and its associated logic can manage up to two data sets: data set 0 (ds0) and data set 1 (ds1). The ability to have two data sets in the FIFO supports back-to-back receptions.

In many ways, the receive FIFO is symmetrical to the transmit FIFO. The FIU writes to the FIFO location specified by the *write pointer*, which increments by one automatically following a write. The *write marker* points to the first byte of data written to a data set, and the *read pointer* points to the next FIFO location to be read by the 8*x*931. The read pointer increments by one automatically following a read.

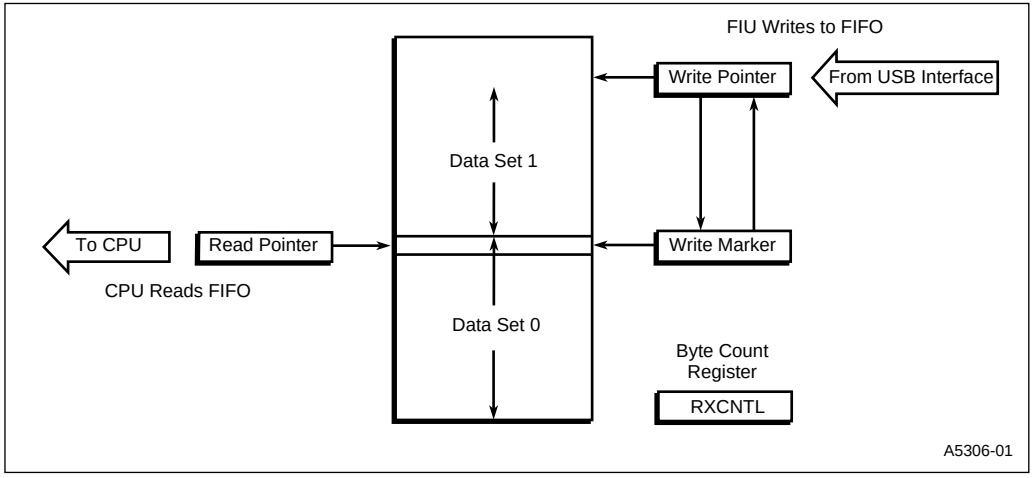


Figure 6-12. Receive FIFO

When a good reception is completed, the write marker can be advanced to the position of the write pointer to set up for writing the next data set. When a bad reception is completed, the write pointer

† When operating in dual packet mode, the maximum packet size should be at most half the FIFO size to ensure that both packets will simultaneously fit in the FIFO (see the Endpoint description in the *Universal Serial Bus Specification*).

can be reversed to the position of the write marker to enable the FIU to rewrite the last data set after receiving the data again. The write marker advance and write pointer reversal can be accomplished two ways: explicitly by firmware or automatically by hardware, as specified by bits in the receive FIFO control register.

The 8*x*931 should not read data from the receive FIFO before all bytes are received and successfully acknowledged because the reception may be bad.

The FIU can monitor the FIFO full flag (RXFULL bit in RXFLG) to avoid overwriting data in the receive FIFO. In the single packet mode, 8*x*931 can monitor the FIFO empty flag (RXEMP bit in RXFLG) to avoid reading a byte when the FIFO is empty.

6.4.1 Receive FIFO Registers

There are five registers directly involved in the operation of the receive FIFOs:

- RXDAT, the receive FIFO data register
- RXCNTL, the receive FIFO byte count register
- RXCON, the receive FIFO control register
- RXFLG, the receive FIFO flag register

These registers are endpoint indexed, i.e., they are used as a set to control the operation of the receive FIFO associated with the current endpoint specified by the EPINDEX register. Figures 6-13 through 6-15 beginning on page 6-26 describe the receive FIFO registers and provide bit definitions.

6.4.1.1 Receive FIFO Data Register (RXDAT)

Received data bytes are written to the receive FIFO via the receive FIFO data register (RXDAT).

RXDAT Address: E3H
(Endpoint-indexed) Reset: xxxx xxxxB

Receive FIFO Data Register. Receive FIFO data specified by EPINDEX is stored and read from this register.

7							0
Receive Data Byte							

Bit Number	Bit Mnemonic	Function
7:0	RXDAT7:0	Receive Data Byte: To write data to the receive FIFO, the FIU writes to this register. To read data from the receive FIFO, the 8*x*931 reads from this register. The write pointer and read pointer are incremented automatically after a write and read, respectively.

Figure 6-13. RXDAT: Receive FIFO Data Register

6.4.1.2 Receive FIFO Byte Count Registers (RXCNTL)

The receive FIFO byte count register (RXCNTL) is used as five-bit ring buffer to accommodate packet sizes of 0 to 16 bytes. This format is shown in Figure 6-14.

RXCNTL† Address: E6H
(Endpoint-indexed) Reset State: xxxx xxxxB

Receive FIFO Byte-count Low Register. Ring buffer used to store the byte count for the data packets received in the receive FIFO specified by EPINDEX.

7							**0**
—	—	—	BC4	BC3	BC2	BC1	BC0

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved. Write zeros to these bits.
4:0	BC4:0	Receive Byte Count. Five-bit, ring buffer. Stores receive byte count.

† Not implemented for hub endpoint 1.

Figure 6-14. RXCNTL: Receive FIFO Byte Count Register

RXCNTL stores the number of bytes in either of the two data sets, data set 0 (ds0) and data set 1 (ds1). The FIFO logic for maintaining the data sets assumes that data is written to the FIFO in the following sequence:

1. The USB interface writes the received data packet into the receive FIFO.
2. The USB interface writes the number of bytes written into the receive FIFO to the byte count register RXCNTL.

The CPU reads the byte count register to determine the number of bytes in the set.

The receive byte count register has a *read/write index* that allows it to access the byte count for either of the two data sets. This is similar to the methodology used for the transmit byte count register. After reset, the read/write index points to data set 0. Thereafter, the following logic determines the position of the read/write index:

- After a read of RXCNTL, the read/write index (RXFIF) is unchanged.
- After a write of RXCNTL, the read/write index (RXFIF) is toggled.

The position of the read/write index can also be determined from the data set index bits, FIF1:0 (see “Receive FIFO Data Set Management” on page 6-27).

CAUTION

Do not read RXCNTL to determine if data is present in the receive FIFO. A read attempt to RXCNTL during the time the receive FIFO is empty causes the RXURF flag in RXFLG to be set. Always read the RXFIF bits in RXFLG to determine if data is present in the receive FIFO. The RXFIF bits are updated after RXCNTL is written (at the end of the receive operation and at the SOF for ISO data).

6.4.2 Receive FIFO Data Set Management

As in the transmit FIFO, the receive FIFO uses a pair of bits (FIF1:0 in the RXFLG register) to indicate which data sets are present in the receive FIFO (see Table 6-6).

Table 6-6. Status of the Receive FIFO Data Sets

FIF1:0		Data Sets Written		
		ds1	ds0	
0	0	No	No	(Empty)
0	1	No	Yes	(1 set)
1	0	Yes	No	(1 set)
1	1	Yes	Yes	(2 sets)

Table 6-7 summarizes how the actions following a reception depend on the RXISO bit, the ARM bit, and the handshake issued by the 8*x*931.

Table 6-7. Truth Table for Receive FIFO Management

RXISO (RXCON.3)	ARM (RXCON.2)	RXERR (RXSTAT.1)	RXACK (RXSTAT.0)	Action at End of Transfer Cycle
X	X	0	0	No operation.
X	0	0	1	Write marker, write pointer, and RXFIF bits remain unchanged. Managed by firmware.
X	0	1	0	Write marker, write pointer, and RXFIF bits remain unchanged. Managed by firmware.
0	1	0	1	Write marker advanced automatically. The RXFIF bit for the corresponding data set is set.
0	1	1	0	Write pointer reversed automatically.The RXFIF bit for the corresponding data set is cleared.
1	1	X	X	Write marker advanced automatically. If data was written to the receive FIFO, the RXFIF bit for the corresponding data set is set.

NOTE

For normal operation, set the ARM bit in RXCON: hardware will automatically control the write pointer and write marker and track the RXFIF bits.

CAUTION

Do not read RXCNTL to determine if data is present in the receive FIFO. Always read the FIF bits in the RXFLG register. RXCNTL contains random data during a receive operation. A read attempt to RXCNTL during the time the receive FIFO is empty causes the RXURF flag in RXFLG to be set. Always read the FIF bits to determine if data is present in the receive FIFO. The RXFLG FIF bits are updated after RXCNTL is written (at the end of the receive operation).

RXCON Address: E4H
(Endpoint-indexed) Reset State: 0xx0 0100B

Receive FIFO Control Register. Controls the receive FIFO specified by EPINDEX.

7							0
RXCLR	—	—	RXFFRC	RXISO	ARM	ADVWM	REVWP

Bit Number	Bit Mnemonic	Function
7	RXCLR	Clear the Receive FIFO: Set this bit to flush the entire receive FIFO. All flags in RXFLG revert to their reset states (RXEMP is set; all other flags clear). The ARM, RXISO, and RXWS bits in this register and the RXSEQ bit in the RXSTAT register are not affected by this operation. Hardware clears this bit when the flush operation is completed.
6:5	—	Reserved: Values read from this bit are indeterminate. Write zero to this bit.
4	RXFFRC	FIFO Read Complete: Set this bit to release the receive FIFO when a data set read is complete. Setting this bit “clears” the RXFIF “bit” (in the RXFLG register) corresponding to the data set that was just read. Hardware clears this bit after the RXFIF bit is cleared. All data from this data set must have been read. **NOTE:** FIFO Read Complete only works if STOVW and EDOVW are cleared.
3	RXISO	Isochronous Data Type: Set this bit to indicate that the receive FIFO is programmed to receive isochronous data and to set up the USB Interface to handle an isochronous data transfer. This bit is not reset when the RXCLR bit is set; it must be cleared by firmware.
2	ARM	Auto Receive Management: When set, the write pointer and write marker are adjusted automatically based on the following conditions: **RXISO / RX Status / Write Pointer / Write Marker** X / ACK / Unchanged / Advanced 0 / NAK / Reversed / Unchanged 1 / NAK / Unchanged / Advanced When this bit is set, setting REVWP or ADVWM has no effect. Hardware neither clears nor sets this bit. This is a sticky bit that is not reset when RXCLR is set. **NOTE:** This bit should always be set, except for testing.

† ARM mode is recommended. ADVWM and REVWP, which control the write marker and write pointer when ARM = 0, are used for test purposes,

Figure 6-15. RXCON: Receive FIFO Control Register

RXCON (Continued) Address: E4H
(Endpoint-indexed) Reset State: 0xx0 0100B

Receive FIFO Control Register. Controls the receive FIFO specified by EPINDEX.

7							0
RXCLR	—	—	RXFFRC	RXISO	ARM	ADVWM	REVWP

Bit Number	Bit Mnemonic	Function
1	ADVWM	Advance Write Marker: † (For non-ARM mode only) Set this bit to advance the write marker to the origin of the next data set. Advancing the write marker is used for back-to-back receptions. Hardware clears this bit after the write marker is advanced. Setting this bit is effective only when the REVWP, ARM, and RXCLR bits are clear.
0	REVWP	Reverse Write Pointer: † (For non-ARM mode only) Set this bit to return the write pointer to the origin of the last data set received, as identified by the write marker. The FIU can then receive the last data packet again and write to the receive FIFO starting from the same origin when the host re-sends the same data packet. Hardware clears this bit after the write pointer is reversed. Setting this bit is effective only when the ADVWM, ARM, and RXCLR bits are all clear. REVWP is used when a data packet is bad. When the function interface receives the data packet again, the write starts at the origin of the previous (bad) data set.

† ARM mode is recommended. ADVWM and REVWP, which control the write marker and write pointer when ARM = 0, are used for test purposes,

Figure 6-15. RXCON: Receive FIFO Control Register (Continued)

RXFLG Address: E5H
(Endpoint-indexed) Reset State: 00xx 1000B

Receive FIFO Flag Register. These flags indicate the status of data packets in the receive FIFO specified by EPINDEX.

7							0
RXFIF1	RXFIF0	—	—	RXEMP	RXFULL	RXURF	RXOVF

Bit Number	Bit Mnemonic	Function
7:6	RXFIF1:0	Receive FIFO Index Flags: (read-only) These read-only flags indicate which data packets are present in the receive FIFO. The RXFIF bits are updated after each write to RXCNTL to reflect the addition of a data packet. Likewise, the RXFIF bits are cleared in sequence after each setting of the RXFFRC bit. The next-state table for RXFIF bits is shown below for operation in dual-packet mode.

RXFIF1:0	Operation	Flag	Next RXFIF1:0	Next Flag
00	Adv WM	X	01	Unchanged
01	Adv WM	X	01	Unchanged
10	Adv WM	X	11	Unchanged
00	Set RXFFRC	X	00	Unchanged
01	Set RXFFRC	X	00	Unchanged
11	Set RXFFRC	X	10/01	Unchanged
10	Set RXFFRC	X	00	Unchanged
XX	Rev WP	X	Unchanged	Unchanged

When the receive FIFO is programmed to operate in single-packet mode (RXSPM set in EPCON), valid RXFIF states are 00 and 01 only.

In ISO mode, RXOVF, RXURF, and RXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. RXFIF is “incremented” by the USB and “decremented” by firmware.Therefore, setting RXFFRC “decrements” RXFIF immediately. However, a successful USB transaction within a frame “increments” RXFIF only at SOF. For traceability, you must check the RXFIF flags before and after reads from the receive FIFO and the setting of RXFFRC in RXCON.

NOTE: To simplify firmware development, it is recommended that you utilize control endpoints in single-packet mode only.

5:4	—	Reserved: Values read from these bits are indeterminate. Write zeros to these bits.

† When set, all transmissions are NAKed.

Figure 6-16. RXFLG: Receive FIFO Flag Register

RXFLG (Continued) Address: E5H
(Endpoint-indexed) Reset State: 00xx 1000B

Receive FIFO Flag Register. These flags indicate the status of data packets in the receive FIFO specified by EPINDEX.

7							0
RXFIF1	RXFIF0	—	—	RXEMP	RXFULL	RXURF	RXOVF

Bit Number	Bit Mnemonic	Function
3	RXEMP	Receive FIFO Empty Flag (read-only): Hardware sets this flag when the write pointer is at the same location as the read pointer ***and*** the write pointer equals the write marker and neither pointer has rolled over. Hardware clears the bit when the empty condition no longer exists. This is not a sticky bit and always tracks the current status of the receive FIFO, regardless of ISO or non-ISO mode.
2	RXFULL	Receive FIFO Full Flag (read-only): Hardware sets this flag when the write pointer has rolled over and equals the read pointer. Hardware clears the bit when the full condition no longer exists. This is not a sticky bit and always tracks the current status of the receive FIFO, regardless of ISO or non-ISO mode.

† When set, all transmissions are NAKed.

Figure 6-16. RXFLG: Receive FIFO Flag Register (Continued)

RXFLG (Continued) Address: E5H
(Endpoint-indexed) Reset State: 00xx 1000B

Receive FIFO Flag Register. These flags indicate the status of data packets in the receive FIFO specified by EPINDEX.

7							0
RXFIF1	RXFIF0	—	—	RXEMP	RXFULL	RXURF	RXOVF

Bit Number	Bit Mnemonic	Function
1	RXURF	Receive FIFO Underrun Flag†: Hardware sets this bit when an additional byte is read from an empty receive FIFO or RXCNTL. Hardware does ***not*** clear this bit, so you must clear it in firmware. When the receive FIFO underruns, the read pointer will not advance — it remains locked in the empty position. In ISO mode, RXOVF, RXURF, and RXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since underrun can only be caused by firmware, RXURF is updated immediately. You must check the RXURF flag after reads from the receive FIFO before setting the RXFFRC bit in RXCON. **NOTE:** When this bit is set, the FIFO is in an unknown state. It is recommended that you reset the FIFO in the error management routine using the RXCLR bit in the RXCON register.
0	RXOVF	Receive FIFO Overrun Flag†: This bit is set when the FIU writes an additional byte to a full receive FIFO or writes a byte count to RXCNTL with FIF1:0 = 11. This is a sticky bit that ***must*** be cleared through firmware, although it can be cleared by hardware if a SETUP packet is received after an RXOVF error had already occurred.† When this bit is set, the FIFO is in an unknown state; thus it is recommended that you reset the FIFO in the error management routine using the RXCLR bit in the RXCON register. When the receive FIFO overruns, the write pointer will not advance — it remains locked in the full position. In ISO mode, RXOVF, RXURF, and RXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since overrun can only be caused by the USB, RXOVF is updated only at the next SOF regardless of where the overrun occurred during the current frame.†

† When set, all transmissions are NAKed.

Figure 6-16. RXFLG: Receive FIFO Flag Register (Continued)

6.5 SIE DETAILS

The USB employs differential data signaling; refer to the signaling levels table in the "Electrical" chapter of *Universal Serial Bus Specification.* The specification defines: differential'1', differential'0', idle ('J' state), non-idle ('K' state), start-of-packet, end-of-packet, disconnect, connect, reset, and resume. The USB employs NRZI data encoding when transmitting packets. Refer to "Data Encoding/Decoding" in the *Universal Serial Bus Specification* for a description of NRZI data encoding and decoding. To ensure adequate signal transitions, bit stuffing is employed by the SIE when transmitting data. The SIE also performs bit unstuffing when receiving data. Consult the "Flow Diagram for Bit Stuffing" figure in the "Bit Stuffing" section of the "Electrical" chapter for more information on bit stuffing.

Bits are sent out onto the bus, least significant bit (LSb) first, followed by the next LSb, and so on. Bytes are sent out onto the bus least significant byte (LSB) first, followed by the next LSB and so on. The serial bus interface engine (SIE) ensures that the LSb is first, but the 8*x*931 programmer must ensure the order of the bytes.

The SIE decodes and takes care of all packet types and packet fields mentioned in "Protocol Layer" chapter of *Universal Serial Bus Specification*. The FIU communicates data information and handshaking instructions to the SIE. Programmers should consult the "Interconnect Description," "USB Devices," and "USB Host" chapters of *Universal Serial Bus Specification* for detailed information on how the host and function communicate.

6.6 SETUP TOKEN RECEIVE FIFO HANDLING

SETUP tokens received by a control endpoint must be ACKed even if the receive FIFO is not empty. When a SETUP token is detected by the FIU, the FIU sets the STOVW bit of RXSTAT and then flushes the receive FIFO by hardware, setting the RXCLR bit of RXCON. The STOVW indicates a SETUP-initiated over-write (flush) is in progress. After the SETUP transaction is completed (i.e., ACK handshake), the FIU clears STOVW and sets EDOVW, indicating the receive FIFO over-write is complete and FIFO contents are stable. Reception of any SETUP packet, regardless of whether the receive FIFO is full or empty always sequences through the STOVW, EDOVW sequence described above.

Note that if the receive FIFO flush occurs in the middle of an 8*x*931 CPU data read cycle (from a previous USB transaction), the receive FIFO could underrun, thus setting the RXURF bit of RXFLG and positioning the read pointer in an unknown state. To prevent this, STOVW resets and locks the read pointer. The read pointer will remain locked until both the STOVW and EDOVW bits are cleared.

CAUTION

For SETUP packets only, firmware must clear EDOVW prior to reading data from the FIFO. If this is not done, data read from the FIFO will be invalid.

After processing a SETUP packet, firmware should always check the STOVW and EDOVW flags before setting the RXFFRC bit. When a SETUP packet either has been or is being received, setting RXFFRC has no effect if either STOVW or EDOVW is set. It is up to the user to clear EDOVW which disables the RXFFRC blocking mechanism. Also note that the RXSETUP = 1 condition causes IN and OUT tokens to be NAKed automatically until RXSETUP is cleared. This

is true even if the transmit and/or receive endpoint is stalled (TXSTL = 1, RXSTL = 1), and is done to allow the clearing of a stall condition on a control endpoint.

NOTE

To simplify firmware development, it is recommended that you utilize control endpoints in single-packet mode only.

6.7 ISO DATA MANAGEMENT

ISO data management must always be performed in dual-packet mode. Interrupts are not generated when an ISO transmit or receive cycle is completed; ISO protocols should always be synchronized to the SOF interrupt.

When transmitting, data written into the transmit FIFO at frame *n* is pre-buffered to be transmitted in frame *n*+1. This guarantees that data is always available to the host when requested anytime in a frame. When receiving, data written into the receive FIFO at frame *n* is pre-buffered to be read-out in frame *n* + 1. This guarantees that data from the host is always available to the function every frame.

Isochronous data transfer is always guaranteed if the OUT or IN tokens from the host are not corrupted. When IN or OUT tokens to a function are corrupted, the host does not re-send the token. Function firmware needs to recognize this error condition and reconfigure the endpoints accordingly.

6.7.1 Transmit FIFO ISO Data Management

When an IN token is corrupted, the data to be transmitted from the transmit FIFO for an isochronous endpoint in the current frame will be flushed. Due to latency concerns, this is handled by hardware. This error condition can be detected by checking TXFIF1:0 = 11 at SOF. When this occurs, the oldest data packet will be flushed and the transmit FIFO read-pointers and read-markers will be advanced to the start “address” of the second data packet. The TXFIF will also be updated. Therefore, the second packet will be ready to be transmitted for the next frame. The first data packet is lost. The transmit flush bit, TXFLUSH in TXSTAT, is also set when this occurs.

For firmware traceability of FIFO status flags, some flags are updated immediately while others are updated only at SOF. TXOVF, TXURF, and TXFIF are handled using the following rule: firmware events cause status change immediately while USB events only cause status change at SOF. For example:

- TXOVF: Since overrun can only be caused by firmware, TXOVF is updated immediately.
- TXURF: Since underrun can only be caused by SIE, TXURF is updated at SOF.
- TXFIF: TXFIF is “incremented” by firmware and “decremented” by hardware. Therefore, writes to TXCNTL will “increment” TXFIF immediately. However, a successful USB transaction anytime in a frame will only “decrement” TXFIF at SOF.

The following bits do not follow the above rule:

- TXEMP/TXFULL: These always reflect the current status of the FIFO.

- TXFLUSH: Firmware can detect a flush by monitoring this bit.

6.7.2 Receive FIFO ISO Data Management

For firmware traceability of FIFO status flags, some flags are updated immediately while others are updated only at SOF. RXOVF, RXURF, and RXFIF are handled using the following rule: firmware events cause status change immediately while USB events only cause status change at SOF. For example:

- RXURF: Since underrun can only be caused by firmware, RXURF is updated immediately.
- RXOVF: Since overrun can only be caused by SIE, RXOVF is updated at SOF.
- RXFIF: RXFIF is "incremented" by hardware and "decremented" by firmware. Therefore, setting RXFFRC will "decrement" RXFIF immediately. However, a successful USB transaction anytime in a frame will only "increment" RXFIF at SOF.
- RXEMP/RXFULL: The rule does not apply to the RXEMP and RXFULL flags, which always reflect the current status of the FIFO.

7

USB Hub

CHAPTER 7
USB HUB

This chapter describes the operation of the Intel Universal Serial Bus (USB) on-chip hub, as implemented in the 8*x*931HA. This chapter introduces on-chip hub operation and includes information on bus enumeration, hub endpoint status and configuration, hub port control, hub suspend and resume, and hub power control.

To see how the hub fits in the 8*x*931HA architecture, see Chapter 2, “Architectural Overview”.

NOTE

The 8*x*931AA microcontroller does not support hub operations. Specific details of the 8*x*931AA are covered in Appendix E, “8x931AA Design Considerations”.

7.1 HUB FUNCTIONAL OVERVIEW

The on-chip hub provides an electrical interface between the USB host and the downstream ports. In many cases, this relationship exists as an interface between a USB host and other discrete USB devices. Besides serving as a control interface between the host and the downstream ports, the hub is also a USB device and must respond to the standard USB requests and hub class-specific requests described in the *Universal Serial Bus Specification*.

The functionality between the PC host and the downstream ports that is handled by the hub includes:

- Connectivity management
- Downstream device connect/disconnect detection
- Power management, including suspend and resume functions
- Bus fault detection and recovery
- Full and low-speed device support

The hub functionality can be divided into two sub-functions: the hub repeater and the hub controller. The hub architecture is described in “Universal Serial Bus Module” on page 2-11.

The hub controller function is split among four modules:

- Hub interface unit (HIU)
- Serial bus interface engine (SIE)
- Transmit and receive FIFOs for hub endpoint 0 and endpoint 1
- 8*x*931HA CPU

A functional diagram of the hub is shown in Figure 7-1 on page 7-2. The diagram shows the root port, which is the upstream port (port 0); the repeater, which is responsible for managing connectivity on a per-packet basis; the hub controller, which provides status and control and permits host access to the hub; four external downstream ports, which provide a means of expanding the USB by permitting the connection of additional PC peripherals; and the internal downstream port, which provides an interface to the embedded function.

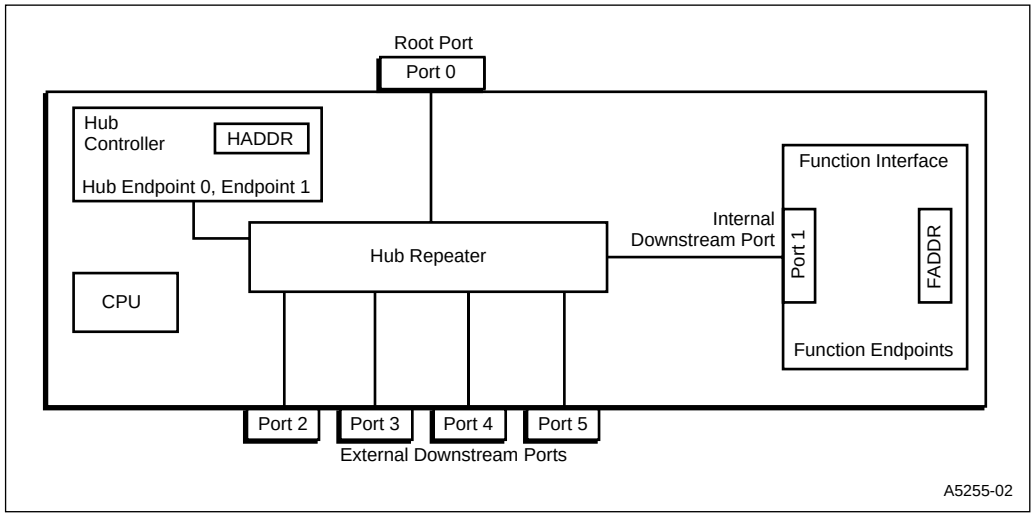


Figure 7-1. 8*x*931HA Hub Functional Diagram

Refer to chapter 11 of *Universal Serial Bus Specification* for a more detailed description of the hub and its functionality. For a description of the transceiver see the "Driver Characteristics" and "Receiver Characteristics" sections of the *Universal Serial Bus Specification*'s "Electrical" chapter. For electrical characteristics and data signal timing, see the "Bus Timing/Electrical Characteristics" and "Timing Diagram" sections of the same chapter.

SFRs used to control and access USB hub functionality are listed in Table 7-1. Figure 7-2 shows the bits contained in the hub SFRs.

Table 7-1. USB Hub SFRs

Mnemonic	Name	Address	Page
HADDR	**Hub Address Register.** Used by the HIU to perform token address decoding.	97H	page 7-8
HIE	**Hub Interrupt Enable Register.** Contains the hub interrupt enable bits.	A1H	page 5-15
HIFLG	**Hub Interrupt Flag Register.** Contains the hub interrupt status flags.	E8H	page 5-16
HPCON	**Hub Port Control.** Enables, disables, resets, suspends, and resumes the hub ports. USB port-indexed using HPINDEX.	CFH	page 7-15
HPINDEX	**Hub Port Index Register.** Provides port indexing into the HPSC, HPSTAT, and HPCON registers.	D4H	page 7-24
HPPWR	**Hub Port Power Control.** Controls power to the downstream ports.	9AH	page 7-28
HPSC	**Hub Port Status Change.** Indicates a change in reset, suspend, enable, disable, or connect status. USB port-indexed using HPINDEX.	D5H	page 7-21
HPSTAT	**Hub Port Status.** Provides D_P, D_M, low-speed device, power, reset, suspend, enable, and disable status for the hub ports**.** USB port-indexed using HPINDEX.	D7H	page 7-18
HSTAT	**Hub Status and Configuration.** Used to examine or enable remote wake-up, stall feature, endpoint 1, over-current status, and local power status	AEH	page 7-9

	7							0
HADDR	—	Hub Address						
HIE	—	—	—	—	—	—	HRXE0	HTXE0
HIFLG	—	—	—	—	—	—	HRXD0	HTXD0
HPCON	—	—	—	—	—	HPCON2	HPCON1	HPCON0
HPINDEX	—	—	—	—	—	HPIDX2	HPIDX1	HPIDX0
HPPWR	—	—	HPPWR5	HPPWR4	HPPWR3	HPPWR2	HPPWR1	—
HPSC	—	—	—	RSTSC	—	PSSC	PESC	PCSC
HPSTAT	DPSTAT	DMSTAT	PPSTAT	LSSTAT	PRSTAT	PSSTAT	PESTAT	PCSTAT
HSTAT	OVRIEN	HRWUPE	EP1STL	EP1EN	OVISC	—	OVI	—

Figure 7-2. Bits of the USB Hub SFRs

7.1.1 Port Connectivity States

In addition to the root port (port 0), the hub contains four external downstream ports (ports 2, 3, 4, and 5) and one internal downstream port, port 1.

Hub downstream ports may be in one of five possible states:

- powered-off — Power-switched ports are a USB option supported by the 8*x*931. A powered off port supplies no power downstream, ignores all upstream-directed bus activity on the port, and its signal output buffers are placed in the high impedance state.
- disconnected — Initial state upon power-up or reset, the port cannot propagate any upstream or downstream signaling. The port can detect a connect event, which causes it to transition to the disabled state.
- disabled — Port can only propagate downstream-directed signaling arising from a reset request. A disabled port does not propagate upstream signaling if the hub is awake, but will detect disconnects and initiate resume signaling to the root port if the hub is suspended.
- enabled — Port propagates all downstream and upstream signaling.
- suspended — When suspended, the port will not stop propagating in the middle of a transaction. If hub is awake, no upstream or downstream connectivity can propagate through the port, except for downstream-directed reset signaling. If hub is suspended, idle-to-resume is propagated.

The transitions between these states are shown in Figure 7-3.

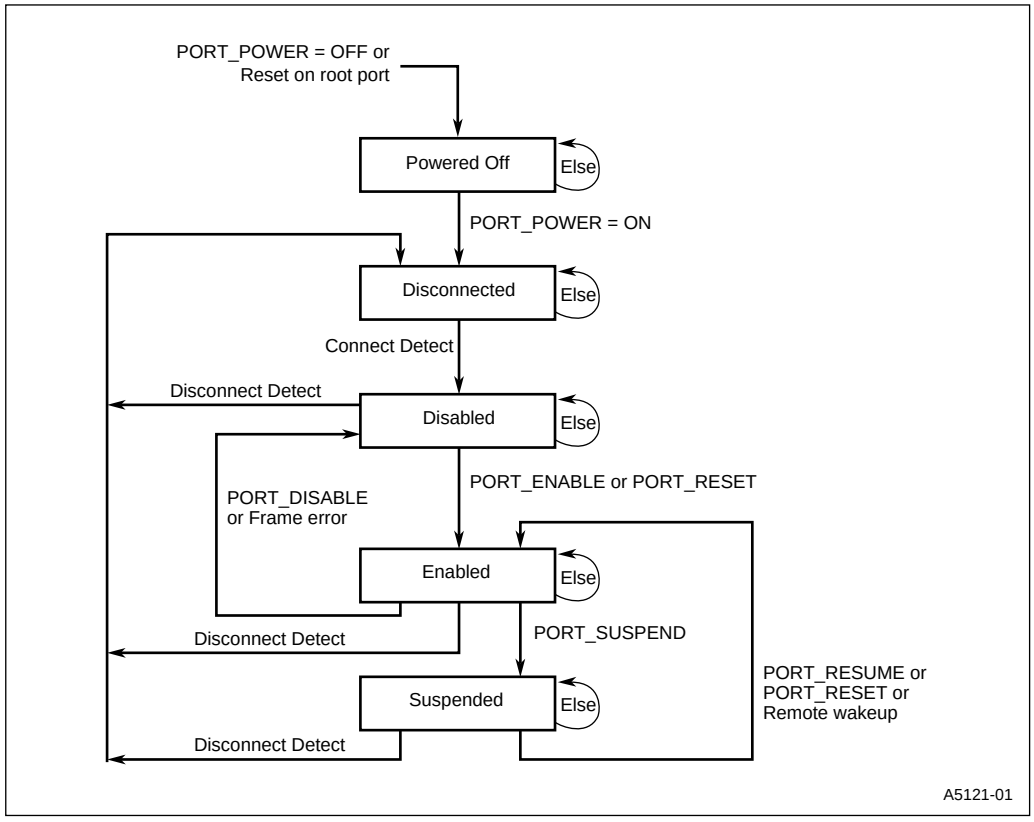


Figure 7-3. Hub State Flow

These port states are tracked and managed in the hub repeater based on hardware events (e.g., physical connection/disconnection of a device on a port) and firmware execution of host commands. Normal packet traffic is allowed to propagate through ports that are in the "enabled" state only, as described in "Per-packet Signaling Connectivity" on page 7-6.

The root port is the only upstream port; it is permanently powered on and enabled. Hub ports 2, 3, 4, and 5 are external downstream ports. They are power-switched ports that must be powered-on by host command, detect a device connection and then become enabled via host command prior to propagating USB packet traffic. Hub port 1 is an internal downstream port that is always powered on and always physically connected. It functionally supports port enabling. That is, the downstream port connectivity will not be enabled unless a port enable has been received from the host.

7.1.2 Per-packet Signaling Connectivity

The hub repeater establishes connectivity between ports for upstream and downstream traffic on a per-packet basis. Packet signaling connectivity for downstream, upstream, and idle traffic is illustrated in Figure 7-4. While the host can communicate with all the downstream ports simultaneously, as shown in the "downstream connectivity" illustration in Figure 7-4, only one port can communicate with the host at one time, as shown in the "Upstream Connectivity" illustration of the same figure. The host selects one of the downstream ports for upstream communication.

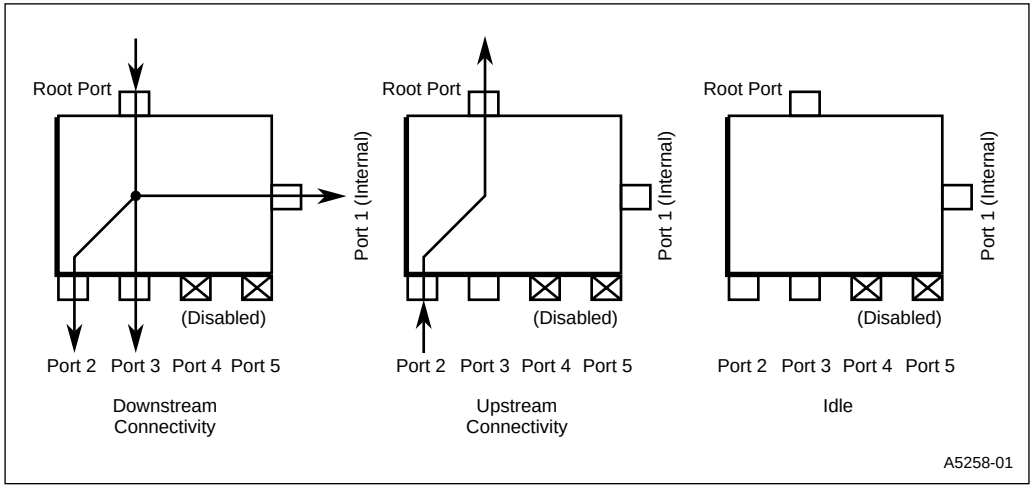


Figure 7-4. Packet Signaling Connectivity

Connections made by the repeater also depend on whether the port is attached to a full-speed or low-speed device and whether the USB packet is a full-speed or low-speed packet. Low-speed packets are identified by a PREamble token. Connections are made by the repeater using asynchronous control logic in order to meet the USB signal propagation requirements.

7.1.2.1 Connectivity to Downstream Ports Attached With Full-speed Devices

Downstream connectivity is established upon detection of a start of packet (SOP) transmitted on the root or upstream port by the USB host. As shown in Figure 7-4, the connection is made from the root port (port 0) to all enabled downstream ports attached with full-speed devices (ports 2 and 3 in this case). Connectivity is *not* established to any enabled ports attached with low-speed devices. Upon detection of the end-of-packet (EOP), the repeater terminates the connectivity, reverting to the idle state, as shown in Figure 7-4.

Upstream connectivity is established upon detection of a SOP transmitted on any enabled downstream port. The connection is only made between a single downstream port and the root port by the repeater, as shown in "upstream connectivity" in Figure 7-4. The USB protocol does not allow packets to be transmitted by more than one downstream port simultaneously, but in an error scenario where this happens, the repeater would choose only one downstream port to connect upstream. Once again, upon detection of an EOP, the connectivity is terminated.

7.1.2.2 Connectivity to Downstream Ports attached with Low-speed Devices

Downstream connectivity is established in the same fashion for low-speed packets as for full-speed packets, with the following exceptions:

1. Downstream low-speed packets are routed to all enabled ports, including ports attached with low-speed and full-speed devices.
2. Downstream low-speed packets contain a low-speed PREamble field which is recognized by the SIE. Upon detection of the PREamble, the repeater establishes the connection to all enabled low-speed downstream ports.
3. Packet data is inverted at the ports attached to low-speed devices for both upstream and downstream traffic.

Upstream connectivity is established in the same fashion for low-speed packets as for full-speed packets, with the exception that no PREamble is propagated prior to low-speed packets. The root port propagates low-speed packets upstream using full-speed signaling (edge rates).

7.2 BUS ENUMERATION

The USB host manages bus enumeration at system start-up or whenever a new USB device is attached to the host or to a hub's downstream port. Initially, the USB hub is in the unenumerated state and the hub address register (HADDR) contains the default value 00H. The host PC performs bus enumeration in which it identifies and addresses devices attached to the bus. During enumeration, a unique address assigned by the host is written to the HADDR of every hub device.

Information on descriptors and the HADDR register, required for bus enumeration, is provided in the following subsections.

7.2.1 Hub Descriptors

The 8*x*931 has five descriptors, as shown in Table 7-2. All are standard USB descriptors except the hub descriptor, which is class-specific. There is no descriptor for endpoint 0. A hub has only one valid configuration and interface. The actual descriptor field values are given in the section of *Universal Serial Bus Specification* referenced in the table.

The host reads the hub descriptors during bus enumeration. The host uses the values within the descriptors to determine device configuration. The hub descriptor is divided into several parts, which are shown in Table 7-3.

Table 7-2. 8*x*931 Descriptors

Descriptor	Size	*Universal Serial Bus Specification* Reference
Device	18 bytes	Section 9.7.1
Configuration	9 bytes	Section 9.7.2
Interface	9 bytes	Section 9.7.3
Endpoint	7 bytes	Section 9.7.4
Hub	9 bytes	Section 11.11.2

Table 7-3. Hub Descriptors

Field	Size	Offset	Description
bDescLength	1 byte	0	Number of bytes in this descriptor, including this byte.
bDescriptorType	1 byte	1	Descriptor Type
bNbrPorts	1 byte	2	Number of downstream ports this hub supports.
wHubCharacteristics	2 bytes	3	Determines power switching mode, identifies device as a compound device, and describes the over-current protection mode used by the device.
bPwrOn2PwrGood	1 byte	5	Time elapsed from when the power on sequence begins on a port until power is good on that port.
bHubContrCurrent	1 byte	6	Maximum current requirements of the hub controller.
DeviceRemovable	1 byte	7	Indicates if a port has a removable device attached.
PortPwrCtrlMask	1 byte	Variable	Indicates if a port is affected by a gang-mode power control request.

7.2.2 The Hub Address Register (HADDR)

During bus enumeration, the host PC communicates a unique address for the hub through hub endpoint 0 using the set address command. Device firmware must interpret and write this hub address to the Hub Address register (HADDR, as shown in Figure 7-5). This procedure is outlined in "Enumeration" on page 8-2.

HADDR

Address: 97H
Reset State: 0000 0000B

Hub Address Register. This SFR holds the address for the hub device. During bus enumeration it is written with a unique value assigned by the host.

7	6	5	4	3	2	1	0
—	Hub Address						

Bit Number	Function
7	Reserved. Write a zero to this bit.
6:0	Hub address register: Updated using a SET_ADDRESS USB host request. This address is used by the HIU to perform token decoding.

Figure 7-5. HADDR: Hub Address Register

7.3 HUB STATUS

Status and configuration of the USB hub function is performed by both standard and hub class-specific USB requests. These requests, generated by the host PC, manage and configure the status of the hub and its downstream ports. These USB requests are listed and explained in “Hub Status and Configuration” on page 8-17.

The hub has an internal downstream port (port 1) which operates differently than the external downstream ports. Because this port is physically connected to the embedded function and is powered-on at all times, USB requests intended for the internal downstream port are handled differently than similar requests to the other downstream ports. The management of the individual hub ports is discussed in “USB Hub Ports” on page 7-14.

The host PC may request that firmware check and change bits of the HSTAT SFR (Figure 7-6). See Table 8-1 on page 8-17 for a list of USB requests and their associated firmware actions.

HSTAT

Address: AEH
Reset State: 0000 0000B

Hub Status and Configuration Register. This SFR contains bits for remote wake-up request, status and status change indicators for over-current and hub endpoint 1 stall and enable.

7							0
OVRIEN	HRWUPE	EP1STL	EP1EN	OVISC	—	OVI	—

Bit Number	Bit Mnemonic	Function
7	OVRIEN	Overcurrent Detect Enable Bit: This bit is used to gate off the overcurrent input detect which is multiplexed with P3.0. When set, a low on P3.0/OVRI# pin will trigger over current detection logic. When this bit is ’0’ the over current detection logic is disabled.
6	HRWUPE	Hub Remote Wake-up Enable Bit: Set if the device is currently enabled to request remote wake-up. This bit is modified through the SetFeature and ClearFeature requests using the DEVICE_REMOTE_WAKEUP feature selector. When ‘0,’ the hub blocks resume signaling for connect/disconnect and resume events detected on downstream ports. **NOTE:** Do not set this bit until after the hub is enumerated and the host issues a SET_FEATURE command with a DEVICE_ REMOTE_WAKEUP feature selector.

† Bits 1 and 3 are returned in response to a Get Hub Status request from the USB host. This response is a four-byte field with zero padding (MSB at left): 0000 0000 0000 00[3] 0 0000 0000 0000 00[1] 0.

Figure 7-6. HSTAT: Hub Status and Configuration Register

HSTAT (Continued) Address: AEH
Reset State: 0000 0000B

Hub Status and Configuration Register. This SFR contains bits for remote wake-up request, status and status change indicators for over-current and hub endpoint 1 stall and enable.

7							0
OVRIEN	HRWUPE	EP1STL	EP1EN	OVISC	—	OVI	—

Bit Number	Bit Mnemonic	Function
5	EP1STL	Hub Endpoint 1 Stall Field: Set to '1' via the USB SetFeature request with endpoint stall feature selector. When '1,' will force a stall response when endpoint 1 is addressed. Reset with USB ClearFeature request with endpoint stall feature selector.
4	EP1EN	Hub Endpoint 1 Enable: Set to '1' upon receipt of a USB SetConfiguration request value of 0001H. Endpoint 1 cannot respond unless this bit is set. Bit is reset upon receipt of configuration value other than 0001H or a system or USB reset. **NOTE:** This bit must be set in order for the UPWEN# pin to enable power to the downstream ports. Downstream power cannot be applied until this is done.
3	OVISC	Hub Over-current Indicator Status Change (read/clear-only): † Set to '1' if change is detected in the over-current status, even if the condition goes away before it is detected by firmware. Cleared via a USB ClearFeature request with C_HUB_OVER_CURRENT feature selector. Cleared to '0' if no change.
2	—	Reserved: The value read from this bit is indeterminate. Write a zero to this bit.
1	OVI	Latest Over-current Indicator (read-only): † Hardware sets and clears this bit via the OVRI# input pin.'1' indicates an over-current condition. '0' indicates normal power operation.
0	—	Reserved: The value read from this bit is indeterminate. Write a zero to this bit.

† Bits 1 and 3 are returned in response to a Get Hub Status request from the USB host. This response is a four-byte field with zero padding (MSB at left): 0000 0000 0000 00[3] 0 0000 0000 0000 00[1] 0.

Figure 7-6. HSTAT: Hub Status and Configuration Register (Continued)

7.4 USB HUB ENDPOINTS

Table 7-4 gives the packet size, transfer type and implementation of the 8*x*931 hub endpoints. Bulk and isochronous transfers are not supported by the hub endpoints. The hub handles control transfers using endpoint 0 with a maximum packet size of eight bytes.

Hub endpoint 1 supports interrupt transfers only and has no endpoint receive buffer. Endpoint 1 is used to inform the host of a hub or port status change. Figure 7-8 on page 7-13 illustrates the format used to transmit status change information to the host. Since endpoint 1 transmits a single byte of information, TXDAT (Figure 7-7 on page 7-12) serves as the data buffer. Endpoint 1 operations are primarily controlled by hardware and do not involve firmware, except for the EP1STL and EP1EN bits in HSTAT (Figure 7-6).

Table 7-4. Hub Endpoint Configuration

Hub Endpoint	Max Packet Size	Transfer Type	Implementation
0	8 bytes	Control	Firmware-controlled
1	1 byte	Status Change Interrupt	Hardware-controlled

7.4.1 Hub Endpoint Indexing Using EPINDEX

The 8*x*931 hub endpoint 0 uses the same communication registers (TXCNTL, RXCNTL, TXDAT, RXDAT, TXFLG, RXFLG, TXSTAT, RXSTAT, TXCON, and RXCON) as the embedded USB function endpoints. The EPINDEX register ( ), used to access the registers of the USB function endpoints, is also used to access the registers for hub endpoints.

To access the communication SFRs for the hub endpoints, first write a ‘1’ to bit 7 of EPINDEX. To access the internal USB function’s registers, write ‘0’ to EPINDEX’ bit 7. Regardless of whether you are accessing the hub or function endpoints, the LSbs of EPINDEX are used to control which endpoint’s registers are accessed.

For additional information on how to use EPINDEX, see “Endpoint Selection” on page 5-4.

7.4.2 Hub Endpoint Control

Hub endpoint 1 of the 8*x*931 is controlled primarily by hardware, with these exceptions:

- Firmware can read endpoint 1’s TXDAT SFR
- Firmware can stall hub endpoint 1 in response to a Set_Feature (ENDPOINT_STALL) request from the host by setting the EP1STL bit in HSTAT (Figure 7-6). Firmware can also clear this bit in response to a Clear_Feature request.
- Firmware can enable hub endpoint 1 in response to a Set_Configuration request from the host by setting the EP1EN bit in HSTAT (Figure 7-6 on page 7-9)

Firmware can control hub endpoint 0 through its EPCON register ( ) when EPINDEX has previously been set to 80H. Hub endpoint control for endpoint 0 behaves identically to function endpoint control, except that hub endpoint 0 is always a single-packet, control endpoint. Therefore, the corresponding bits (CTLEP and RXSPM) of its EPCON SFR are hardwired to ‘1’.

7.4.3 Hub Endpoint Transmit and Receive Operations

The 8*x*931 hardware uses hub endpoint 1's TXDAT register (Figure 7-7) to transmit a port status change interrupt to the host. Figure 7-8 shows how a hub or port status change is reflected in TXDAT.

TXDAT is cleared by firmware upon a ClearPortFeature request from the host. See "Monitoring Port Status Change Using HPSC" on page 7-20 for a description of how firmware interacts with the host to communicate a change in port status.

NOTE

Although the bits of hub endpoint 1's TXDAT SFR are firmware read-only, bits 5:1 of TXDAT can be cleared indirectly by writing to a port's HPSC SFR. Clearing all bits in a port's HPSC causes hardware to clear the bit associated with that port in hub endpoint 1's TXDAT. Hub endpoint 1's TXDAT.0 can be cleared indirectly by clearing HSTAT's OVISC bit.

TXDAT (For hub endpoint 1 only) EPINDEX=81H† Address: F3H
Reset State: xxxx xxxxB

7							0
—	—	TXDAT5	TXDAT4	TXDAT3	TXDAT2	TXDAT1	TXDAT0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Values read from this bit(s) are indeterminate.
5:0	TXDAT5:0	Hub Endpoint 1 Status Change (read-only††): Hardware communicates status changes to the host by setting the appropriate bit: TXDAT0 hub status change TXDAT1 port 1 status change TXDAT2 port 2 status change TXDAT3 port 3 status change TXDAT4 port 4 status change TXDAT5 port 5 status change A '1' indicates a status change and '0' indicates no status change. When endpoint 1 is addressed via an IN token, the entire byte is sent if at least one bit is a '1'. If all bits are zero, a NAK handshake is returned.

† TXDAT SFRs are also used for function (and hub endpoint 0) data transmission (EPINDEX=0xH or 80H). In that case, the bits are defined differently as shown in Figure 6-8 on page 6-16.

†† Bits 5:1 can be set indirectly by firmware by writing to a port's HPSC SFR. Setting any bit in port *x*'s HPSC results in the hardware setting bit *x* in TXDAT. TXDAT bits can be cleared indirectly in firmware by clearing all bits in that port's HPSC.

Figure 7-7. TXDAT: Hub Transmit Data Buffer (Endpoint 1)

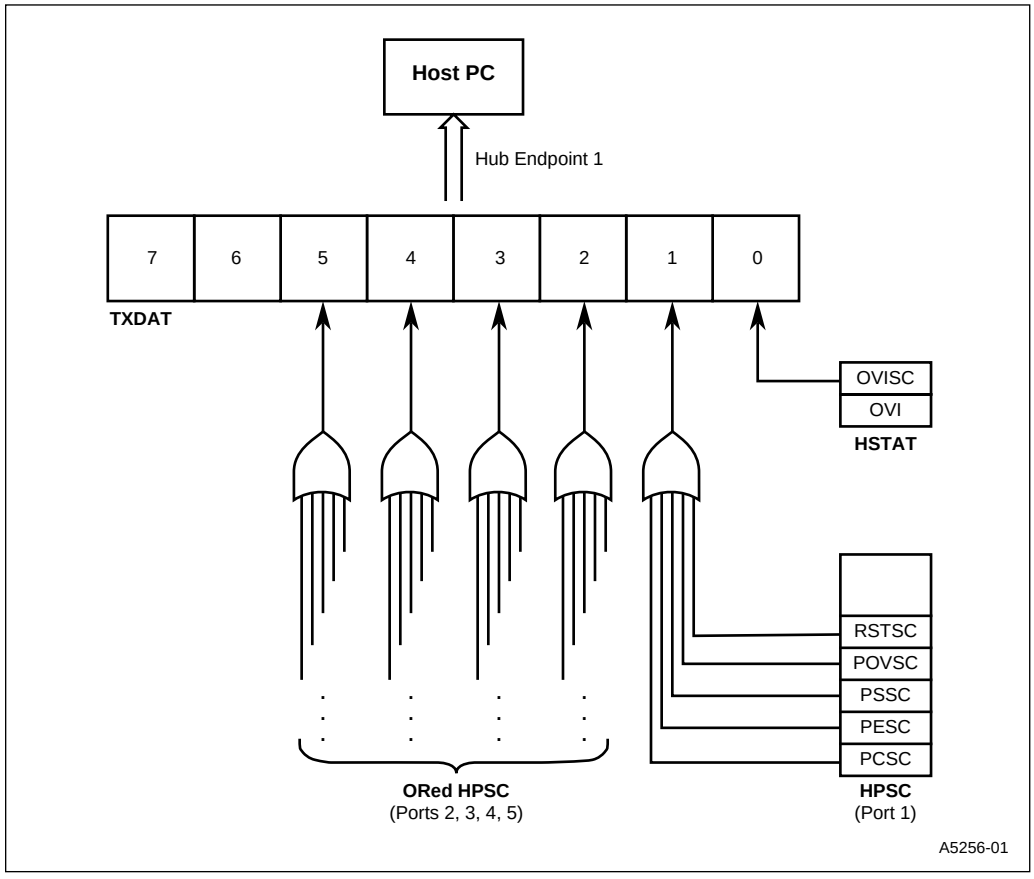


Figure 7-8. Status Change Communication To Host

The remaining hub transmit and receive registers communicate control information between the host and either the internal function or the downstream ports. The 8*x*931 communicates this control information through endpoint 0 using procedures identical to those outlined for the function control endpoint (function endpoint 0) in "Transmit FIFOs" on page 6-14 and "Receive FIFOs" on page 6-24.

NOTE

Hub endpoint 0's transmit SFRs (e.g., TXDAT, TXCNTL, TXFLG, and TXSTAT) behave identically to their function counterparts. For example, when firmware writes to endpoint 0's TXDAT, hardware automatically transfers the byte into the transmit FIFO before the next write to TXDAT. Placing the byte count into hub endpoint 0's TXCNTL prepares the bytes to be transmitted from the FIFO through hub endpoint 0 at the next IN token.

7.5 USB HUB PORTS

In addition to the root port (port 0) and the embedded function addressed by port 1, the hub contains four external downstream ports, ports 2, 3, 4, and 5.

7.5.1 Controlling a Port Using HPCON

You can change a port's status by writing an encoded hub port control command to the hub port control register (HPCON, as shown in Figure 7-9). All ports can be controlled by HPCON using the HPINDEX SFR for indexing. See "Hub Port Indexing Using HPINDEX" on page 7-23 for a description of how port indexing works. Table 7-6 on page 7-16 gives a complete description of the encoded hub port control commands. The 8*x*931 hardware can also change the status of a port, and some port features (i.e., low-speed/full-speed and connect/disconnect) can only be changed by hardware.

8*x*931 hardware ignores certain USB port requests if the request has no meaning within the context of the current port state. For example, there is no need to activate power to a port that is enabled, disabled, or suspended, because a port in one of these states already has power applied. An activate power request (SetPortFeature with a port power feature selector) is supported for a given port only when that port is in the powered-off state. For all other states, the request is ignored by hardware. Table 7-5 depicts the state-related USB requests and the port states for which they are ignored. Upon receipt of a state-related USB request, firmware must examine the HPSTAT SFR to determine the current port state. If the port is in a state where the request will be ignored by hardware, firmware must respond to the host by sending a STALL during the transaction status stage to indicate the command was not completed. Port states are discussed in "Port Connectivity States" on page 7-4 and shown in Figure 7-3 on page 7-5.

Table 7-5. USB Requests Ignored by Hardware (by Port State)

USB Request	Response by Port State [as indicated by the given bit in HPSTAT]				
	Powered-Off [PPSTAT = 0]	Disconnected [PCSTAT = 0]	Disabled [PESTAT = 0]	Enabled [PESTAT = 1]	Suspended [PSSTAT = 1]
SetPortFeature (Port Power)		Ignored	Ignored	Ignored	Ignored
ClearPortFeature (Port Power)	Ignored				
SetPortFeature (Port Enable)	Ignored	Ignored		Ignored	Ignored
ClearPortFeature (Port Enable)	Ignored	Ignored	Ignored		Ignored
SetPortFeature (Port Reset)	Ignored	Ignored			
SetPortFeature (Port Suspend)	Ignored	Ignored	Ignored		Ignored
ClearPortFeature (Port Suspend)	Ignored	Ignored	Ignored	Ignored	

After you request a port status change through HPCON, it may take the 8*x*931 hardware a period of time to affect the change, depending on the current state of the hub port and its current operation. You can check the HPSC SFR to see that your latest change has taken effect, as described in “Monitoring Port Status Change Using HPSC” on page 7-20.

NOTE

Port connect status cannot be changed through HPCON. This port feature is controlled by physically connecting or disconnecting a device from the port.

HPCON
(Indexed by HPINDEX)

Address: CFH
Reset State: xxxx x000B

Hub Port Control Register. Firmware writes to this register to disable, enable, reset, suspend, and resume a port.

7							0
—	—	—	—	—	HPCON2	HPCON1	HPCON0

Bit Number	Bit Mnemonic	Function
7:3	—	Reserved: Write zeros to these bits.
2:0	HPCON2:0	Encoded Hub Port Control Commands: All bits should be set and cleared by firmware after receiving the USB requests ClearPortFeature and SetPortFeature from the host. The bits are encoded as follows (all other bit combinations are ignored by the hardware): 000 — Disable port 001 — Enable port 010 — Reset and enable port 011 — Suspend port 100 — Resume port See Table 7-6 on page 7-16 for a complete description of the encoded hub port control commands.

Figure 7-9. HPCON: Hub Port Control Register

Port 1 represents the internal downstream port and differs from the external downstream ports. The internal downstream port is always connected (and cannot be disconnected). Hub port control commands have a different effect on port 1 than they do on the external downstream ports, as shown in Table 7-6 below.

Table 7-6. Encoded Hub Port Control Commands

Code	Command	Condition	Results [Port x (x=2,3,4,5)]	Results [Port 1]
000	Disable port	Firmware should write '000' to HPCON upon receipt of a ClearPortFeature with a PORT_ENABLE feature selector.	Places port in the disabled state the next time the bus is idle.	Same
001	Enable port	Firmware should write '001' to HPCON upon receipt of a SetPortFeature with a PORT_ENABLE feature selector.	Places port in the enabled state the next time the bus is idle.	Same
010	Reset and enable port	Firmware should write '010' to HPCON upon receipt of SetPortFeature with PORT_RESET feature selector.	Causes port ***x*** to immediately drive an SE0 downstream for at least 15 msec and then places the port in the enabled state.	Causes an internal hardware reset of the FIU and FIFO circuitry relating to the embedded function. Certain embedded function SFRs are reset to their default values (as listed in "Embedded Function Reset" on page 7-24). After at least 15 ms, hardware automatically places the port in the enabled state. Firmware should handle reset of any other firmware and hardware features relating to the embedded function immediately after initiating the reset and enable through this SFR (must be complete by 15 ms from start of reset).
011	Suspend port	Firmware should write '011' to HPCON upon receipt of SetPortFeature with PORT_SUSPEND feature selector.	Places the port in an idle "J" state the next time the bus is idle and prevents the port from propagating USB traffic.	Suspends the embedded function's port the next time the bus is idle, preventing port 1 from generating any USB traffic. Firmware should suspend port 1 only after doing any necessary processing (i.e., putting any external components in a low-power state) to place the embedded function into a suspended state.
100	Resume port	Firmware should write '100' to HPCON upon receipt of ClearPortFeature with PORT_SUSPEND feature selector.	Causes port ***x*** to immediately drive a "K" state downstream for at least 20 msec followed by a low-speed EOP, and then places the port back in the enabled state.	Places port 1 into the enabled state after 20 ms. Firmware should resume port 1 only after doing any necessary processing to take the embedded function out of the suspended (low-power) state.

7.5.2 Examining a Port's Status Using HPSTAT

You can examine a port's status using the hub port status register (HPSTAT, as shown in Figure 7-10 on page 7-18). The HPSTAT SFR can show the status for any of the ports by using the HPINDEX SFR for indexing. See "Hub Port Indexing Using HPINDEX" on page 7-23 for a description of how this indexing works.

HPSTAT gives the current D_P and D_M values for the selected port; these implement the Get Bus State diagnostic aid to facilitate system debug (See the *Universal Serial Bus Specification*). HPSTAT contains a bit that indicates when a low-speed device is attached to a port. HPSTAT also shows a given port's reset status, and whether the port is powered on or off, connected or disconnected, enabled or disabled, or suspended.

NOTE

Firmware-initiated port status changes are not reflected in HPSTAT until the next end-of-frame.

The HPSTAT SFR is read-only. To change the status of a port feature, you must do so indirectly using the HPCON SFR. The 8*x*931HA hardware can also change the status of a port, and some features can only be changed by hardware. See "Controlling a Port Using HPCON" on page 7-14.

HPSTAT Address: D7H
(Indexed by HPINDEX) Reset State: 100d 0000B

Hub Port Status Register. This register indicates the current status for a port, including power, reset, suspend, low-speed device, enable, connect, D_P, and D_M status.

7							0
DPSTAT	DMSTAT	LSSTAT	PPSTAT	PRSTAT	PSSTAT	PESTAT	PCSTAT

Bit Number	Bit Mnemonic	Function
7	DPSTAT	D_P Status (read-only): Value of D_P for port ***x*** at end of last frame. Firmware must return this bit in response to a GetBusState request from the host. Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware at the EOF2 point near the end of a frame (used for diagnostics). Port 1: Hard-wired to '1', since there is no D_P signal for the embedded port
6	DMSTAT	D_M Status (read-only): Value of D_M for port ***x*** at end of last frame. Firmware must return this bit in response to a GetBusState request from the host. Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware at the EOF2 point near the end of a frame (used for diagnostics). Port 1: Hard-wired to '0', since there is no D_M signal for the embedded port.
5	LSSTAT	Low-speed Device Attach Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware upon detection of the presence or absence of a low-speed device at the EOF2 point near end-of-frame. '1' = low-speed device is attached to port ***x***. '0' = full-speed device is attached to port ***x***. Port 1: Hard-wired to '0' (full-speed), since port 1 is permanently attached to the embedded USB function.
4	PPSTAT	Port Power Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware based on the present power status of the port, as controlled either by firmware using the HPPWR register, or by an overcurrent condition in hardware. '1' = port ***x*** is powered on. '0' = port ***x*** is powered off. The port ***x*** power status is only sampled at the EOF2 point near end-of-frame. Port 1: Hard-wired to '1', since the internal function is always powered-on.

NOTES:

Firmware returns the bits of this register in the first word of the 8***x***931' response to the host's GetPortStatus request. See "GetPortStatus Request Firmware" on page 8-25.

Overcurrent indication is not represented on a per-port basis because the 8***x***931 supports ganged power control and overcurrent indication.

Figure 7-10. HPSTAT: Hub Port Status Register

HPSTAT (Continued) Address: D7H
(Indexed by HPINDEX) Reset State: 100d 0000B

Hub Port Status Register. This register indicates the current status for a port, including power, reset, suspend, low-speed device, enable, connect, D_P, and D_M status.

7 0

DPSTAT	DMSTAT	LSSTAT	PPSTAT	PRSTAT	PSSTAT	PESTAT	PCSTAT

Bit Number	Bit Mnemonic	Function
3	PRSTAT	Port Reset Status (read-only): Port ***x*** ((***x***=2,3,4,5): Set and cleared by hardware as a result of initiating a port ***x*** reset by writing to HPCON. '1' = reset signaling is currently asserted for port ***x***. '0' = reset signaling is not asserted. Sampled only at the EOF2 point near end of frame. Port 1: Same as port ***x***.
2	PSSTAT	Port Suspend Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware as controlled by firmware via HPCON. '1' = port ***x*** is currently suspended. '0' = not suspended. Sampled only at the EOF2 point near end of frame. Port 1: Same as port ***x***.
1	PESTAT	Port Enable/Disable Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware as controlled by firmware via HPCON. '1' = port ***x*** is currently enabled. '0' = port is disabled. Sampled only at the EOF2 point near end of frame. Port 1: Same as port ***x***.
0	PCSTAT	Port Connect Status (read-only): Port ***x*** connect status from previous frame time. Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware after sampling the connect state at EOF2 near the end of the present frame. '1' = device is present on port ***x***. '0' = device is not present. This bit will be set if either a physical connection is detected, or during a hub reset when a downstream device is already connected. This bit will be cleared if a disconnect is detected. Port 1: Hard-wired to '1', since the internal function is permanently connected.

NOTES:

Firmware returns the bits of this register in the first word of the 8***x***931' response to the host's GetPortStatus request. See "GetPortStatus Request Firmware" on page 8-25.

Overcurrent indication is not represented on a per-port basis because the 8***x***931 supports ganged power control and overcurrent indication.

Figure 7-10. HPSTAT: Hub Port Status Register (Continued)

7.5.3 Monitoring Port Status Change Using HPSC

When firmware changes the status of a port, there may be a delay between the time firmware requests the status change (using the HPCON register, as described in "Controlling a Port Using HPCON" on page 7-14) and the time hardware actually changes the state. This occurs because some port changes require hardware to perform auxiliary functions (such as driving a state downstream for up to 20ms). Additionally, some status changes are initiated by hardware. Firmware can determine when a port status change has occurred by monitoring the HPSC register (Figure 7-11).

NOTE

Firmware-initiated port status changes are not reflected in HPSC until the next end-of-frame.

The 8*x*931HA uses the 1-byte TXDAT register associated with endpoint 1 to communicate a port status change to the host (Figure 7-7 on page 7-12). Bits in this register are set by the 8*x*931HA hardware to indicate which ports (or the hub itself) have changed status.

After receiving notification of a port status change through endpoint 1, the host may request additional information regarding the status change using a GetPortStatus request. 8*x*931HA firmware must respond to the GetPortStatus request by transmitting the contents of the HPSTAT and HPSC registers to the host in a two-word format. This process is described in "GetPortStatus Request Firmware" on page 8-25.

The HPSC register (Figure 7-11) indicates which port feature has changed status. Port features whose status changes are reflected by HPSC include reset, suspend, enable, and connect.

HPSC
(Indexed by HPINDEX)

Address: D5H
Reset State: xxx0 0000B

Hub Port Status Change Register. This register indicates a change in status for a port, including over-current, reset, suspend, low-speed device, enable and connect status.

7							0
—	—	—	RSTSC	—	PSSC	PESC	PCSC

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved: Write zeros to these bits.
4	RSTSC	Reset Status Change (read-, clear-only): This bit is cleared in firmware via the USB host request ClearPortFeature with a C_PORT_RESET feature selector. ‘1’ indicates reset of port complete; ‘0’ indicates no change. Port *x* (*x*=2,3,4,5): This bit is set by hardware approximately 10 msec after receipt of a port reset and enable command (SetPortFeature with PORT_RESET feature selector). Port 1: This bit is set by hardware at the EOF2 point near the end of a frame after completion of the hardware-timed reset due to firmware execution of a port reset and enable command (SetPortFeature with PORT_RESET feature selector).
3	—	Reserved: Write a zero to this bit.
2	PSSC	Port Suspend Status Change (read-, clear-only): This bit is cleared by firmware upon a USB host request ClearPortFeature with C_PORT_SUSPEND feature selector. ‘1’ = resume process complete; ‘0’ = no change. The resume process is initiated by firmware upon reception of a SetPortFeature request with a PORT_SUSPEND feature selector. Port *x* (*x*=2,3,4,5): This bit is set by hardware upon completion of the firmware-initiated resume process. Port 1: This bit is set by hardware 20 msec after the next EOF2 point after completion of the resume process.

NOTE: Bits are returned as part of the second word (2 bytes) in response to a Get Port Status request from the USB host. The upper 11 MSbs are reserved and always ‘0’ per USB 1.0: 0000 0000 000.4.3.2.1.0 (MSB at left).

Figure 7-11. HPSC: Hub Port Status Change Register

HPSC (Continued) Address: D5H
(Indexed by HPINDEX) Reset State: xxx0 0000B

Hub Port Status Change Register. This register indicates a change in status for a port, including over-current, reset, suspend, low-speed device, enable and connect status.

7							0
—	—	—	RSTSC	—	PSSC	PESC	PCSC

Bit Number	Bit Mnemonic	Function
1	PESC	Port Enable/Disable Status Change (read, clear-only): This bit's status does not change due to USB requests. This bit is cleared by firmware via the USB host request ClearPortFeature with a C_PORT_ENABLE feature selector. '1' indicates port enabled/ disabled status change; '0' indicates no change. Port *x* (*x*=2,3,4,5): This bit is set by hardware due to hardware events only (this bit indicates the port was disabled due to babble, physical disconnects, or overcurrent). Port 1: This bit is set by hardware at the EOF2 point near the end of frame due to hardware events only (e.g., the port was disabled due to babble).
0	PCSC	Port Connect Status Change (read-, clear-only): This bit is cleared by firmware via a USB host request ClearPortFeature with C_PORT_CONNECTION feature selector. '1' indicates connect status change; '0' indicates no change. Port *x* (*x*=2,3,4,5): This bit is set by hardware at the EOF2 point near the end of a frame due to hardware connects and disconnects. Port 1: This bit is set by hardware at the next EOF2 after completion of a hub reset (since the internal port is always connected).

NOTE: Bits are returned as part of the second word (2 bytes) in response to a Get Port Status request from the USB host. The upper 11 MSbs are reserved and always '0' per USB 1.0: 0000 0000 000.4.3.2.1.0 (MSB at left).

Figure 7-11. HPSC: Hub Port Status Change Register (Continued)

NOTE

While the HPSC register indicates which port features have changed status, it does not show the current status of any feature. Firmware must examine the HPSTAT register to determine if a given port is currently reset, suspended, powered on or off, connected or disconnected, enabled or disabled. See "Examining a Port's Status Using HPSTAT" on page 7-17 for details.

7.5.4 Hub Port Indexing Using HPINDEX

A port indexing scheme is used for port-specific SFRs for reasons similar to those described in "Endpoint Selection" on page 5-4 for endpoint-specific registers. Three sets of SFRs have been mapped into the port-indexed scheme: HPSC, HPSTAT, and HPCON.

Ports 1-4 are indexed by the binary value of the two lower bits of HPINDEX (Figure 7-12). Port 0 is reserved for the root port, but it is not indexed by HPINDEX since there are no port-specific SFRs for the root port.

CAUTION

Firmware writers may choose to set the contents of HPINDEX once at the start of each routine instead of writing to HPINDEX prior to each access of a port-indexed SFR. Because of this, interrupt service routines must save the contents of the HPINDEX register at the start of the routine and restore the contents at the end of the ISR. This will prevent HPINDEX from being corrupted.

HPINDEX Address D4H
Reset State xxxx x000B

Hub Port Index Register. This register contains the binary value of the port whose HPSC, HPSTAT, and HPCON registers are to be accessed.

7							0
—	—	—	—	—	HPIDX2	HPIDX1	HPIDX0

Bit Number	Bit Mnemonic	Function
7:3	—	Reserved: Write zeros to these bits.
2:0	HPIDX.2:0	Port Index Select: Used to select the port to be indexed by the following registers: HPSC, HPSTAT, and HPCON. This register is hardware read-only. The ports are addressed using the following HPIDX2:0 bit combinations: Port 1 = "001" (internal port) Port 2 = "010" Port 3 = "011" Port 4 = "100" Port 5 = "101"

NOTE: Port 0 = "000" (the root port) and all other combinations not shown above are not valid port indexes and are ignored.

Figure 7-12. HPINDEX: Hub Port Index Register

7.5.5 Embedded Function

The following subsections discuss considerations involved with the embedded function on internal downstream port 1. See "Embedded Function Suspend and Resume" on page 7-26 for additional embedded function information.

7.5.5.1 Embedded Function Reset

The USB host can generate an embedded port reset command to the hub to reset the 8*x*931HA embedded function. When this command is received, the embedded function's EPCON, FIFLG, FIE, TXSTAT, RXSTAT, TXCON, RXCON, FADDR, and PCON1 SFRs are reset to their default values, as are the SOFACK, ASOF, SOFIE, and SOFODIS bits of SOFH. The EPINDEX and SOFL SFRs remain unchanged. These SFRs are reset immediately after the write to HPCON, however bus traffic to the embedded function remains inactive for 15 ms. You may use this time frame to initialize the embedded function.

After an embedded function reset, the internal function must be re-enumerated by the host. This procedure is given in "Enumeration" on page 8-2.

7.5.5.2 Embedded Function Remote Wake-up

The HRWUPE bit in HSTAT (Figure 7-6 on page 7-9) must be set in order for any downstream port to initiate resume signaling. This includes hub port 1, the internal downstream port. This port must be suspended and the HRWUPE bit in HSTAT must be set before the embedded function can initiate a remote wake-up. This is done by setting the RWU bit in PCON1 (Figure 14-2 on page 14-4).

7.6 SUSPEND AND RESUME

7.6.1 Hub Global Suspend and Resume

USB requirements state that a USB device must be capable of being placed in a low-power suspend mode in which the device draws less than 500 µA from the USB lines. The hub and the embedded function are placed in suspend mode when a continuous idle state of more than 3.0 msec is detected on the hub root port. For an in-depth discussion of 8*x*931 suspend and resume, see "USB Power Control" on page 14-7.

Once the suspend has been detected, the GSUS bit in the PCON1 SFR is set and a microcontroller interrupt is generated (See).

Firmware services the global suspend interrupt by setting the PD bit of the PCON SFR. This shuts off the device's clocks and crystal oscillator, placing the hub and embedded function in a USB suspend mode.

A resume event can be signaled in any of three ways:

1. The hub repeater asynchronously detects a resume state due to resume signaling or a connect/disconnect on the bus
2. The hub repeater detects a reset state on the bus' root port
3. An external interrupt powers-up the entire device, with a resume sequence initiated in firmware by setting the RWU bit in the PCON1 SFR (Figure 14-2 on page 14-4)

7.6.2 Remote Connectivity

During the suspend state of the hub, logical connectivity can also be established if a physical connection/disconnection is made on one of the downstream ports, or if a resume condition is signaled on a port, as shown in Figure 7-13 on page 7-26.

7.6.2.1 Resume Connectivity

The HRWUPE bit must be set in the HSTAT register (Figure 7-6 on page 7-9) before the connect or disconnect of a downstream device can initiate a remote wake-up. If this bit is not set, the downstream connect or disconnect will be ignored as a remote wake-up event. If a remote wake-up device signals a resume on a downstream port when the hub is in the suspend state (see Figure 7-13), the following process occurs:

1. The resume signaling causes the hub to wake up.

2. The repeater then establishes a connection from the port with the resume signal to the root port and all other enabled downstream ports.
3. The connectivity is then changed to downstream-only from the root port to all enabled downstream ports. This allows the host to drive the resume signaling downstream to the rest of the USB bus.

NOTE

The 8*x*931HA hub cannot request a remote wake-up, although its embedded function can. For this to happen, the HRWUPE bit must be set in HSTAT and the embedded function must be enabled. The embedded function triggers the remote wake-up by setting the RWU bit in PCON1.

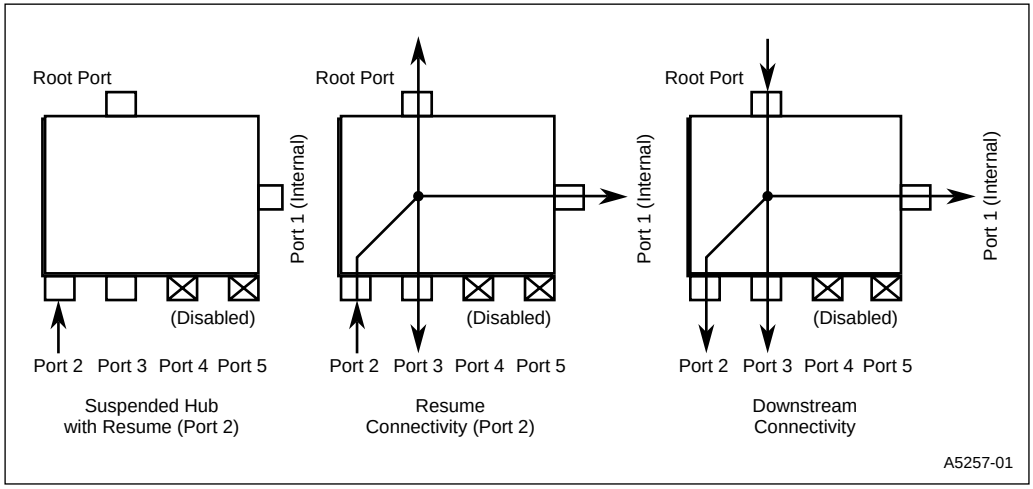


Figure 7-13. Resume Connectivity

7.6.2.2 Connectivity Due to Physical Connect/Disconnect

If a disconnect is made to a disabled port and the hub is in a global suspend state, a resume state is signaled as described in "Resume Connectivity" on page 7-25.

7.6.2.3 Embedded Function Suspend and Resume

Selective suspend is initiated on a downstream port when a SetPortFeature (suspend) command is received from the host via the USB bus. Individual external ports or the internal port can be suspended by USB command; however, the hub cannot be suspended by command. Refer to *Universal Serial Bus Specification* for more detail on the behavior of selective suspend in the USB system.

USB requirements state that the host can suspend the embedded function by issuing a SetPort-Feature (PORT_SUSPEND) request to the hub's port 1. Since the hub and function share hard-

ware such as the SIE, it is not possible to simply shut-off the clock to all circuitry associated with the function when the hub is to remain operational.

When placed into the suspended state, the embedded function must behave as if it were connected to a hub whose actual downstream port was suspended. This means that the embedded function must not respond to SOFs or any normal bus traffic. This is done automatically by hardware. Firmware should place any external circuitry associated with the embedded function in a low-power state, if one exists. The embedded function should remain in this suspended state until the host initiates a ClearPortFeature (PORT_SUSPEND) or a SetPortFeature (PORT_RESET) request to the hub, or until a remote wake-up is signaled by the embedded function via an external interrupt.

7.7 HUB POWER DISTRIBUTION

USB hubs can supply a specified amount of power to their downstream components and are responsible for reporting their power distribution capabilities to the host during enumeration. Hubs may be either locally powered, bus powered, or a combination of the two. The distinction is made depending on how the user implements the power scheme at the board level, which should be indicated in the value of the bmAttributes field of the configuration descriptor.

A hub can only supply power in a downstream direction and must never drive power upstream. Bus-powered hubs must have port power switching for the downstream ports and are required to power off all downstream ports when the hub comes out of power-up or when it receives a reset on its root port. Port power can also be switched on or off under control of the host PC. Port power switching is optional for self-powered devices.

NOTE

Port power switching and over-current detection (discussed in the following subsections) are mutually exclusive. Over-current detection is required only for self-powered hubs, while port power switching is required only for bus-powered hubs.

7.7.1 Port Power Switching

Port power switching is only supported on a ganged basis, therefore there is only one output pin used to enable power to the downstream devices.

From a USB perspective, power can be enabled on a per-port basis, but the power enable is active if any of the ports are powered-on by the host. The host PC can selectively switch power on or off for a given port using a Set_Feature request with a Port_Power feature selector. The 8*x*931 firmware must respond to this port power request by setting or clearing the appropriate bit in the HPPWR SFR (Figure 7-14). An exception to this is the internal downstream port, port 1, which is statically powered-on. The host PC may inquire about a port's power status using Get_Feature (Port_Power). Firmware must respond to this inquiry by checking and reporting on the PPSTAT bit (bit 4) of HPSTAT (see Section 7.5.2, Examining a Port's Status Using HPSTAT).

HPPWR Address: 9AH
Reset State: xxx0 001xB

Hub Port Power Control Register. This register is used to control power to the hub's downstream ports.

7							0
—	—	HPPWR5	HPPWR4	HPPWR3	HPPWR2	HPPWR1	—

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5:2	HPPWR5:2	Port Power Control for USB Ports 5-2: Bit 5 is power control for port 5, bit 4 for port 4, bit 3 for port 3, and bit 2 for port 2. These bits are set and cleared by firmware via a USB host request SetPortFeature with the PORT_POWER feature selector. These bits will also be cleared by hardware upon detection of an over-current condition. This is done to prevent oscillation of the UPWEN# pin during an over-current condition with bus-powered devices. A value of '1' enables power to the downstream port and puts the port in a disconnected state. A value of '0' turns the downstream port power off. **NOTE:** The UPWEN# pin is set to '1' only if all port power enable bits are '0,' due to the use of a ganged (shared) power enable scheme.
1	HPPWR1	Port Power Control for USB Port 1 (read-only): Port 1 is an internal port and is always powered on. This bit is hard-wired to '1.'
0	—	Reserved: Write a zero to this bit.

Figure 7-14. HPPWR: Hub Port Power Control

7.7.2 Overcurrent Detection

The OVRI# pin is an input pin that indicates when an overcurrent condition has been detected on one of the downstream devices at the board level. OVRI# is enabled by setting HSTAT.7 and is used to asynchronously disable the UPWEN# output pin, which switches power off to all external ports. When the overcurrent condition is removed, the OVRI# pin is deasserted to a ‘1’ state; however, the UPWEN# signal remains inactive since the HPPWR5:2 bits are reset when an overcurrent condition is detected (*unless* firmware has asserted one or more of these bits since the time the overcurrent was first detected). Due to the asynchronous nature of this signal, the user must be careful to guarantee that the OVRI# input is not “glitchy” or noisy, since glitches on this signal could have a detrimental impact on the system.

The state of the OVRI# pin can be read by the USB host via firmware, using the HSTAT (Figure 7-6 on page 7-9) OVI (HSTAT.1 — latest overcurrent indicator) and OVISC (HSTAT.3 — hub overcurrent status change) bits. OVI indicates if the overcurrent bit is presently asserted (‘0’) or de-asserted (‘1’). OVISC indicates whether the overcurrent status has changed since this bit was initially cleared by firmware (i.e., this bit acts as a “sticky” bit which must be cleared in firmware).

Another fact to consider about the overcurrent condition is that all external ports are placed in the “powered off” state. This is true for both bus-powered and self-powered ports, even though self-powered ports may still be powered. This condition will remain until the host enables power to the ports via one of the HPPWR5:2 bits. To disable OVRI# pin, clear OVRIEN (HSTAT.7 — overcurrent detect enable bit).

7.7.3 Ganged Power Enable

The 8*x*931HA uses a ganged power enable scheme to enable power to the external downstream ports. This means that a single output pin, UPWEN# should be used at the board-level to switch power to all of the downstream ports. The state of this power enable pin is controlled in two ways:

- by the collective ORed value of bits 5:2 of the HPPWR SFR (Figure 7-14) under control of firmware, and
- by the present state of the overcurrent sense input pin, OVRI#.

If any of the HPPWR bits are set, then the UPWEN# signal will be asserted (to a ‘0’) as long as the OVRI# signal is not asserted (i.e., OVRI # = ‘1’). If the OVRI# signal is asserted (‘0’), or if all of the power enable bits in HPPWR are cleared, then the UPWEN# signal will be deasserted (to a ‘1’).

Table 7-7 describes the state of the UPWEN# signal for all conditions of the HPPWR5:2 signals and the OVRI# pin. Port power enable bits in the HPPWR SFR (Figure 7-14 on page 7-28) are set via the SetPortFeature PORT_POWER request from the USB host. They are cleared via the ClearPortFeature PORT_POWER request, or by hardware upon detection of an overcurrent condition.

Table 7-7. UPWEN# Pin State Truth Table

HPPWR5	HPPWR4	HPPWR3	HPPWR2	OVRI#	UPWEN#
0 (disabled)	0 (disabled)	0 (disabled)	0 (disabled)	1 (disabled)	1 (disabled)
X	X	X	1	1	0 (enabled)
X	X	1	X	1	0
X	1	X	X	1	0
1	X	X	X	1	0
X	X	X	X	0 (enabled)	1

Since a single power enable output is used for all downstream ports, the value of the corresponding HPPWR bit does *not* necessarily reflect the actual state of the port power, since all HPPWR bits must be disabled for power to be disabled. Similarly, a '1' bit in the HPPWR SFR might not reflect that power is actually enabled to any devices in the event of an overcurrent condition.

Note that the power enable signal for the internal port, HPPWR1, does not affect the state of the UPWEN# pin. Also, note that bus-powered devices *must* use the UPWEN# signal to switch power to downstream ports, however, port power switching for self-powered devices is optional.

NOTE

Before the UPWEN# pin can be enabled, the EP1EN bit in the HSTAT SFR (Figure 7-6 on page 7-9) must be set. See Section 11.9 of the *Universal Serial Bus Specification*.

7.8 HUB DEVICE SIGNALS

Table 7-8 lists device signals associated with the hub. Pin assignments are shown in Appendix B.

Table 7-8. Signal Descriptions

Signal Name	Type	Description	Alternate Function
D_{P0}, D_{M0}	I/O	**USB (Upstream) Port 0**. D_{P0} and D_{M0} are the data plus and data minus lines of USB port 0. These lines do not have internal pullup resistors. For low-speed devices, provide an external 1.5 KΩ pullup resistor at D_{M0}. For full-speed devices, provide an external 1.5 KΩ pullup resistor at D_{P0}. NOTE: For the 8*x*931HA, provide an external 1.5 KΩ pullup resistor at D_{P0} so the device indicates to the host that it is a full-speed device.	—
D_{P2}, D_{M2} D_{P3}, D_{M3} D_{P4}, D_{M4} D_{P5}, D_{M5}	I/O	**USB Downstream Ports 2,3,4,5**. These pins are the data plus and data minus lines for the four USB external downstream ports. You must supply an external 15 KΩ pulldown resistor for these lines. If the USB downstream ports are not used, the two data lines are still required to be pulled low externally (similar to a disconnect) so that the inputs are not left floating.	—

8

USB Programming Models

CHAPTER 8
USB PROGRAMMING MODELS

This chapter describes the programming models of the USB function interface and the hub interface. It provides flow charts of firmware routines needed to perform data transfers between the host PC and the embedded function, as well as routines needed to handle hub-oriented USB requests. It also describes briefly how the firmware interacts with the USB module hardware during these operations. Data operations refer to data transfers over the USB, whereas event operations are hardware operations such as attach and detach. For a description of the USB function interface as well as its FIFOs and special functions registers (SFRs), refer to Chapter 6, “USB Function”. For further information about the USB hub interface, see Chapter 7, “USB Hub”. For details on data flow in USB transactions refer to Appendix D, “Data Flow Model”.

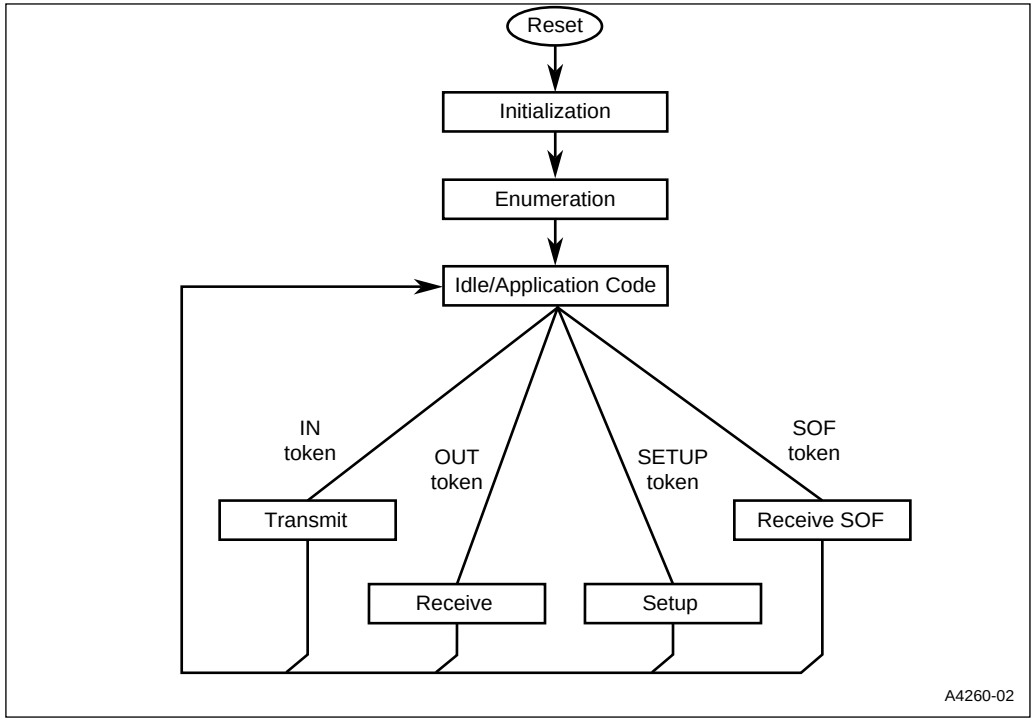


Figure 8-1. Program Flow

8.1 OVERVIEW OF PROGRAMMING MODELS

The USB function interface employs four types of routines: receive, transmit, setup, and receive SOF. Program flow is depicted in Figure 8-1 along with the type of token associated with each routine. Following device reset, the USB function enters the unenumerated state and after enumeration by the host, the idle state. From the idle state, it can enter any of the four routines.

8.1.1 Enumeration

Following device reset, the USB hub and function enter the unenumerated state. Initially, the hub address register HADDR and the function address register FADDR contain the default value 00H. The host PC performs bus enumeration at system start-up or whenever a new USB device is attached to the host or to a hub's downstream port. During bus enumeration the host identifies and addresses devices attached to the bus. During enumeration, a unique address assigned by the host is written to HADDR and FADDR.

NOTE

Since the 8*x*931AA microcontroller does not support a hub interface (and hence, has no HADDR SFR or downstream ports), its enumeration process is simpler. The 8*x*931AA enumeration process is given in Appendix E, "8x931AA Enumeration Process" on page E-2.

An example enumeration for the hub and downstream ports is given here:

1. Get device descriptor. The host requests and reads the device descriptor to determine maximum packet size.
2. Set address. The host sends the 8*x*931HA's hub address in a data packet using hub endpoint 0. Device firmware interprets the data and instructs the CPU to write the hub address to HADDR. See "The Hub Address Register (HADDR)" on page 7-8.
3. Get device descriptor. The host requests and reads the device descriptor to determine such information as device class, USB specification compliance level, maximum packet size for endpoint 0, vendor id, product id, etc. For additional information on the 8*x*931HA descriptors, see "Hub Descriptors" on page 7-7.
4. Get configuration descriptor. The host requests and reads the device's configuration descriptor to determine such information as the number of interfaces and endpoints; endpoint transfer type, packet size, and direction; power source; maximum power; etc. When the host requests the configuration descriptor, all related interface endpoint descriptors are returned. For additional information on the 8*x*931HA descriptors, see "Hub Descriptors" on page 7-7.

   NOTE

   Some versions of the operating system send a Get string descriptor at this point in the enumeration process.

5. Set configuration. The host assigns a configuration value to the device to establish the current configuration.
6. Get hub descriptor. The host requests and reads the hub descriptor to determine such information as number of downstream ports, hub characteristics, controller current, removable devices, etc. For additional information, see Table 7-3, "Hub Descriptors" .
7. Next, the hub downstream ports start the state flow shown in Figure 7-3 on page 7-5. The host issues a SetPortPowerFeature request to the downstream ports that were declared in the hub descriptor. This moves the hub downstream ports to the disconnect state.

8. As connect detects occur, the host is notified through hub endpoint 1 (status change endpoint). The host then issues a GetPortStatus command retrieving the contents of HPSTAT and HPSC to determine the change for a specific downstream port. The host then issues a ClearPortConnectionFeature command which should cause the firmware to clear the PCSC bit in the HPSC register. This will indirectly clear the appropriate bit in TXDAT for hub endpoint 1. This moves the hub downstream port to the disabled state.
9. The host sends a SetPortResetFeature request for the specified downstream port. The host receives a response through hub endpoint 1 (status change endpoint). The host issues a GetPortStatus command retrieving the contents of HPSTAT and HPSC to determine the change for the specified downstream port. The host then issues a ClearPortResetFeature command, causing firmware to clear the RSTSC bit in the HPSC register. This moves the hub downstream port to the enabled state.
10. At this point, the downstream ports go through the function enumeration process, beginning with the embedded function:
    a. Get descriptor. The host requests and reads the device descriptor to determine such information as device class, USB specification compliance level, maximum packet size for endpoint 0, vendor id, product id, etc. For detailed information on device descriptors, see the “Device Framework” chapter in *Universal Serial Bus Specification*.
    b. Set address. The host sends the function address in a data packet using function endpoint 0. Device firmware interprets the data and instructs the CPU to write the function address to FADDR.
    c. Get configuration. The host requests and reads the device configuration descriptor to determine such information as the number of interfaces and endpoints; endpoint transfer type, packet size, and direction; power source; maximum power; etc. For detailed information on configuration descriptors, see the “Device Framework” chapter in *Universal Serial Bus Specification*. When the host requests the configuration descriptor, all related interface and endpoint descriptors are returned.
    d. Set configuration. The host assigns a configuration value to the device to establish the current configuration. Devices can have multiple configurations.
11. The external ports must go through steps 8 through 10.

8.1.2 Idle State

Following bus enumeration, the USB function enters the idle state. In this state, the 8*x*931 executes application code associated with the embedded function. Upon receipt of a token with the assigned address, the module enters the designated routine. The 8*x*931 remains in the idle state when not processing USB transmissions.

8.1.3 Transmit and Receive Routines

When the 8*x*931 is sending and receiving packets in the transmit and receive modes, its operation depends on the type of data that is transferred—isochronous or non-isochronous—and the adjustment of the FIFO markers and pointers—automatic or manual. These differences affect both the 8*x*931 firmware and the operation of the 8*x*931 hardware. For isochronous data, a failed transfer is not retried (lossy data). For non-isochronous data, a failed transfer can be repeated. Data that

can be repeated is considered lossless data. Automatic adjustment of the FIFO markers and pointers is accomplished by the function interface hardware. Manual adjustment is accomplished by the 8*x*931 firmware.

8.1.4 USB Interrupts

For an explanation of the USB global suspend/resume, function, hub, and SOF interrupts, see Chapter 5, "Interrupt System".

8.2 TRANSMIT OPERATIONS

8.2.1 Overview

A transmit operation occurs in three major steps:

1. Pre-transmit data preparation by firmware
2. Data packet transmission by function interface hardware
3. Post-transmit management by firmware

These steps are depicted in a high-level view of transmit operations (Figure 8-2). The pre-transmit and post-transmit operations are executed by the two firmware routines shown on the left side of the figure. Function interface hardware (right side of the figure) transmits the data packet over the USB line. Details of these operations are described in "Pre-transmit Operations" on page 8-7 and "Post-transmit Operations" on page 8-8.

Transmit operations for non-isochronous data begin with an interrupt request from the embedded function (e.g., a keyboard entry). The pre-transmit routine (ISR) for the function writes the data from the function to the transmit FIFO where it is held until the next IN token. Upon receipt of the next valid IN token, the function interface shifts the data out of the FIFO and transmits it over the USB. If the data packet is not ready for transmission, 8*x*931 hardware responds to the IN token with a NAK. The post-transmit routine checks the transmission status and performs data management tasks.

Completion of data transmission is indicated by a handshake returned by the host. This is then used to generate a transmit done interrupt to signal the end of data transmission to the CPU. The interrupt can also be used for activity tracking and fail-safe management. Fail-safe management permits recovery from lockups that can only be cleared by firmware.

Because a transmit done interrupt is generated regardless of transmission errors, this condition means either:

1. The transmit data has been transmitted and the host has sent an acknowledgment to indicate that is was successfully received; or
2. A transmit data error occurred during transmission of the data packet, which requires servicing by firmware to be cleared. You must check for these conditions and respond accordingly in the ISR.

For ISO data transmission, the cycle is similar. The significant differences are: the cycle is initiated by a start-of-frame (SOF) interrupt, there is no handshake associated with ISO transfer, and a transmit done interrupt is not generated. For ISO data transfers, the transaction status is updated at the end of the USB frame. The 8*x*931 supports one ISO packet per frame per endpoint.

Two bits in the transmit FIFO control register (TXCON, Figure 6-10 on page 6-19) have a major influence on transmit operation:

- The TXISO bit (TXCON.3) determines whether the transmission is for isochronous data (TXISO = 1) or non-isochronous data (TXISO = 0). For non-isochronous data only, the function interface receives a handshake from the host, toggles or does not toggle the sequence bit, and generates a transmission done interrupt (Figure 8-2). Also, for non-isochronous data, the post-transmit routine is an ISR; for isochronous data the post-transmit routine is an ISR initiated by an SOF token.
- The ATM bit (TXCON.2) determines whether the FIFO read marker and read pointer are managed automatically by the FIFO hardware (ATM = 1) or manually by the second firmware routine (ATM = 0). Use of the ATM mode is recommended. The ADVRM and REVRP bits, which control the read marker and read pointer when ATM = 0, are used primarily for test purposes. See bit definitions in TXCON (Figure 5-12).

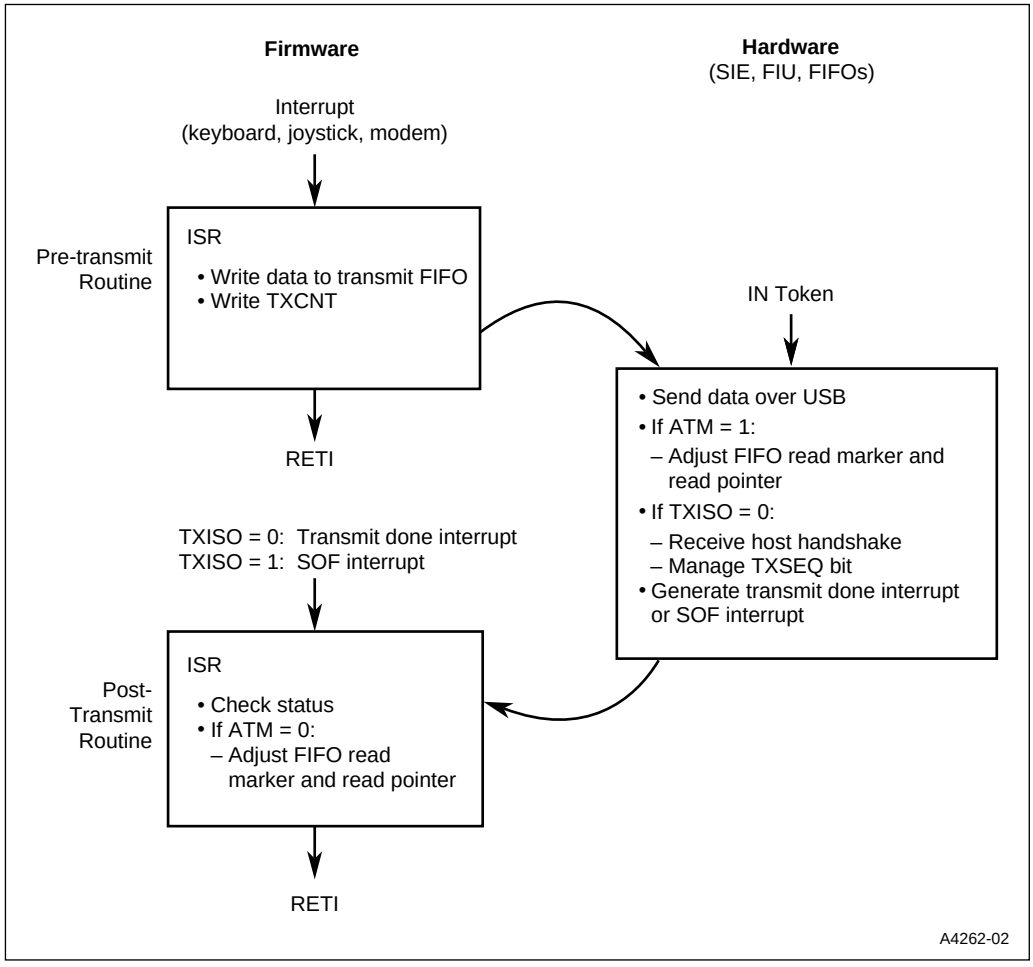


Figure 8-2. High-level View of Transmit Operations

8.2.2 Pre-transmit Operations

Transmitted data originates in the embedded function, which might be a keyboard, mouse, joystick, scanner, etc. In event-control applications, the end function signals the availability of data with an interrupt request for the pre-transmit interrupt service routine (ISR). The ISR should prepare the data for transmission and initiate the transmission process. The flow chart in Figure 8-3 illustrates a typical pre-transmit ISR.

For the case of isochronous data, the interrupt is triggered by the USB function in response to a start-of-frame (SOF) packet.

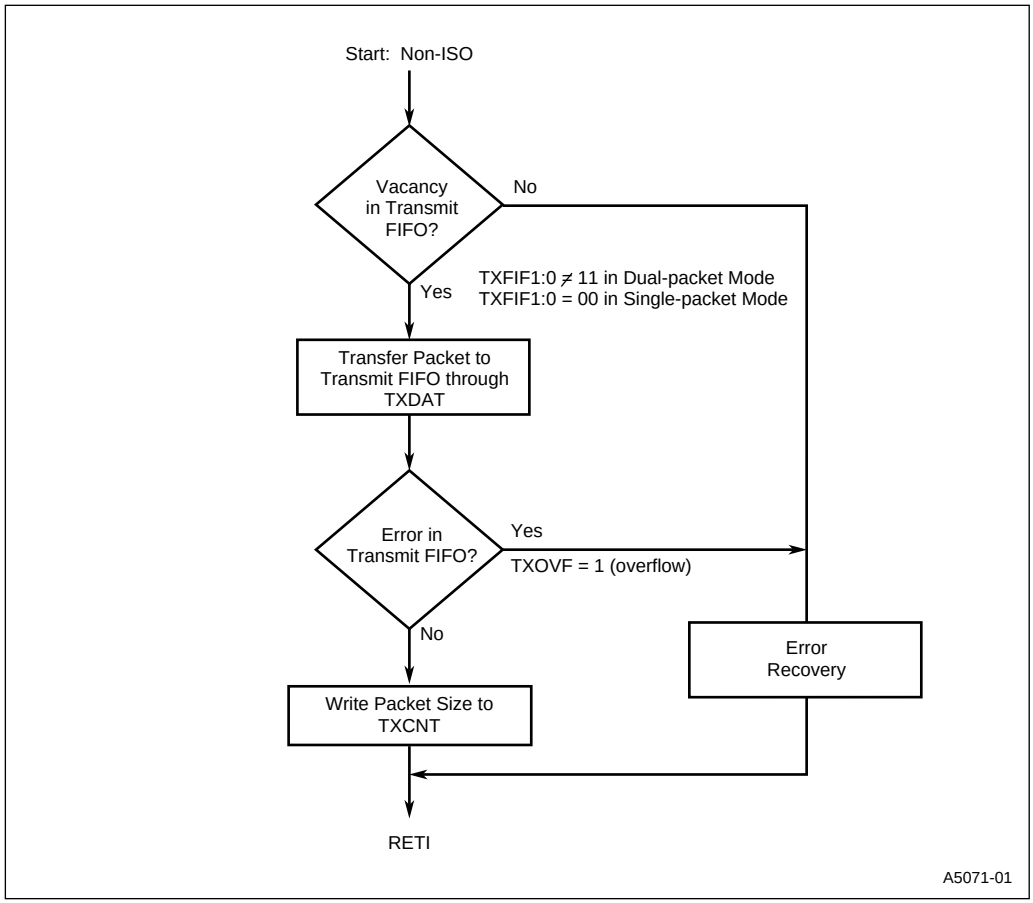


Figure 8-3. Pre-transmit ISR (Non-Isochronous)

8.2.3 Post-transmit Operations

Transmission status is updated at the end of data transmission based on the handshake received from the host (non-isochronous data) or based on the transmission process itself (isochronous data). For a non-isochronous transfer, the function interface generates a transmit done interrupt. The purpose of the post-transmit service routines is to manage the transmitter's state and to ensure data integrity for the next transmission. For isochronous data, the post-transmit routine should be embedded within the transfer request routine because both are triggered by an SOF. The flow of operations of typical post-transmit ISRs is illustrated in Figure 8-4 (non-isochronous data) and Figure 8-5 (isochronous data).

Figure 8-4. Post-transmit ISR (Non-isochronous)

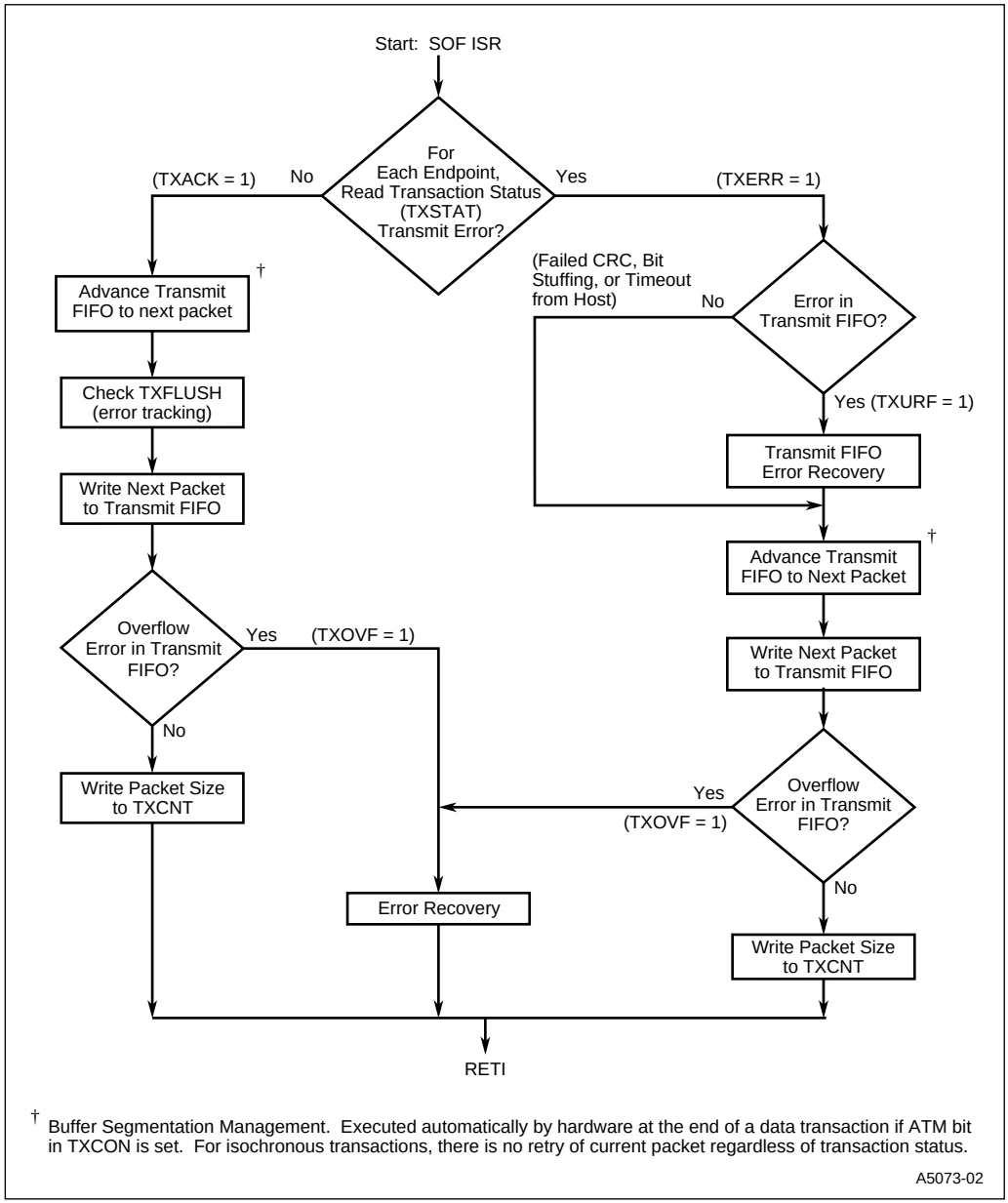


Figure 8-5. Post-transmit ISR (Isochronous)

8.3 RECEIVE OPERATIONS

8.3.1 Overview

A receive operation is always initiated by the host, which sends an OUT token to the 8*x*931. The operation occurs in two major steps:

1. Data packet reception by the function interface (hardware)
2. Post-receive management by firmware

These steps are depicted in a high-level view of the receive operations in Figure 8-6. The post-receive operations are executed by the firmware routine shown on the left side of the figure. For details see "Post-receive Operations" on page 8-11. Function interface hardware (right side of figure) receives the data packet over the USB line.

Receive operations for non-isochronous data begin when the 8*x*931 receives a valid OUT token from the host. The received data is written to a data buffer FIFO. The 8*x*931 indicates completion of data received by returning a handshake to the host.

At the end of the receive cycle, the 8*x*931 generates a receive done interrupt to notify the CPU that a receive operation has occurred. Program execution branches to the interrupt service routine and transfers the data packet from the receive FIFO to its destination. The interrupt can also be used for fail-safe management and activity tracking.

For isochronous data, receive cycles are somewhat different. Data transactions are initiated by an OUT token. At the end of the OUT transaction, the 8*x*931 does not return handshake to the host and the receive done interrupt is not generated. Instead, the SOF interrupt is used for post receive management. The data reception status is updated at the next SOF. The 8*x*931 supports one ISO packet per frame per endpoint.

Two bits in the receive FIFO control register (RXCON, Figure 6-15 on page 6-29) have a major influence on receive operation:

- The ISO bit (RXCON.3) determines whether the reception is for isochronous data (ISO = 1) or non-isochronous data (ISO = 0). For non-isochronous data only, the function interface sends a handshake to the host, checks the sequence bit, and generates a receive-done (FRXD*x*) interrupt. Also, for non-isochronous data, the post-receive routine is an ISR; for isochronous data the post-receive routine can be a normal subroutine or ISR initiated by an SOF token.
- The ARM bit (RXCON.2) determines whether the FIFO write marker and write pointer are managed automatically by the FIFO hardware (ARM = 1) or manually by the firmware routine (ARM = 0). Use of the ARM mode is recommended. The ADVWM and REVWP bits, which control the write marker and write pointer when ARM = 0, are used primarily for test purposes. See bit definitions in RXCON (Figure 6-15 on page 6-29).

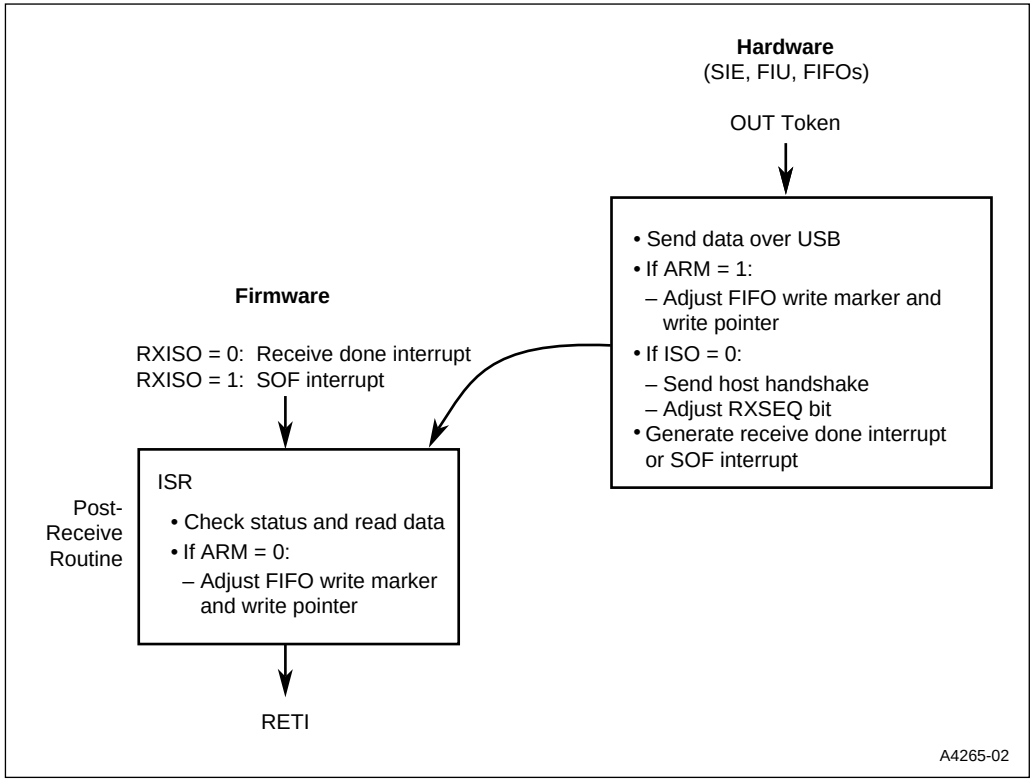


Figure 8-6. High-level View of Receive Operations

8.3.2 Post-receive Operations

Reception status is updated at the end of data reception based on the handshake received from the host (non-isochronous data) or based on the transmission process itself (isochronous data). For a non-isochronous transfer, the function interface generates a receive done interrupt (FRXD*x*). The purpose of the post-receive service routine is to manage the receiver's state to ensure data integrity and latency for the next reception. The post-receive routine also transfers the data in the receive FIFO to the end function. For isochronous data, the post-receive routine should be called by the SOF ISR.

Flow diagrams for typical post-receive routines are presented in Figure 8-7 (non-isochronous data) and Figure 8-8 (isochronous data).

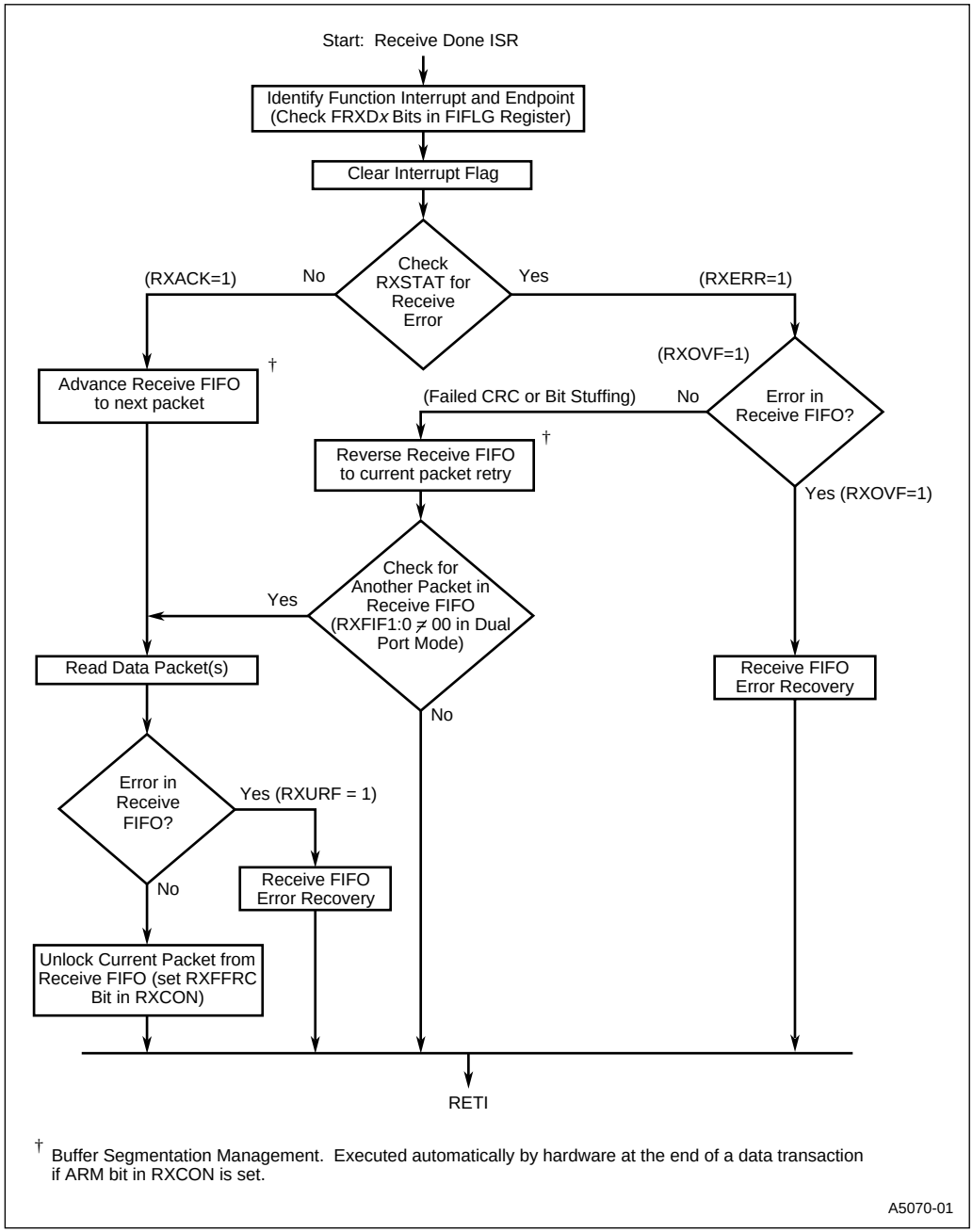
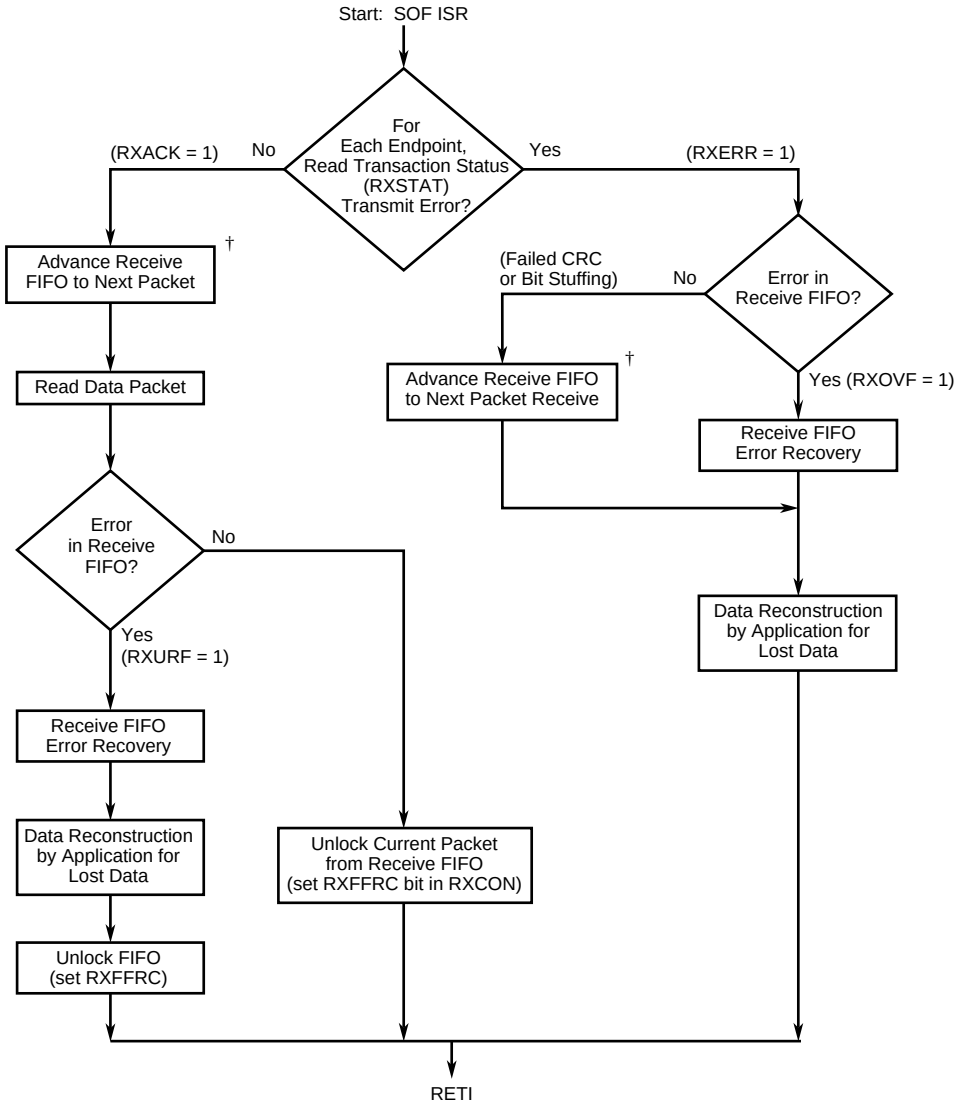


Figure 8-7. Post-receive ISR (Non-isochronous)



† Buffer Segmentation Management. Executed automatically by hardware at the end of a data transaction if ARM bit in TXCON is set. For isochronous transactions, there is no retry of current packet regardless of transaction status.

A5074-01

Figure 8-8. Receive SOF ISR (Isochronous)

8.4 SETUP TOKEN

An endpoint must be configured as a control endpoint in order to respond to SETUP tokens. (This will only be endpoint 0, since it must serve as a control endpoint.) Refer to the "Protocol Layer" section of the *Universal Serial Bus Specification* for details of SETUP token transactions and protocol.

A control data transfer is initiated by a valid SETUP token (i.e., the token PID received is good). Receive data transfer operations for a control endpoint are very similar to data transfers on non-control endpoints for non-setup tokens. However, the response of a control endpoint is different when it receives a setup token.

USB protocol specifies that setup tokens must be received and ACKed. Following receipt of a setup token, a control endpoint flushes the contents of the receive FIFO before writing it with received setup data. This may create an error condition in the FIFO due to the asynchronous nature of FIFO reads by the CPU and simultaneous writes by the function interface. To prevent this, STOVW and EDOVW are used to track when an overwrite is occurring. When the overwrite is complete, the user must clear EDOVW to read the SETUP packet. If EDOVW is not cleared, user firmware will only be able to read the first byte of the SETUP packet. Figure 8-9 illustrates the operations of a typical post-receive routine for a control endpoint.

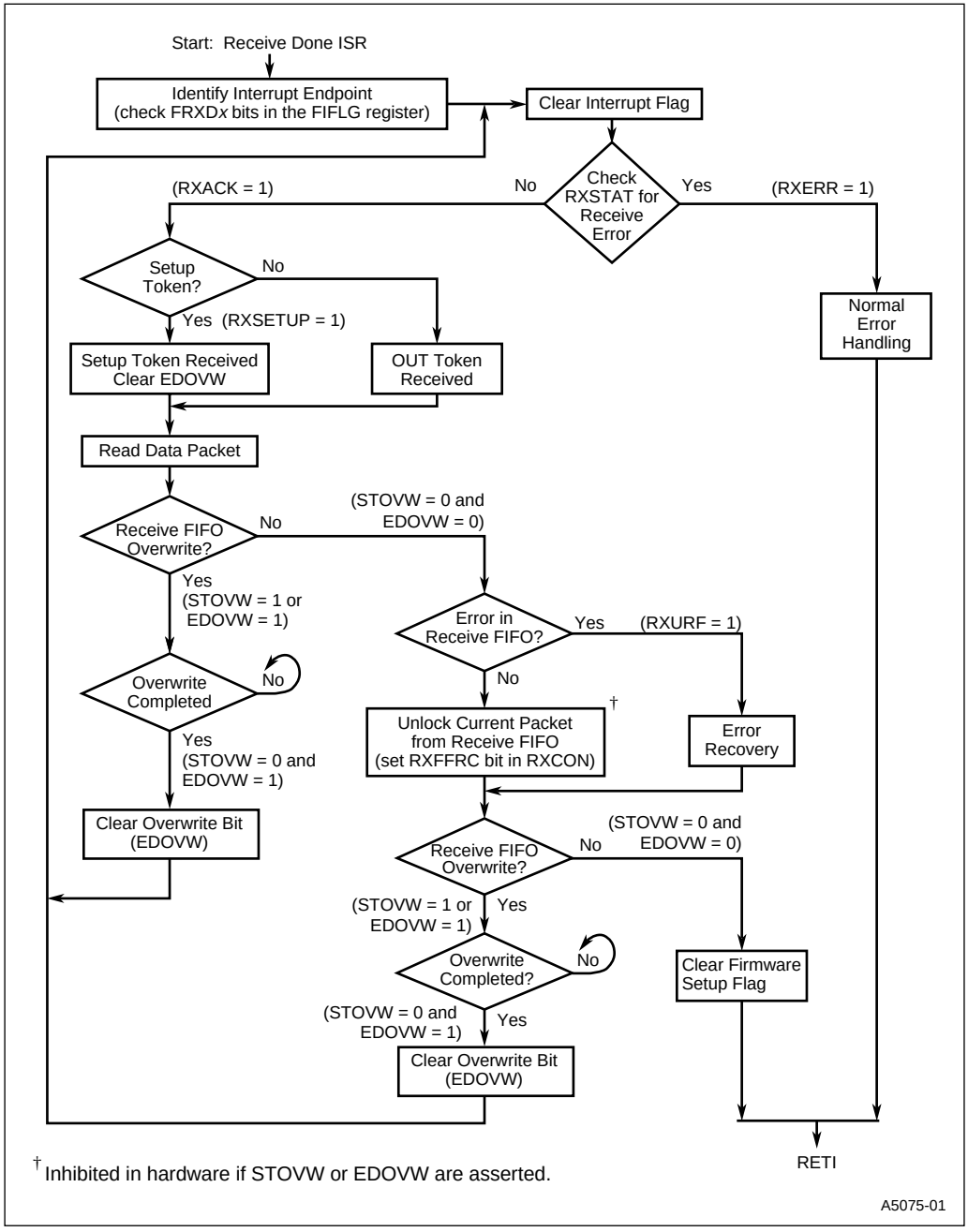


Figure 8-9. Post-receive ISR (Control)

8.5 START-OF-FRAME (SOF) TOKEN

Figure 8-10 illustrates the hardware operations performed by the function interface for a start-of-frame (SOF) token. The host issues an SOF token at a nominal rate of once every 1.0 ms. An SOF token is valid if the PID is good. The SOF token is not endpoint-specific; it should be received by every node on the bus.

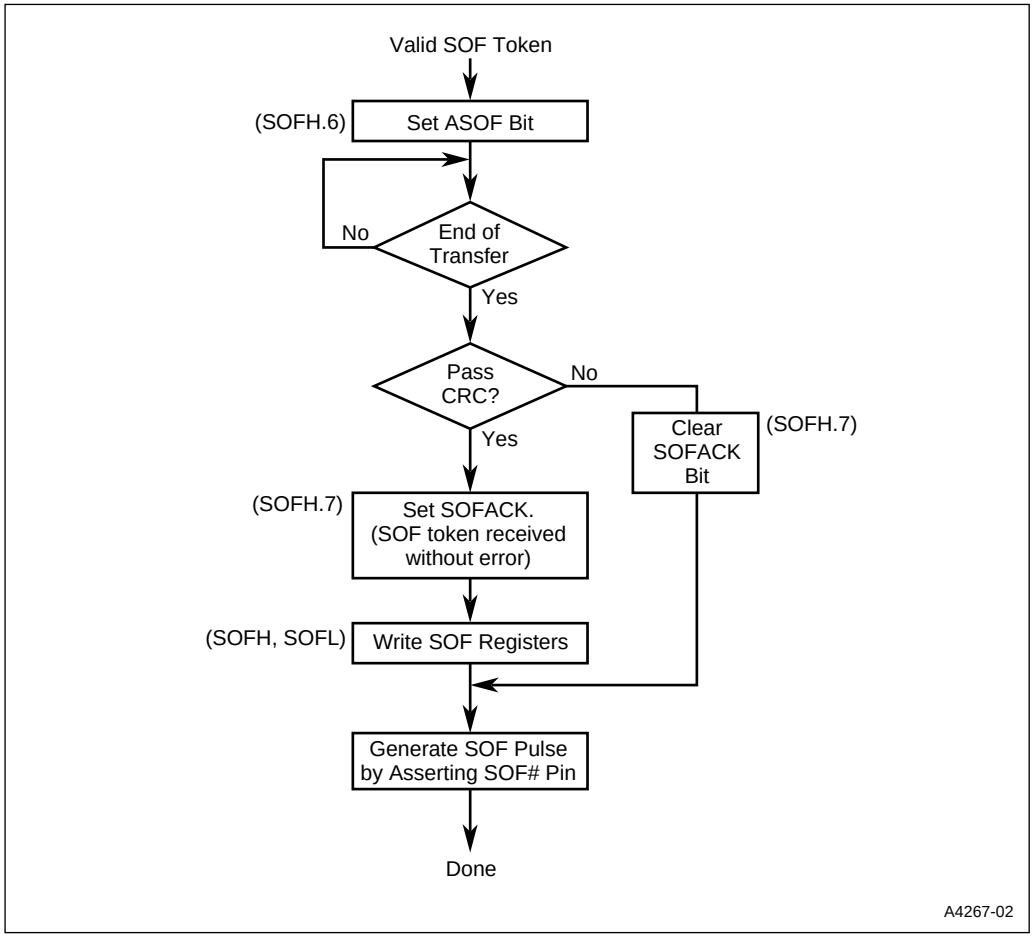


Figure 8-10. Hardware Operations for SOF Token

8.6 HUB OPERATION

The primary objective of the programming model suggested here is to explain the linkage between the hardware and firmware of the 8*x*931HA in operation.

NOTE

Since the 8*x*931AA microprocessor does not support a hub interface, the programming models in this section are unnecessary. Specific details of the 8*x*931AA are covered in Appendix E, “8x931AA Design Considerations”.

8.6.1 Hub Status and Configuration

USB communication with the USB hub function is performed via the standard and hub class-specific USB requests. These requests control status management and configuration of the hub and its downstream ports. Since the hub is part of a compound device, it has an internal downstream port (port 1) which is unique from the external downstream ports. This is because port 1 is physically connected to the embedded function and is powered-on at all times. Thus several USB requests intended for internal downstream port 1 are handled differently from similar requests to the other downstream ports, as shown in Table 7-6 on page 7-16.

Table 8-1 is a summary of firmware actions required for standard USB requests sent to hub endpoint 0.

Table 8-1. Firmware Actions for USB Requests Sent to Hub

USB Request	Feature Selector / Type	Firmware Action Required
SET_FEATURE	DEVICE_REMOTE_WAKEUP	Set the HRWUPE bit of the HSTAT SFR. See “Hub Status” on page 7-9.
	ENDPOINT_STALL	Stall the endpoint specified in the Setup PID. See “Hub Endpoint Control” on page 7-11. Endpoint 0 specified: 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set RXSTL and TXSTL bits of EPCON SFR. Endpoint 1 specified: Set EP1STL bit of HSTAT SFR.
CLEAR_FEATURE	DEVICE_REMOTE_WAKEUP	Clear HRWUPE bit of HSTAT SFR. See “Hub Status and Configuration” on page 8-17.
	ENDPOINT_STALL	Cancel stall for the specified endpoint. See “Hub Endpoint Control” on page 7-11. Endpoint 0 specified: 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Clear RXSTL and TXSTL bits of EPCON SFR Endpoint 1 specified: Clear EP1STL bit of HSTAT SFR.

Table 8-1. Firmware Actions for USB Requests Sent to Hub (Continued)

USB Request	Feature Selector / Type	Firmware Action Required
SET_CONFIGURATION	N/A	1. Store hub endpoint 1 configuration value from value field in memory 2. Set EP1EN bit of HSTAT SFR (Figure 7-6 on page 7-9) after the Status stage if 2-byte configuration value = 0001H
GET_CONFIGURATION	N/A	Read configuration value (one byte) from memory and send to the host.
GET_DESCRIPTOR	Device	Read device descriptor from memory and transmit to USB host through hub endpoint 0.
	Configuration	Read configuration, interface, endpoint, and hub descriptors from memory and transmit to USB host through hub endpoint 0.
GET_INTERFACE	N/A	Optional request for hubs which is not supported. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
GET_STATUS	Device	Read HSTAT SFR bit HRWUPE (Figure 7-6 on page 7-9) and power configuration from memory and transmit to USB host using hub endpoint 0.
	Interface	Load 2 bytes of zero into transmit buffer and transmit to USB host. These bits are reserved in the initial version of USB.
	Endpoint	Endpoint 0 specified: Load transmit buffer with value of zero if endpoint 0 is not stalled. No data can be returned if endpoint 0 is stalled, since STALL will be transmitted instead. Endpoint 1 specified: Load value of EP1STL bit of HSTAT SFR into transmit buffer (Figure 7-6 on page 7-9).
SET_ADDRESS	N/A	Read address value contained in request value field and store in HADDR SFR (Figure 7-5 on page 7-8) after successful completion of control transaction status stage.
SET_DESCRIPTOR	N/A	Optional request for hubs which is not supported. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
SET_INTERFACE	N/A	Optional request for hubs which is not supported. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
SYNCH_FRAME	N/A	Optional request for hubs which is not supported. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage

Table 8-2 summarizes firmware action for hub class-specific USB requests.

NOTE

Upon receipt of a state-related USB request (i.e., SetPortFeature, ClearPortFeature), firmware must examine the HPSTAT SFR to determine the current port state. If the port is in a state where the request will be ignored by hardware, instead of performing the action given in Table 8-2, firmware must respond to the host by sending a STALL during the transaction status stage to indicate the command was not completed. Table 7-6 on page 7-16 depicts the state-related USB requests and the port states for which they are ignored. See“Controlling a Port Using HPCON” on page 7-14 for additional information.

Table 8-2. Firmware Action for Hub Class-Specific Requests

USB Requests	Feature Selector / Type / Index	Firmware Action Required
SetHubFeature	—	Unsupported request since there are no current feature selectors to match this request in the initial version of USB. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
ClearHubFeature	C_HUB_OVER_CURRENT	Clear HSTAT SFR bit OVISC (hub over-current status change bit). HSTAT is shown in Figure 7-6 on page 7-9.
	C_HUB_LOCAL_POWER	Unsupported request. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
GetBusState	Port number	Transfer the port bus signal values (D_P and $D_{M)}$ to the host for diagnostic purposes. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Transfer the DPSTAT and DMSTAT bits of HPSTAT (Figure 7-10 on page 7-18) to the transmit buffer of hub endpoint 0. Transmit these bits in a single byte, with DMSTAT as bit 0, DPSTAT as bit 1, and bits 2-7 as '0'.
GetHubDescriptor	N/A	Read hub descriptor from memory and transmit to USB host using hub endpoint 0.
SetHubDescriptor	N/A	Optional request for hubs which is not supported. 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
GetHubStatus	N/A	Communicate the hub over-current status change, local power status change, current-overcurrent indicator, and current local power status to the host: Load HSTAT bits OVISC and OVI into transmit buffer, with LPS as the LSb. The HSTAT SFR is shown in Figure 7-6 on page 7-9.
GetPortStatus	Port number	Load the indicated port's HPSTAT and HPSC SFRs into the transmit buffer. See "GetPortStatus Request Firmware" on page 8-25 for additional information, including bit ordering and a flowchart.

Table 8-2. Firmware Action for Hub Class-Specific Requests (Continued)

USB Requests	Feature Selector / Type / Index	Firmware Action Required
SetPortFeature	PORT_ENABLE	Enables address and endpoint decoding for the downstream ports. For hub port 1, this enables address and endpoint decoding for the embedded function. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Write “001” to bits 2:0 of the port’s HPCON SFR (Figure 7-9 on page 7-15)
	PORT_SUSPEND	Write “011” to bits 2:0 of the port’s HPCON SFR. If hub port 1 is specified, the user cannot suspend the embedded function without also suspending the hub. Firmware must suspend any non-hub functionality associated with the embedded function prior to writing to HPCON. This is done by placing any external device hardware into a low-power suspend mode. See “SetPortFeature (PORT_SUSPEND) Firmware” on page 8-26 for additional information and a flowchart.
	PORT_RESET	Write “010” to bits 2:0 of the port’s HPCON SFR. If port 1 is specified, firmware needs to reset all non-hub functionality in the microcontroller. Upon writing to the embedded function’s HPCON SFR, a hardware reset is generated for the FIU and function FIFOs. Firmware must gracefully shut-down the application code, peripherals, etc. prior to writing to port 1’s HPCON. Once written, the reset will be active in hardware for 10-11 ms. See “SetPortFeature (PORT_RESET) Firmware” on page 8-27 for additional information and a flowchart.
	PORT_POWER	Set bit ***x*** of HPPWR (where ***x*** is the port specified in the request index field) Port power-on is also supported for port 1, but only for reasons of port compatibility since power for the embedded function cannot be switched (i.e., writing bit 1 of HPPWR does not affect any hardware).

Table 8-2. Firmware Action for Hub Class-Specific Requests (Continued)

USB Requests	Feature Selector / Type / Index	Firmware Action Required
ClearPortFeature	PORT_ENABLE	Requests port disable. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Write "000" to bits 2:0 of the port's HPCON SFR (Figure 7-9 on page 7-15) For hub port 1, this will disable address and endpoint decoding for the embedded function.
	PORT_SUSPEND	Requests port resume. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Write "100" to bits 2:0 of the port's HPCON SFR (Figure 7-9 on page 7-15) If port 1 is specified, firmware must also resume any non-hub functionality associated with the embedded function prior to writing to port 1's HPCON. This requires taking any external device hardware out of a low-power suspend mode.
	PORT_POWER	Request port power off. If any port other than port 1 is specified: Clear bit ***x*** of HPPWR (where ***x*** is the port specified in the request index field) Port power off is not supported for port 1. If port 1 is specified: 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
	C_PORT_ CONNECTION	Request to clear port connect status change. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Clear PCSC bit of HPSC SFR (Figure 7-11 on page 7-21)
	C_PORT_ ENABLE	Request to clear hardware-initiated port enable/disable status change. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Clear PESC bit of HPSC SFR (Figure 7-11 on page 7-21)
	C_PORT_ SUSPEND	Request to clear port suspend status change. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Clear PSSC bit of HPSC SFR (Figure 7-11 on page 7-21)
	C_PORT_ OVERCURRENT	Unsupported request to clear port over-current status change. The 8*x*931HA implements over-current detection on a hub-wide basis, not on a per-port basis. If received: 1. Load 80H into EPINDEX (for hub endpoint 0) 2. Set TXSTL bit of EPCON SFR so STALL is sent during status stage
	C_PORT_RESET	Request to clear port reset status change. 1. Load ***xxx***B into HPINDEX2:0, where ***xxx*** is the binary representation of the port index 2. Clear RSTSC bit of HPSC SFR (Figure 7-11 on page 7-21)

8.6.2 Port Status Change Communication

The flowchart in Figure 8-11 shows how the hub communicates a change in port status to the host. This process involves 8*x*931HA hardware, 8*x*931HA firmware, and PC host firmware. The flowchart illustrates the complete process at a high level. The process contains the following steps:

1. Any change in a port's reset, suspend, enable, or connect status is communicated to the host via hub endpoint 1's TXDAT register, as shown in Figure 7-8 on page 7-13. The information passed through hub endpoint 1 is sufficient to indicate which port (or the hub itself) changed status, but it does not indicate which status value changed or the current value of any status indicator.

   Firmware has no involvement with USB communication to hub endpoint 1 (status change endpoint 1). This communication is handled completely in hardware and is discussed in "USB Hub Endpoints" on page 7-10.

2. After the host receives notice of a change in port status through hub endpoint 1, host firmware can determine which status value changed and the current value of all the port's status indicators by transmitting a GetPortStatus request through hub endpoint 0. This request includes a Port_Index to tell the 8*x*931HA which port is of interest to the host. See the *Universal Serial Bus Specification* for additional information.

3. The host's GetPortStatus request triggers the 8*x*931HA GetPortStatus routine. The firmware response to the GetPortStatus request provides the host with the port's current status along with an indication of any status changes that have occurred. See "GetPortStatus Request Firmware" on page 8-25 for a complete description of this routine.

4. The host resets the port status change indicators by issuing a separate ClearPortFeature request for each bit in HPSC that showed a change. Each ClearPortFeature request will include one of the following feature selectors:
   a. C_PORT_CONNECTION — to clear HPSC.PCSC
   b. C_PORT_ENABLE — to clear HPSC.PESC
   c. C_PORT_SUSPEND— to clear HPSC.PSSC
   d. C_PORT_RESET — to clear HPSC.RSTSC

5. 8*x*931HA firmware responds to each ClearPortFeature request by performing the actions shown in Table 8-2 on page 8-20.

6. Finally, the host must perform any actions necessitated by the status change.

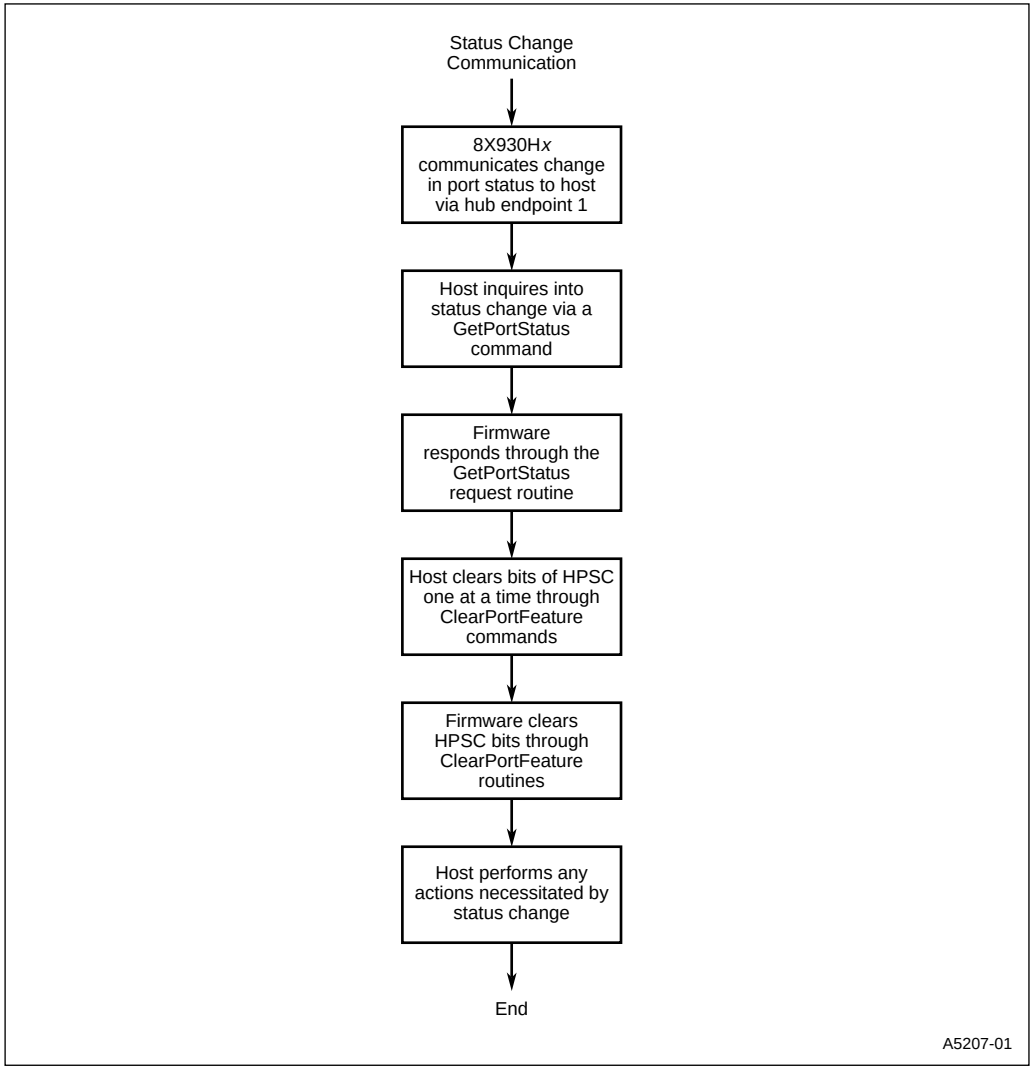


Figure 8-11. Hub-to-Host Port Status Communication

8.6.3 Hub Firmware Examples

Several of the firmware routines given in Table 8-2 have been selected as examples. The remaining routines should be coded similarly. The following subsections contain a flowchart and an additional explanation for these routines:

- GetPortStatus (Port_Index)
- SetPortFeature (PORT_SUSPEND)
- SetPortFeature (PORT_RESET)

8.6.3.1 GetPortStatus Request Firmware

Firmware responds to a GetPortStatus call by returning four bytes to the host using the flowchart procedure shown in Figure 8-12. The four bytes are arranged into a two-byte port status field and a two-byte port change field containing the contents of the HPSTAT and HPSC SFRs, respectively. Figure 8-13 shows the relationship between the four bytes returned by firmware and the contents of the HPSTAT and HPSC registers.

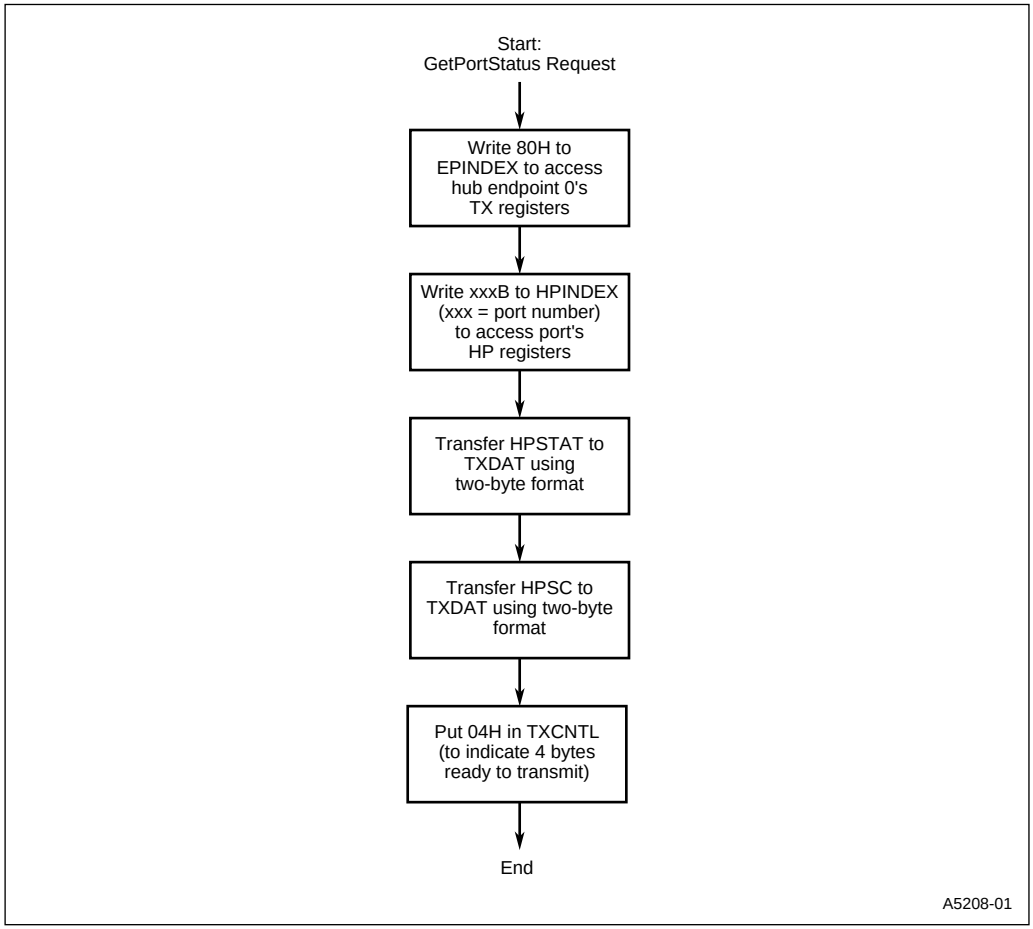


Figure 8-12. GetPortStatus Request

Transferring the contents of HPSTAT and HPSC into TXDAT requires additional code not shown in the flowchart. The bits of HPSTAT must be "converted" into a two-byte port status field, as shown in Figure 8-13, and transmitted to the host LSB first. The bits of HPSC must also be transmitted in a two-byte format, called the port change field.

The bit names are given in Figure 8-13, along with their position in the register (shown below the bit name) and their position in the transmitted two-byte field (shown above the bit names). Firmware must transmit the four bytes to the host in the byte order indicated (above the bit position).

NOTE

The HPSTAT bits are not directly mapped into the port status field. Firmware must clear bit three of byte one to indicate that power is normal (not overcurrent) for the port. This is done because the 8*x*931HA indicates overcurrent on a ganged, not per-port, basis.

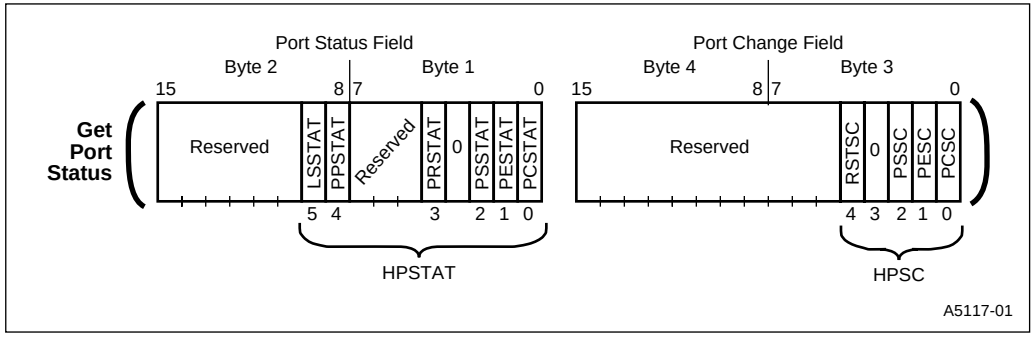


Figure 8-13. Firmware Response to GetPortStatus

8.6.3.2 SetPortFeature (PORT_SUSPEND) Firmware

This USB request suspends the downstream ports. The number of the port to be suspended is included in the request from the host. If hub port 4 is specified, firmware must also suspend any non-hub functionality associated with the embedded function and place any external device hardware into low-power suspend mode prior to writing to hub port 4's HPCON SFR.

To implement this routine, firmware must write "011" to bits 2:0 of the port's HPCON SFR. The flowchart in Figure 8-14 illustrates the process.

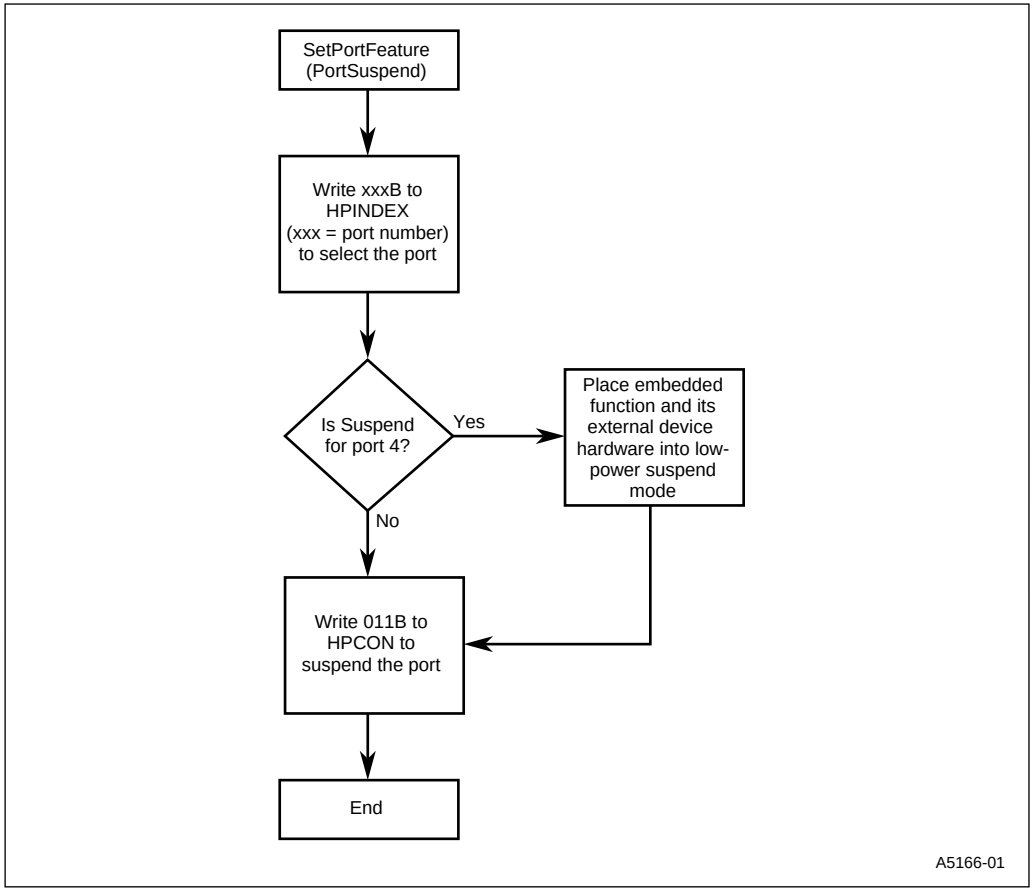


Figure 8-14. SetPortFeature (PORT_SUSPEND) Routine

8.6.3.3 SetPortFeature (PORT_RESET) Firmware

This USB request resets the downstream ports. The number of the port to be reset is included in the request from the host. To implement this routine, firmware must write “010” to bits 2:0 of the port’s HPCON SFR. The flowchart in Figure 8-15 illustrates the process. Refer to Section 11.6.2 of the *Universal Serial Bus Specification* for a detailed description of this USB command.

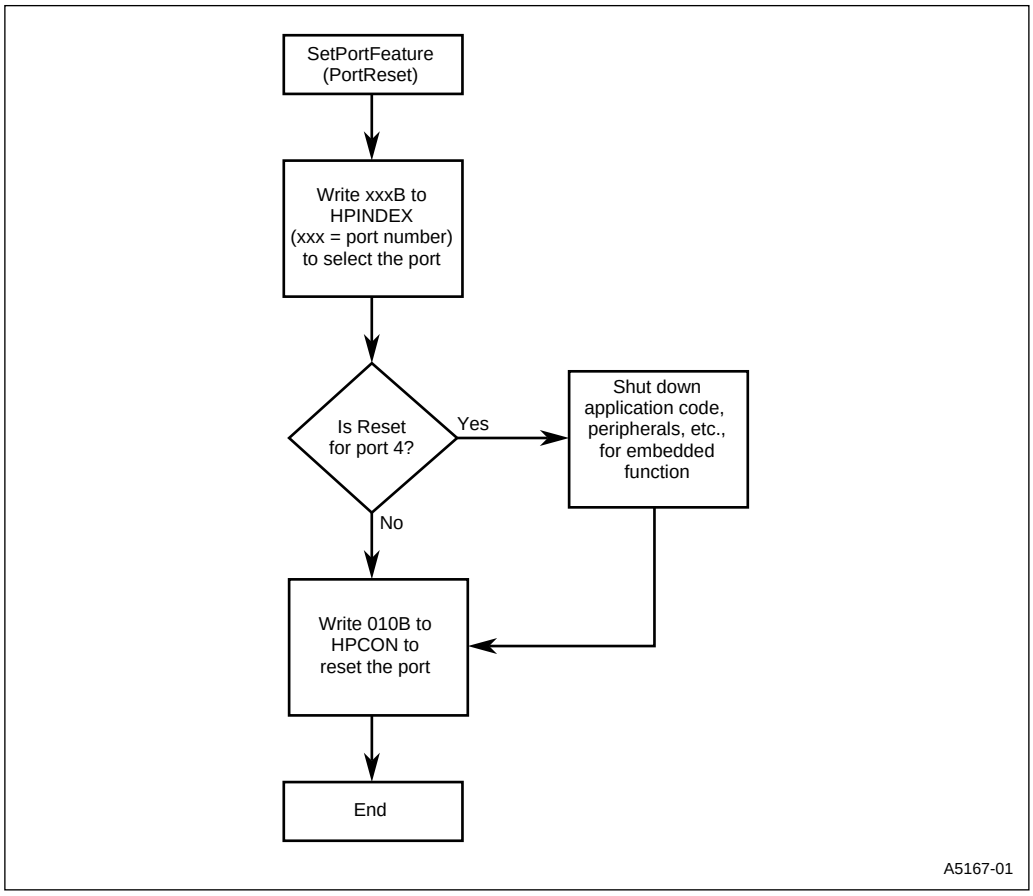


Figure 8-15. SetPortFeature (PORT_RESET) Routine

If port 4 is specified, firmware must reset all non-hub functionality in the microcontroller. Firmware must gracefully shut-down the application code, peripherals, etc. prior to writing to port 4's HPCON.

Upon writing to port 4's HPCON SFR, a hardware reset is applied to the FIU and function FIFOs. When this reset is applied, the embedded function's EPCON, FIFLG, FIE, TXSTAT, RXSTAT, TXCON, RXCON, FADDR, and PCON1 SFRs are reset to their default values, as are the SOFACK, ASOF, SOFIE, and SOFODIS bits of SOFH. The EPINDEX and SOFL SFRs remain unchanged. These SFRs are reset immediately after the write to HPCON, however bus traffic to the embedded function remains inactive for 15 ms. You may use this time frame to initialize the embedded function.

9

Input/Output Ports

CHAPTER 9
INPUT/OUTPUT PORTS

The 8*x*931 has four 8-bit input/output (I/O) ports for general-purpose I/O, external memory operations, and specific alternate functions (see Table 9-1). This chapter describes the ports and provides information on port loading, read-modify-write instructions, and external memory accesses. Chapter 16, “External Memory Interface,” contains additional information about external memory operations.

9.1 INPUT/OUTPUT PORT OVERVIEW

All four 8*x*931 I/O ports are bidirectional. Each port contains a latch, an output driver, and an input buffer. Port 0 and port 2 output drivers and input buffers facilitate external memory operations. Port 0 drives the lower address byte onto the parallel address bus, and port 2 drives the upper address byte onto the bus. The data is multiplexed with the lower address byte on port 0. Port 1 and port 3 provide both general-purpose I/O and special alternate functions.

Table 9-1. Input/Output Port Pin Descriptions

Pin Name	Type	Alternate Pin Name	Alternate Description	Alternate Type
P0.7:0	I/O	AD7:0/KSI7:0	Address/Data Lines, Keyboard Scan Input	I/O
P1.0	I/O	T2/KSO0	Timer 2 Clock Input/Output, Keyboard Scan Output	I/O
P1.1	I/O	T2EX/KSO1	Timer 2 External Input, Keyboard Scan Output	I/O
P1.5:2	I/O	KSO5:2	Keyboard Scan Output	O
P1.6	I/O	RXD/KSO6	Receive Serial Data, Keyboard Scan Output	I/O
P1.7	I/O	TXD/KSO7	Transmit Serial Data, Keyboard Scan Output	O
P2.7:0	I/O	A15:8/KSO15:8	Address Lines, Keyboard Scan Output	O
P3.0	I/O	OVRI#	Overcurrent Sense Input	I
P3.1	I/O	SOF#	Start of Frame	O
P3.2	I/O	INT0#	External Interrupt 0	I
P3.3	I/O	INT1#	External Interrupt 1	I
P3.4	I/O	T0/KSO16	Timer 0 Input/Keyboard Scan Output	I/O
P3.5	I/O	T1/KSO17	Timer 1 Input/Keyboard Scan Output	I/O
P3.6	I/O	WR#/KSO19	Write Signal to External Memory/Keyboard Scan Output	O
P3.7	I/O	RD#/KSO18	Read Signal to External Memory/Keyboard Scan Output	O

9.2 I/O CONFIGURATIONS

Each port SFR operates via type-D latches, as illustrated in Figure 9-1 for ports 1 and 3. A CPU "write to latch" signal initiates transfer of internal bus data into the type-D latch. A CPU "read latch" signal transfers the latched Q output onto the internal bus. Similarly, a "read pin" signal transfers the logical level of the port pin. Some port data instructions activate the "read latch" signal while others activate the "read pin" signal. Latch instructions are referred to as read-modify-write instructions (see "Read-Modify-Write Instructions" on page 9-5). Each I/O line may be independently programmed as input or output.

9.3 PORT 1 AND PORT 3

Figure 9-1 shows the structure of ports 1 and 3, which have internal pullups. An external source can pull the pin low. Each port pin can be configured either for general-purpose I/O or for its alternate input or output function (Table 9-1).

To use a pin for general-purpose output, set or clear the corresponding bit in the P*x* register (*x* = 1, 3). To use a pin for general-purpose input, set the bit in the P*x* register. This turns off the output driver FET.

To configure a pin for its alternate function, set the bit in the P*x* register. When the latch is set, the "alternate output function" signal controls the output level (Figure 9-1). The operation of ports 1 and 3 is discussed further in "Quasi-bidirectional Port Operation" on page 9-6.

9.4 PORT 0 AND PORT 2

Ports 0 and 2 are used for general-purpose I/O or as the external address/data bus. Port 0, shown in Figure 9-2, differs from the other ports in not having internal pullups. Figure 9-3 on page 9-4 shows the structure of port 2. An external source can pull a port 2 pin low.

To use a pin for general-purpose output, set or clear the corresponding bit in the P*x* register (*x* = 0, 2). To use a pin for general-purpose input, set the bit in the P*x* register to turn off the output driver FET.

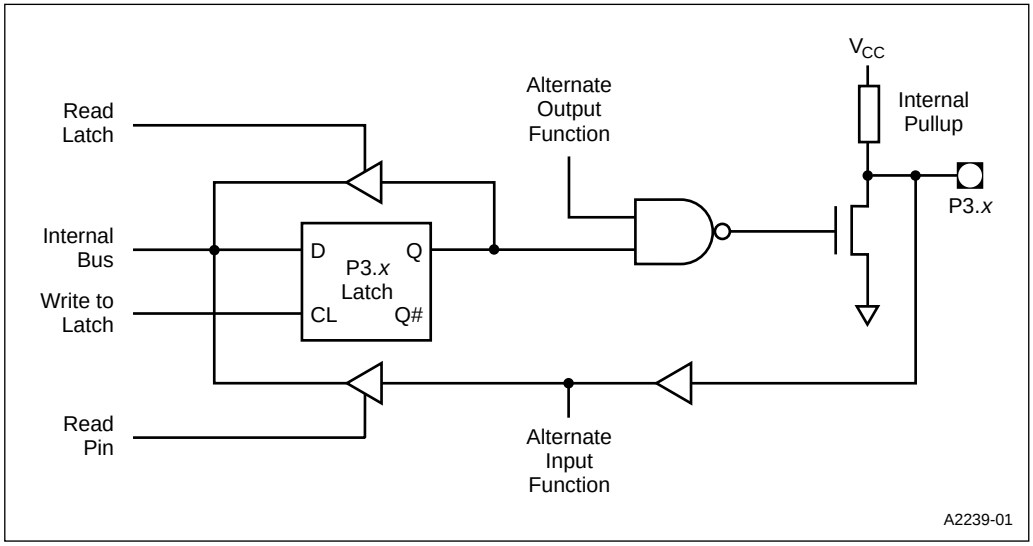


Figure 9-1. Port 1 and Port 3 Structure

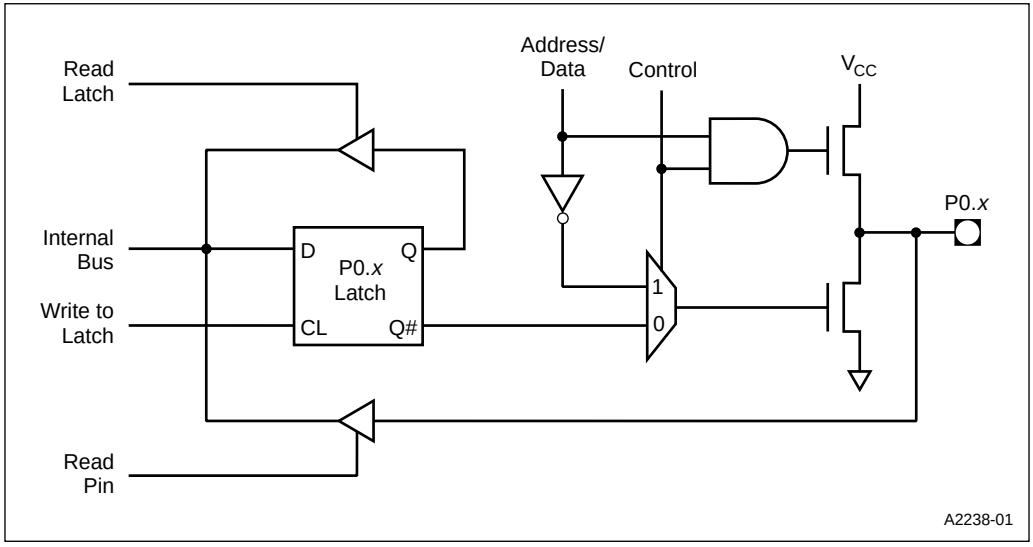


Figure 9-2. Port 0 Structure

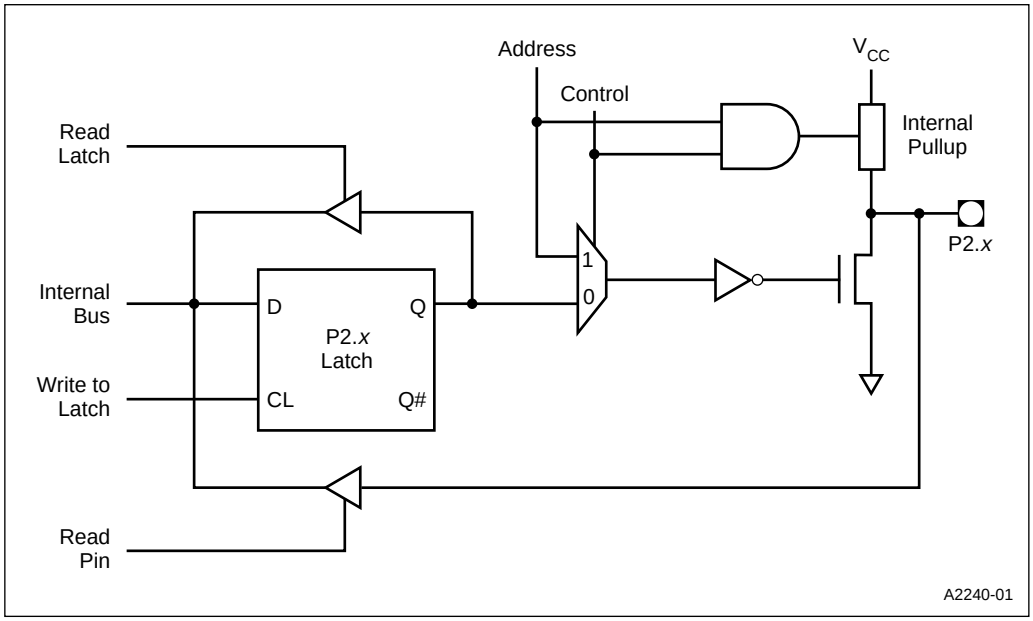


Figure 9-3. Port 2 Structure

When port 0 and port 2 are used for an external memory cycle, an internal control signal switches the output-driver input from the latch output to the internal address/data line. "External Memory Access" on page 9-7 discusses the operation of port 0 and port 2 as the external address/data bus.

NOTE

Port 0 and port 2 are precluded from use as general purpose I/O ports when used as address/data bus drivers.

Port 0 internal pullups assist the logic-one output for memory bus cycles only. Except for these bus cycles, the pullup FET is off. All other port 0 outputs are open drain.

9.5 READ-MODIFY-WRITE INSTRUCTIONS

Some instructions read the latch data rather than the pin data. The latch based instructions read the data, modify the data, and then rewrite the latch. These are called “read-modify-write” instructions. Table 9-2 contains a complete list of these special instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin.

Table 9-2. Read-Modify-Write Instructions

Instruction	Description
ANL	logical AND, e.g., ANL P1, A
ORL	logical OR, e.g., ORL P2, A
XRL	logical EX-OR, e.g., XRL P3, A
JBC	jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL
CPL	complement bit, e.g., CPL P3.0
INC	increment, e.g., INC P2
DEC	decrement, e.g., DEC P2
DJNZ	decrement and jump if not zero, e.g., DJNZ P3, LABEL
MOV PX.Y, C	move carry bit to bit Y of port X
CLR PX.Y	clear bit Y of port X
SETB PX.Y	set bit Y of port x

It is not obvious that the last three instructions in Table 9-2 are read-modify-write instructions. These instructions read the port (all eight bits), modify the specifically addressed bit, and write the new byte back to the latch. These read-modify-write instructions are directed to the latch rather than the pin in order to avoid possible misinterpretation of voltage (and therefore, logic) levels at the pin. For example, a port bit used to drive the base of an external bipolar transistor cannot rise above the transistor’s base-emitter junction voltage (a value lower than V_{IL}). With a logic one written to the bit, attempts by the CPU to read the port at the pin are misinterpreted as logic zero. A read of the latch rather than the pin returns the correct logic-one value.

9.6 QUASI-BIDIRECTIONAL PORT OPERATION

Port 1, port 2, and port 3 have fixed internal pullups and are referred to as "quasi-bidirectional" ports. When configured as an input, the pin impedance appears as logic one and sources current (see the 8*x*931 datasheet) in response to an external logic-zero condition. Port 0 is a "true bidirectional" pin. The pin floats when configured as input. Resets write logical one to all port latches. If logical zero is subsequently written to a port latch, it can be returned to input conditions by a logical one written to the latch. For additional electrical information, refer to the current 8*x*931 datasheet.

NOTE

Port latch values change near the end of read-modify-write instruction cycles. Output buffers (and therefore the pin state) update early in the instruction after the read-modify-write instruction cycle.

Logical zero-to-one transitions in port 1, port 2, and port 3 utilize an additional pullup to aid this logic transition (see Figure 9-4). This increases switch speed. The extra pullup briefly sources 100 times the normal internal circuit current. The internal pullups are field-effect transistors rather than linear resistors. Pullups consist of three p-channel FET (pFET) devices. A pFET is on when the gate senses logical zero and off when the gate senses logical one. pFET #1 is turned on for two oscillator periods immediately after a zero-to-one transition in the port latch. A logic one at the port pin turns on pFET #3 (a weak pullup) through the inverter. This inverter and pFET pair form a latch to drive logic one. pFET #2 is a very weak pullup switched on whenever the associated nFET is switched off. This is a traditional CMOS switch convention. Current strengths are 1/10 that of pFET #3.

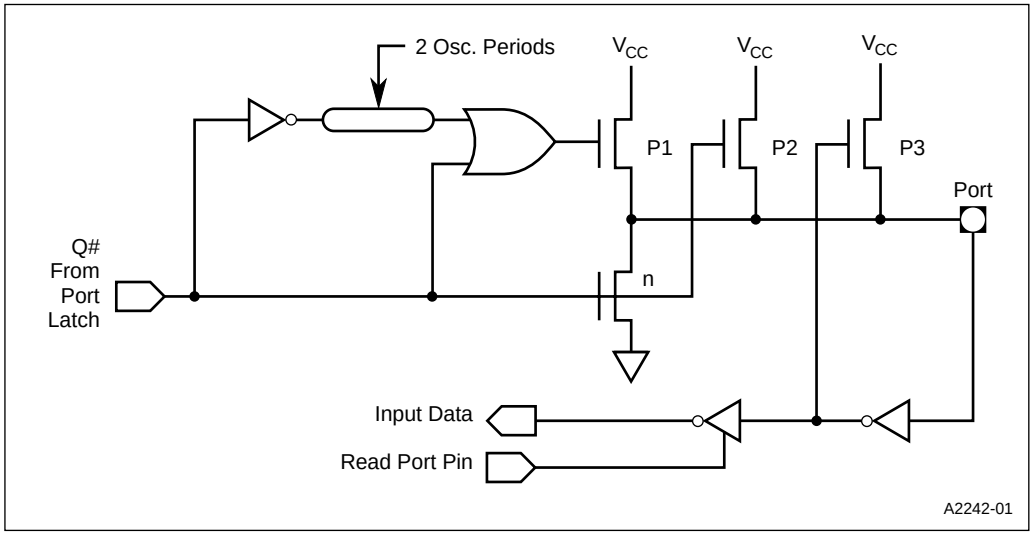


Figure 9-4. Internal Pullup Configurations

9.7 PORT LOADING

Output buffers of port 1, port 2, and port 3 can each sink 1.6 mA at logic zero (see V_{OL} specifications in the 8*x*931 data sheet). These port pins can be driven by open-collector and open-drain devices. Logic zero-to-one transitions occur slowly as limited current pulls the pin to a logic-one condition (Figure 9-4 on page 9-6). A logic-zero input turns off pFET #3. This leaves only pFET #2 weakly in support of the transition. In external bus mode, port 0 output buffers each sink 3.2 mA at logic zero (see V_{OL1} in the 8*x*931 data sheet). However, the port 0 pins require external pullups to drive external gate inputs. See the latest revision of the 8*x*931 datasheet for complete electrical design information. External circuits must be designed to limit current requirements to these conditions.

9.8 EXTERNAL MEMORY ACCESS

Port 2 outputs the upper address byte; the lower address byte and the data are multiplexed on port 0. Port 0 uses a strong internal pullup FET to output ones or a strong internal pulldown FET to output zeros for the lower address byte and the data. Port 0 is in a high-impedance state for data input. Port 2 uses a strong internal pullup FET to output ones or a strong internal pulldown FET to output zeros for the upper address byte.

There are two types of external memory accesses: external program memory and external data memory (see Chapter 15, “External Memory Interface”). External program memories utilize signal PSEN# as a read strobe. Accesses to external data memory use RD# (read) or WR# (write) to strobe the memory.

During instruction fetches, external program memory transfer instructions with 16-bit addresses.

External data memory transfers use an 8-bit or 16-bit address bus, depending on the instruction. Table 9-3 lists the instructions that can be used for these bus widths.

Table 9-3. Instructions for External Data Moves

Bus Width	Instructions
8	MOVX @Ri, A; MOVX A, @Ri
16	MOVX @DPTR, A; MOVX A, @DPTR

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. The Port 2 drivers use the strong pullups during the entire time that they are emitting address bits that are 1s. This occurs when the MOVX @ DPTR instruction is executed. During this time, the Port 2 latch (the special function register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used (MOVX @ Ri), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. In this case, Port 2 pins can be used to page the external data memory.

In either case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDRESS/DATA signal drives both FETs in the Port 0 output buffers. Thus, in external bus mode the Port 0 pins are not open-drain outputs and do not require external pullups.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the special function register), thus obliterating the information in the Port 0 SFR.

NOTE

Avoid MOV P0 instructions for external memory accesses. These instructions can corrupt input code bytes at port 0.

External signal ALE (address latch enable) facilitates external address latch capture. The address byte is valid after the ALE pin drives V_{OL}. For write cycles, valid data is written to port 0 just prior to the write (WR#) pin asserting V_{OL}. Data remains valid until WR# is deactivated. For read cycles, data returned from external memory must appear at port 0 before the read (RD#) pin is deactivated (refer to the 8*x*931 datasheet for specifications).

10

Timer/Counters

CHAPTER 10
TIMER/COUNTERS

This chapter describes the timer/counter peripherals on the 8*x*931. When operating as a timer, a timer/counter runs for a programmed length of time, then issues an interrupt request. When operating as a counter, a timer/counter counts negative transitions on an external pin. After a preset number of counts, the counter issues an interrupt request.

10.1 TIMER/COUNTER OVERVIEW

The 8*x*931 contains three general-purpose, 16-bit timer/counters. Although they are identified as timer 0, timer 1, and timer 2, you can independently configure each to operate in a variety of modes as a timer or as an event counter. Each timer employs two 8-bit timer registers, used separately or in cascade, to maintain the count. The timer registers and associated control and capture registers are implemented as addressable special function registers (SFRs). Four of the SFRs provide programmable control of the timers as follows:

- Timer/counter mode control register (TMOD) and timer/counter control register (TCON) control timer 0 and timer 1
- Timer/counter 2 mode control register (T2MOD) and timer/counter 2 control register (T2CON) control timer 2

Table 10-1 describes the external signals referred to in this chapter. Table 10-2 briefly describes the SFRs referred to in this chapter. For a map of the SFR address space, see Table C-1 on page C-2.

10.2 TIMER/COUNTER OPERATION

The block diagram in Figure 10-1 depicts the basic logic of the timers. Here timer registers TH*x* and TL*x* (*x* = 0, 1, and 2) connect in cascade to form a 16-bit timer. Setting the run control bit (TR*x*) turns the timer on by allowing the selected input to increment TL*x.* When TL*x* overflows it increments TH*x;* when TH*x* overflows it sets the timer overflow flag (TF*x*) in the TCON or T2CON register. Setting the run control bit does not clear the TH*x* and TL*x* timer registers. The timer registers can be accessed to obtain the current count or to enter preset values. Timer 0 and timer 1 can also be controlled by external pin INT*x*# to facilitate pulse width measurements.

The C\T*x*# control bit selects timer operation or counter operation by selecting the divided-down system clock or external pin T*x* as the source for the counted signal.

For timer operation (C/T*x*# = 0), the timer register counts the divided-down system clock. The timer register is incremented once every peripheral cycle (once every six states). That is, at the internal clock frequency divided by six ($F_{CLK}/6$) or at the external oscillator frequency ($F_{OSC}/12$). Exceptions are the timer 2 clock-out and baud rate modes, in which the timer register is incremented at the internal clock rate (F_{CLK}). See “Clock and Reset Unit” on page 2-7. Table 2-5 on page 2-12 and Figure 2-5 on page 2-10 show the relationship between F_{OSC}, F_{CLK}, state times, and peripheral cycles. Also see the 8*x*931 clock circuit block diagram in Figure 2-4 on page 2-9.

NOTE

The timing calculations in this chapter are based on the value of F_{CLK} ($F_{OSC}/2$). Setting the low clock (PCON.5) bit forces the internal clock (F_{CLK}) distributed to the CPU and peripherals to 3MHz. This bit is automatically set after a reset. Clearing this bit through firmware returns F_{CLK} to the normal clock frequency ($F_{OSC}/2$).

For counter operation (C/T*x*# = 1), the timer register counts the negative transitions on the T*x* external input pin. The external input is sampled during every S5P2 state. "Clock and Reset Unit" on page 2-9 describes the notation for the states in a peripheral cycle. When the sample is high in one cycle and low in the next, the counter is incremented. The new count value appears in the register during the next S3P1 state after the transition was detected. Since it takes two peripheral cycles to recognize a negative transition, the maximum count rate is $F_{CLK}/12$. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full peripheral cycle.

Table 10-1. External Signals

Signal Name	Type	Description	Alternate Function
T2	I/O	**Timer 2 Clock Input/Output**. This signal is the external clock input for the timer 2 capture mode; and it is the timer 2 clock-output for the clock-out mode.	P1.0
T2EX	I	**Timer 2 External Input**. In timer 2 capture mode, a falling edge initiates a capture of the timer 2 registers. In auto-reload mode, a falling edge causes the timer 2 registers to be reloaded. In the up-down counter mode, this signal determines the count direction: high = up, low = down.	P1.1
INT1:0#	I	**External Interrupts 1:0**. These inputs set the IE1:0 interrupt flags in the TCON register. TCON bits IT1:0 select the triggering method: IT1:0 = 1 selects edge-triggered (high-to-low);IT1:0 = 0 selects level-triggered (active low). INT1:0# also serves as external run control for timer 1:0 when selected by TCON bits GATE1:0#.	P3.3:2
T1:0	I	**Timer 1:0 External Clock Inputs**. When timer 1:0 operates as a counter, a falling edge on the T1:0 pin increments the count.	P3.5:4

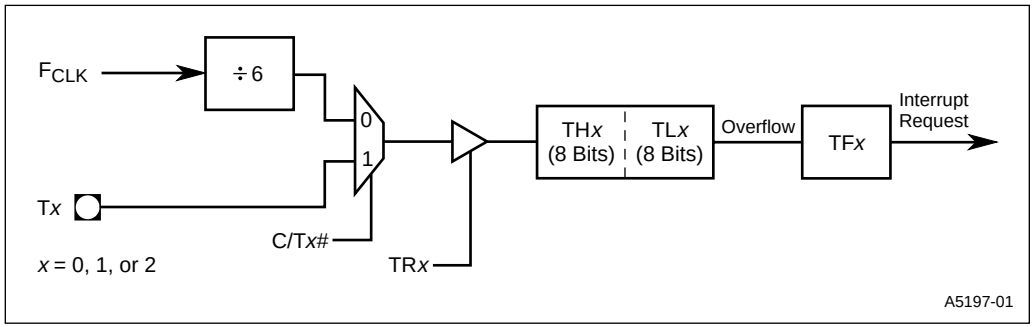


Figure 10-1. Basic Logic of the Timer/Counters

Table 10-2. Timer/Counter and Watchdog Timer SFRs

Mnemonic	Description	Address
TL0 TH0	**Timer 0 Timer Registers.** Used separately as 8-bit counters or in cascade as a 16-bit counter. Counts an internal clock signal with frequency $F_{CLK}/6$ (timer operation) or an external input (event counter operation).	8AH 8CH
TL1 TH1	**Timer 1 Timer Registers.** Used separately as 8-bit counters or in cascade as a 16-bit counter. Counts an internal clock signal with frequency $F_{CLK}/6$ (timer operation) or an external input (event counter operation).	8BH 8DH
TL2 TH2	**Timer 2 Timer Registers.** TL2 and TH2 connect in cascade to provide a 16-bit counter. Counts an internal clock signal with frequency $F_{CLK}/6$ (timer operation) or an external input (event counter operation).	CCH CDH
TCON	**Timer 0/1 Control Register.** Contains the run control bits, overflow flags, interrupt flags, and interrupt-type control bits for timer 0 and timer 1.	88H
TMOD	**Timer 0/1 Mode Control Register.** Contains the mode select bits, counter/timer select bits, and external control gate bits for timer 0 and timer 1.	89H
T2CON	**Timer 2 Control Register.** Contains the receive clock, transmit clock, and capture/reload bits used to configure timer 2. Also contains the run control bit, counter/timer select bit, overflow flag, external flag, and external enable for timer 2.	C8H
T2MOD	**Timer 2 Mode Control Register.** Contains the timer 2 output enable and down count enable bits.	C9H
RCAP2L RCAP2H	**Timer 2 Reload/Capture Registers (RCAP2L, RCAP2H).** Provide values to and receive values from the timer registers (TL2,TH2).	CAH CBH

10.3 TIMER 0

Timer 0 functions as either a timer or event counter in four modes of operation. Figures 10-2, 10-3, and 10-4 show the logical configuration of each mode.

Timer 0 is controlled by the four low-order bits of the TMOD register (Figure 10-5) and bits 5, 4, 1, and 0 of the TCON register (Figure 10-6). The TMOD register selects the method of timer gating (GATE0), timer or counter operation (T/C0#), and mode of operation (M10 and M00). The TCON register provides timer 0 control functions: overflow flag (TF0), run control (TR0), interrupt flag (IE0), and interrupt type control (IT0).

For normal timer operation (GATE0 = 0), setting TR0 allows TL0 to be incremented by the selected input. Setting GATE0 and TR0 allows external pin INT0# to control timer operation. This setup can be used to make pulse width measurements. See "Pulse Width Measurements" on page 10-10.

Timer 0 overflow (count rolls over from all 1s to all 0s) sets the TF0 flag generating an interrupt request.

10.3.1 Mode 0 (13-bit Timer)

Mode 0 configures timer 0 as a 13-bit timer which is set up as an 8-bit timer (TH0 register) with a modulo 32 prescalar implemented with the lower five bits of the TL0 register (Figure 10-2). The upper three bits of the TL0 register are indeterminate and should be ignored. Prescalar overflow increments the TH0 register.

10.3.2 Mode 1 (16-bit Timer)

Mode 1 configures timer 0 as a 16-bit timer with TH0 and TL0 connected in cascade (Figure 10-2). The selected input increments TL0.

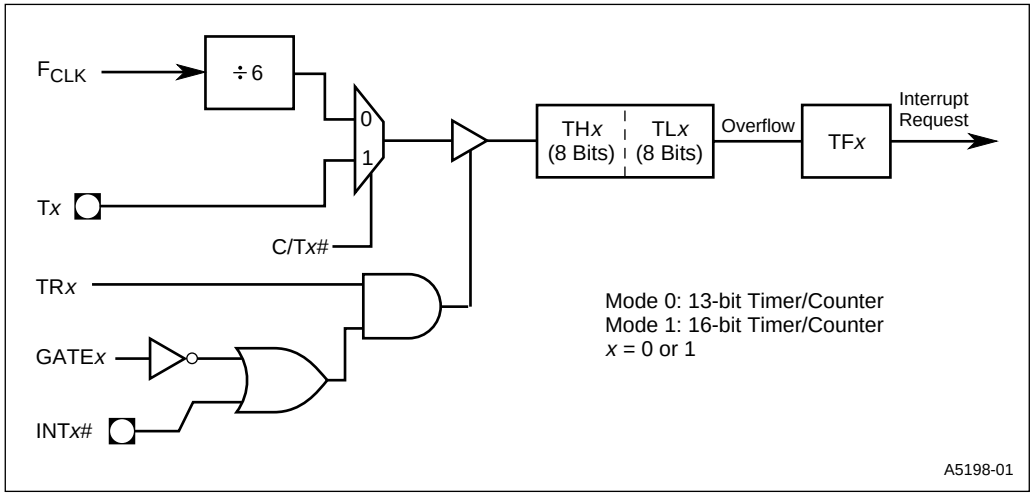


Figure 10-2. Timer 0/1 in Mode 0 and Mode 1

10.3.3 Mode 2 (8-bit Timer With Auto-reload)

Mode 2 configures timer 0 as an 8-bit timer (TL0 register) that automatically reloads from the TH0 register (Figure 10-3). TL0 overflow sets the timer overflow flag (TF0) in the TCON register and reloads TL0 with the contents of TH0, which is preset by firmware. When the interrupt request is serviced, hardware clears TF0. The reload leaves TH0 unchanged. See “Auto-reload Set-up Example” on page 10-9.

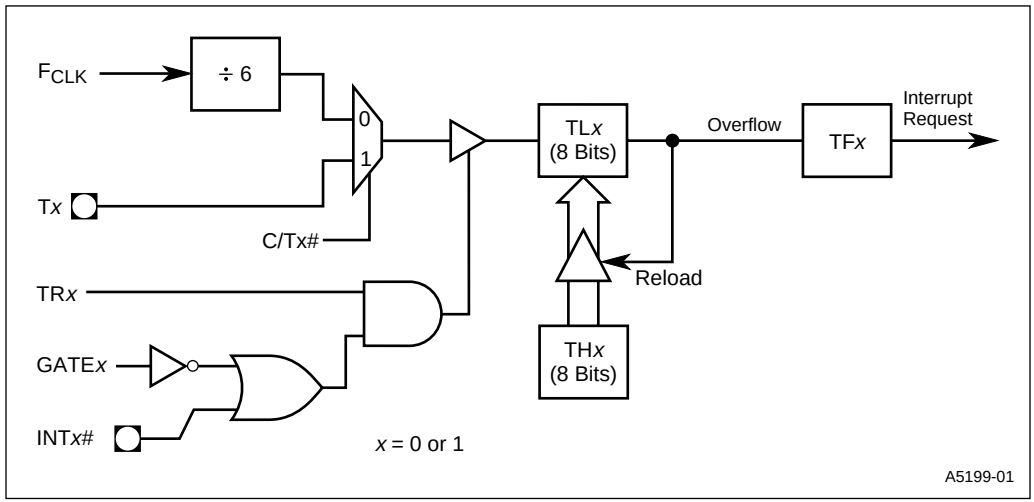


Figure 10-3. Timer 0/1 in Mode 2, Auto-reload

10.3.4 Mode 3 (Two 8-bit Timers)

Mode 3 configures timer 0 such that registers TL0 and TH0 operate as separate 8-bit timers (Figure 10-4). This mode is provided for applications requiring an additional 8-bit timer or counter. TL0 uses the timer 0 control bits C/T0# and GATE0 in TMOD, and TR0 and TF0 in TCON in the normal manner. TH0 is locked into a timer function (counting F_{CLK} /6) and takes over use of the timer 1 interrupt (TF1) and run control (TR1) bits. Thus, operation of timer 1 is restricted when timer 0 is in mode 3. See the last paragraph of “Timer 1” on page 10-6 and “Mode 3 (Halt)” on page 10-9.

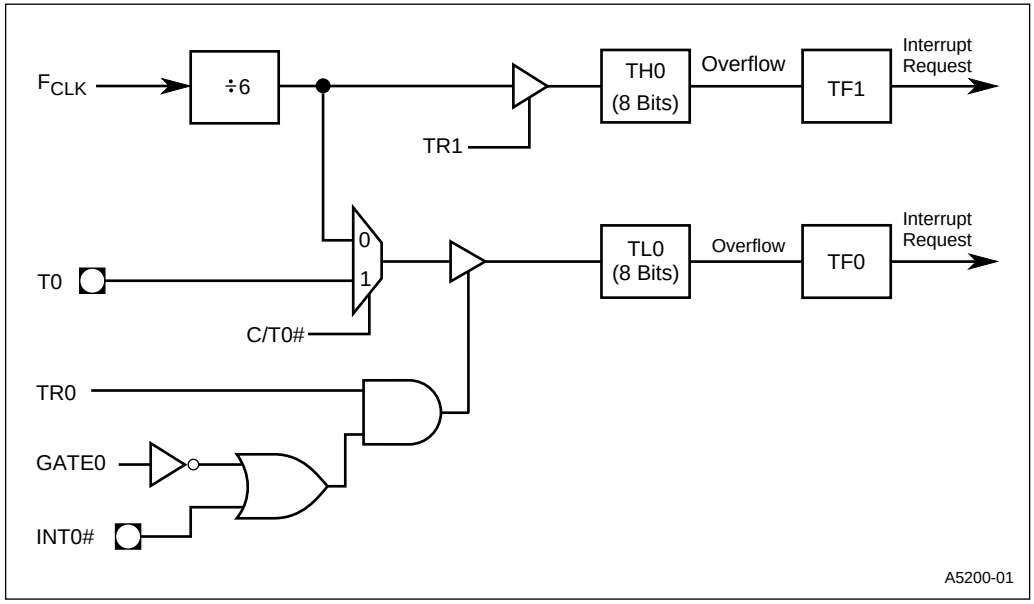


Figure 10-4. Timer 0 in Mode 3, Two 8-bit Timers

10.4 TIMER 1

Timer 1 functions as either a timer or event counter in three modes of operation. Figures 10-2 and 10-3 show the logical configuration for modes 0, 1, and 2. Timer 1's mode 3 is a hold-count mode.

Timer 1 is controlled by the four high-order bits of the TMOD register (Figure 10-5) and bits 7, 6, 3, and 2 of the TCON register (Figure 10-6). The TMOD register selects the method of timer gating (GATE1), timer or counter operation (T/C1#), and mode of operation (M11 and M01). The TCON register provides timer 1 control functions: overflow flag (TF1), run control (TR1), interrupt flag (IE1), and interrupt type control (IT1).

Timer 1 operation in modes 0, 1, and 2 is identical to timer 0. Timer 1 can serve as the baud rate generator for the serial port. Mode 2 is best suited for this purpose.

For normal timer operation (GATE1 = 0), setting TR1 allows timer register TL1 to be incremented by the selected input. Setting GATE1 and TR1 allows external pin INT1# to control timer operation. This setup can be used to make pulse width measurements. See "Pulse Width Measurements" on page 10-10.

Timer 1 overflow (count rolls over from all 1s to all 0s) sets the TF1 flag, generating an interrupt request.

When timer 0 is in mode 3, it uses timer 1's overflow flag (TF1) and run control bit (TR1). For this situation, use timer 1 only for applications that do not require an interrupt (such as a baud rate generator for the serial interface port) and switch timer 1 in and out of mode 3 to turn it off and on.

TMOD Address: S:89H
Reset State: 0000 0000B

Timer/Counter Mode Control Register. Contains mode select, run control select, and counter/timer select bits for controlling timer 0 and timer 1.

7							0
GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

Bit Number	Bit Mnemonic	Function
7	GATE1	Timer 1 Gate: When GATE1 = 0, run control bit TR1 gates the input signal to the timer register. When GATE1 = 1 and TR1 = 1, external signal INT1 gates the timer input.
6	C/T1#	Timer 1 Counter/Timer Select: C/T1# = 0 selects timer operation: timer 1 counts the divided-down system clock. C/T1# = 1 selects counter operation: timer 1 counts negative transitions on external pin T1.
5, 4	M11, M01	Timer 1 Mode Select: **M11 M01** 0 0 Mode 0: 8-bit timer/counter (TH1) with 5-bit prescalar (TL1) 0 1 Mode 1: 16-bit timer/counter 1 0 Mode 2: 8-bit auto-reload timer/counter (TL1). Reloaded from TH1 at overflow. 1 1 Mode 3: Timer 1 halted. Retains count.
3	GATE0	Timer 0 Gate: When GATE0 = 0, run control bit TR0 gates the input signal to the timer register. When GATE0 = 1 and TR0 = 1, external signal INT0 gates the timer input.
2	C/T0#	Timer 0 Counter/Timer Select: C/T0# = 0 selects timer operation: timer 0 counts the divided-down system clock. C/T0# = 1 selects counter operation: timer 0 counts negative transitions on external pin T0.
1, 0	M10, M00	Timer 0 Mode Select: **M10 M00** 0 0 Mode 0: 8-bit timer/counter (T0) with 5-bit prescalar (TL0) 0 1 Mode 1: 16-bit timer/counter 1 0 Mode 2: 8-bit auto-reload timer/counter (TL0). Reloaded from TH0 at overflow. 1 1 Mode 3: TL0 is an 8-bit timer/counter. TH0 is an 8-bit timer using timer 1's TR1 and TF1 bits

Figure 10-5. TMOD: Timer/Counter Mode Control Register

TCON

Address: S:88H
Reset State: 0000 0000B

Timer/Counter Control Register. Contains the overflow and external interrupt flags and the run control and interrupt transition select bits for timer 0 and timer 1.

7							0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit Number	Bit Mnemonic	Function
7	TF1	Timer 1 Overflow Flag: Set by hardware when the timer 1 register overflows. Cleared by hardware when the processor vectors to the interrupt routine.
6	TR1	Timer 1 Run Control Bit: Set/cleared by firmware to turn timer 1 on/off.
5	TF0	Timer 0 Overflow Flag: Set by hardware when the timer 0 register overflows. Cleared by hardware when the processor vectors to the interrupt routine.
4	TR0	Timer 0 Run Control Bit: Set/cleared by firmware to turn timer 0 on/off.
3	IE1	Interrupt 1 Flag: Set by hardware when an external interrupt is detected on the INT1# pin. Edge- or level- triggered (see IT1). Cleared when interrupt is processed if edge-triggered.
2	IT1	Interrupt 1 Type Control Bit: Set this bit to select edge-triggered (high-to-low) for external interrupt 1. Clear this bit to select level-triggered (active low).
1	IE0	Interrupt 0 Flag: Set by hardware when an external interrupt is detected on the INT0# pin. Edge- or level- triggered (see IT0). Cleared when interrupt is processed if edge-triggered.
0	IT0	Interrupt 0 Type Control Bit: Set this bit to select edge-triggered (high-to-low) for external interrupt 0. Clear this bit to select level-triggered (active low).

Figure 10-6. TCON: Timer/Counter Control Register

10.4.1 Mode 0 (13-bit Timer)

Mode 0 configures timer 0 as a 13-bit timer, which is set up as an 8-bit timer (TH1 register) with a modulo-32 prescalar implemented with the lower five bits of the TL1 register (Figure 10-2). The upper three bits of the TL1 register are ignored. Prescalar overflow increments the TH1 register.

10.4.2 Mode 1 (16-bit Timer)

Mode 1 configures timer 1 as a 16-bit timer with TH1 and TL1 connected in cascade (Figure 10-2). The selected input increments TL1.

10.4.3 Mode 2 (8-bit Timer with Auto-reload)

Mode 2 configures timer 1 as an 8-bit timer (TL1 register) with automatic reload from the TH1 register on overflow (Figure 10-3). Overflow from TL1 sets overflow flag TF1 in the TCON register and reloads TL1 with the contents of TH1, which is preset by firmware. The reload leaves TH1 unchanged. See “Auto-reload Setup Example” on page 10-9.

10.4.4 Mode 3 (Halt)

Placing timer 1 in mode 3 causes it to halt and hold its count. This can be used to halt timer 1 when the TR1 run control bit is not available (i.e., when timer 0 is in mode 3). See the final paragraph of “Timer 1” on page 10-6.

10.5 TIMER 0/1 APPLICATIONS

Timer 0 and timer 1 are general purpose timers that can be used in a variety of ways. The timer applications presented in this section are intended to demonstrate timer setup, and do not represent the only arrangement nor necessarily the best arrangement for a given task. These examples employ timer 0, but timer 1 can be set up in the same manner using the appropriate registers.

10.5.1 Auto-reload Setup Example

Timer 0 can be configured as an eight-bit timer (TL0) with automatic reload as follows:

1. Program the four low-order bits of the TMOD register (Figure 10-5) to specify: mode 2 for timer 0, C/T0# = 0 to select $F_{CLK}/6$ as the timer input, and GATE0 = 0 to select TR0 as the timer run control.
2. Enter an eight-bit initial value (n_0) in timer register TL0, so that the timer overflows after the desired number of peripheral cycles.
3. Enter an eight-bit reload value (n_R) in register TH0. This can be the same as n_0 or different, depending on the application.
4. Set the TR0 bit in the TCON register (Figure 10-6) to start the timer. Timer overflow occurs after FFH + 1 - n_0 peripheral cycles, setting the TF0 flag and loading n_R into TL0 from TH0. When the interrupt is serviced, hardware clears TF0.

5. The timer continues to overflow and generate interrupt requests every FFH + 1 - n_R peripheral cycles.
6. To halt the timer, clear the TR0 bit.

10.5.2 Pulse Width Measurements

For timer 0 and timer 1, setting GATE*x* and TR*x* allows an external waveform at pin INT*x*# to turn the timer on and off. This setup can be used to measure the width of a positive-going pulse present at pin INT*x*#. Pulse width measurements using timer 0 in mode 1 can be made as follows:

1. Program the four low-order bits of the TMOD register (Figure 10-5) to specify: mode 1 for timer 0, C/T0# = 0 to select $F_{CLK}/6$ as the timer input, and GATE0 = 1 to select INT0 as timer run control.
2. Enter an initial value of all zeros in the 16-bit timer register TH0/TL0, or read and store the current contents of the register.
3. Set the TR0 bit in the TCON register (Figure 10-6) to enable INT0.
4. Apply the pulse to be measured to pin INT0. The timer runs when the pulse waveform is high.
5. Clear the TR0 bit to disable INT0.
6. Read timer register TH0/TL0 to obtain the new value.
7. Calculate pulse width = $6T_{CLK}$ × (new value - initial value).
8. Example: F_{OSC} = 12 MHz, F_{CLK} = 6 MHz, T_{CLK} = 0.16667 µs. If the new value = $10{,}000_{10}$ counts and the initial value = 0, the pulse width = 6(0.16667) x (10,000 – 0) = 1 µs × 10,000 = 10 ms.

10.6 TIMER 2

Timer 2 is a 16-bit timer/counter. The count is maintained by two 8-bit timer registers, TH2 and TL2, connected in cascade. The timer/counter 2 mode control register (T2MOD) as shown in Figure 10-11 on page 10-16) and the timer/counter 2 control register (T2CON) as shown in Figure 10-12 on page 10-17) control the operation of timer 2.

Timer 2 provides the following operating modes: capture mode, auto-reload mode, baud rate generator mode, and programmable clock-out mode. Select the operating mode with T2MOD and TCON register bits as shown in Table 10-3 on page 10-15. Auto-reload is the default mode. Setting RCLK and/or TCLK selects the baud rate generator mode.

Timer 2 operation is similar to timer 0 and timer 1. C/T2# selects the divided-down system clock (timer operation) or external pin T2 (counter operation) as the timer register input. Setting TF2 allows TL2 to be incremented by the selected input.

The operating modes are described in the following paragraphs. Block diagrams in Figures 10-7 through 10-10 show the timer 2 configuration for each mode.

10.6.1 Capture Mode

In the capture mode, timer 2 functions as a 16-bit timer or counter (Figure 10-7). An overflow condition sets bit TF2, which you can use to request an interrupt. Setting the external enable bit EXEN2 allows the RCAP2H and RCAP2L registers to capture the current value in timer registers TH2 and TL2 in response to a 1-to-0 transition at external input T2EX. The transition at T2EX also sets bit EXF2 in T2CON. The EXF2 bit, like TF2, can generate an interrupt.

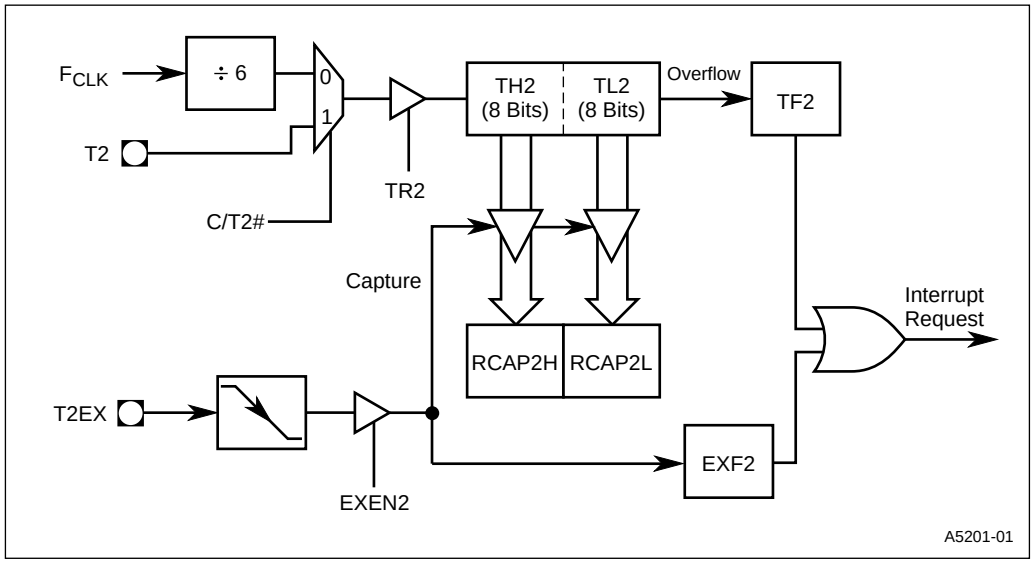


Figure 10-7. Timer 2: Capture Mode

10.6.2 Auto-reload Mode

The auto-reload mode configures timer 2 as a 16-bit timer or event counter with automatic reload. The timer operates an as an up counter or as an up/down counter, as determined by the down counter enable bit (DCEN). At device reset, DCEN is cleared, so in the auto-reload mode, timer 2 defaults to operation as an up counter.

10.6.2.1 Up Counter Operation

When DCEN = 0, timer 2 operates as an up counter (Figure 10-8). The external enable bit EXEN2 in the T2CON register provides two options (Figure 10-12). If EXEN2 = 0, timer 2 counts up to FFFFH and sets the TF2 overflow flag. The overflow condition loads the 16-bit value in the reload/capture registers (RCAP2H, RCAP2L) into the timer registers (TH2, TL2). The values in RCAP2H and RCAP2L are preset by firmware.

If EXEN2 = 1, the timer registers are reloaded by either a timer overflow or a high-to-low transition at external input T2EX. This transition also sets the EXF2 bit in the T2CON register. Either TF2 or EXF2 bit can generate a timer 2 interrupt request.

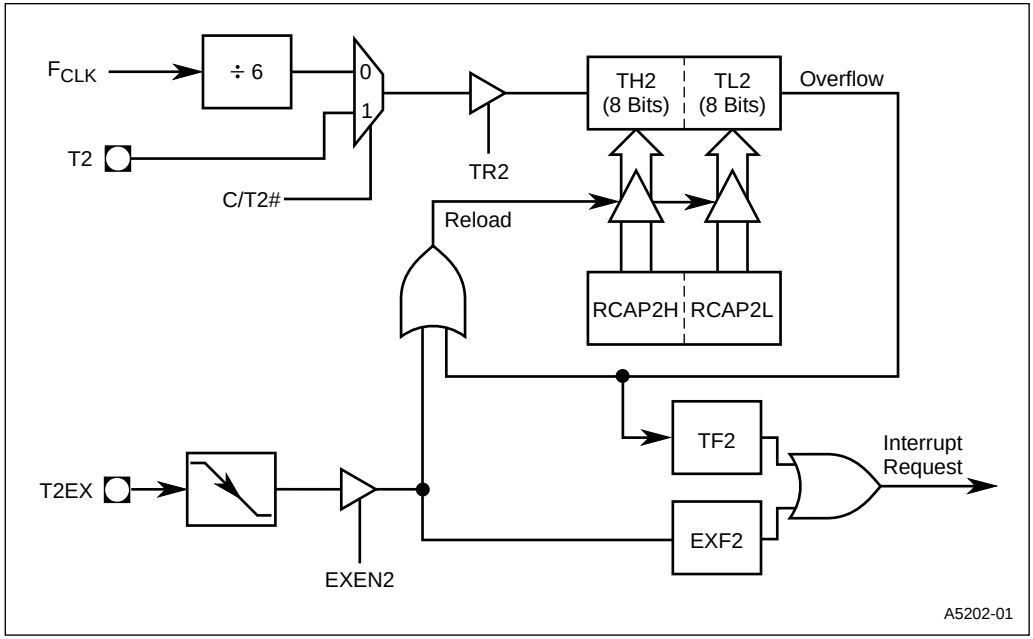


Figure 10-8. Timer 2: Auto-reload Mode (DCEN = 0)

10.6.3 Up/Down Counter Operation

When DCEN = 1, timer 2 operates as an up/down counter (Figure 10-9). External pin T2EX controls the direction of the count (Table 10-1 on page 10-2). When T2EX is high, timer 2 counts up. The timer overflow occurs at FFFFH which sets the timer 2 overflow flag (TF2) and generates an interrupt request. The overflow also causes the 16-bit value in RCAP2H and RCAP2L to be loaded into the timer registers TH2 and TL2.

When T2EX is low, timer 2 counts down. Timer underflow occurs when the count in the timer registers (TH2, TL2) equals the value stored in RCAP2H and RCAP2L. The underflow sets the TF2 bit and reloads FFFFH into the timer registers.

The EXF2 bit toggles when timer 2 overflows or underflows, changing the direction of the count. When timer 2 operates as an up/down counter, EXF2 does not generate an interrupt. This bit can be used to provide 17-bit resolution.

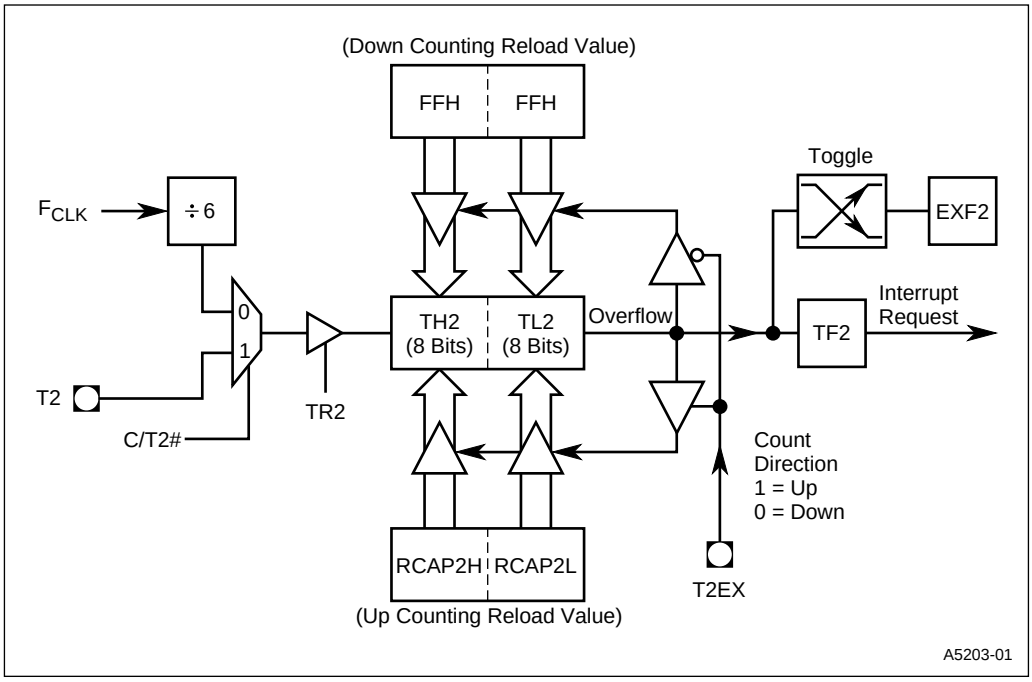


Figure 10-9. Timer 2: Auto-reload Mode (DCEN = 1)

10.6.4 Baud Rate Generator Mode

This mode configures timer 2 as a baud rate generator for use with the serial port. Select this mode by setting the RCLK and/or TCLK bits in T2CON. See Table 10-3. For details regarding this mode of operation, refer to "Baud Rates" on page 13-10.

10.6.5 Clock-out Mode

In the clock-out mode, timer 2 functions as a 50%-duty-cycle, variable-frequency clock (Figure 10-10). The generated clock signal appears on pin T2. The input clock increments TL0 at the internal clock frequency (F_{CLK}). The timer repeatedly counts to overflow from a preloaded value. At overflow, the contents of the RCAP2H and RCAP2L registers are loaded into TH2/TL2. In this mode, timer 2 overflows do not generate interrupts. The formula gives the clock-out frequency as a function of F_{CLK} (Table 2-4 on page 2-10) and the value in the RCAP2H and RCAP2L registers:

$$\text{Clock-out Frequency} = \frac{F_{CLK}}{2 \times (65535 - \text{RCAP2H, RCAP2L})}$$

For a 12MHz system clock (F_{CLK} = 6 MHz), timer 2 has a programmable frequency range of 45.8 Hz to 3 MHz.

Timer 2 is programmed for the clock-out mode as follows:

1. Set the T2OE bit in T2MOD. This gates the timer register overflow to the ÷2 counter.
2. Clear the C/T2# bit in T2CON to select F_{CLK} as the timer input signal. This also gates the output of the ÷2 counter to pin T2.
3. Determine the 16-bit reload value from the formula and enter in the RCAP2H/RCAP2L registers.
4. Enter a 16-bit initial value in timer register TH2/TL2. This can be the same as the reload value, or different, depending on the application.
5. To start the timer, set the TR2 run control bit in T2CON.

Operation is similar to timer 2 operation as a baud rate generator. It is possible to use timer 2 as a baud rate generator and a clock generator simultaneously. For this configuration, the baud rates and clock frequencies are not independent since both functions use the values in the RCAP2H and RCAP2L registers.

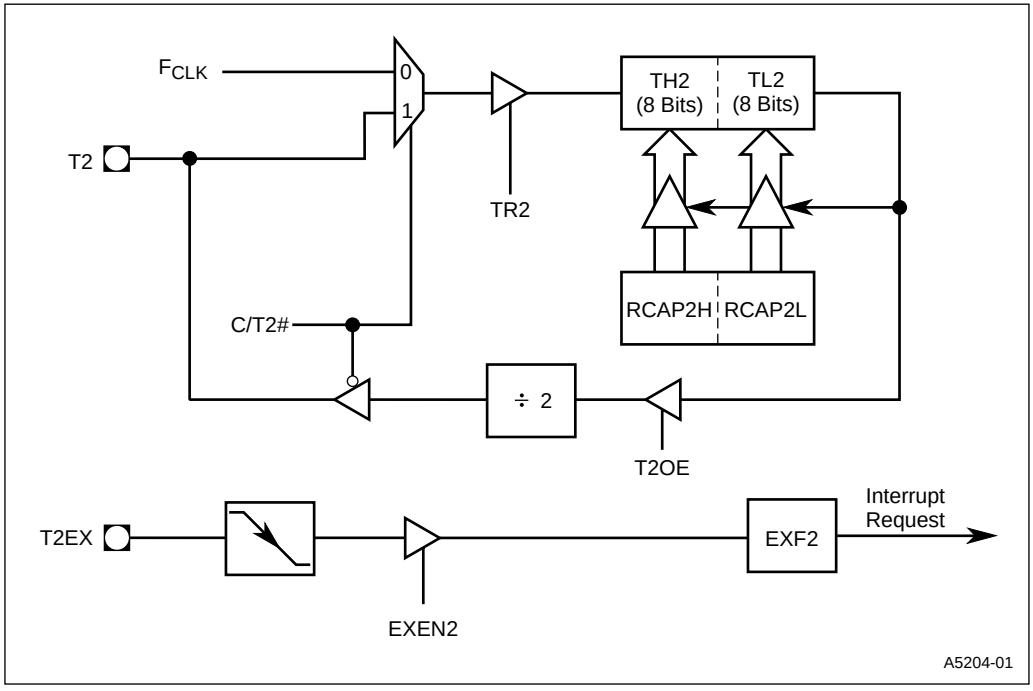


Figure 10-10. Timer 2: Clock Out Mode

Table 10-3. Timer 2 Modes of Operation

Mode	RCLK OR TCLK (in T2CON)	CP/RL2# (in T2CON)	T2OE (in T2MOD)
Auto-reload Mode	0	0	0
Capture Mode	0	1	0
Baud Rate Generator Mode	1	X	X
Programmable Clock-Out	X	0	1

T2MOD Address: S:C9H
Reset State: xxxx xx00B

Timer 2 Mode Control Register. Contains the timer 2 down count enable and clock-out enable bits for timer 2 .

7							0
—	—	—	—	—	—	T2OE	DCEN

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved: Values read from these bits are indeterminate. Write zeros to these bits.
1	T2OE	Timer 2 Output Enable Bit: In the timer 2 clock-out mode, connects the programmable clock output to external pin T2.
0	DCEN	Down Count Enable Bit: Configures timer 2 as an up/down counter.

Figure 10-11. T2MOD: Timer 2 Mode Control Register

T2CON Address: S:C8H
Reset State: 0000 0000B

Timer 2 Control Register. Contains the receive clock, transmit clock, and capture/reload bits used to configure timer 2. Also contains the run control bit, counter/timer select bit, overflow flag, external flag, and external enable for timer 2.

7 **0**

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2#	CP/RL2#

Bit Number	Bit Mnemonic	Function
7	TF2	Timer 2 Overflow Flag: Set by timer 2 overflow. Must be cleared by firmware. TF2 is not set if RCLK = 1 or TCLK = 1.
6	EXF2	Timer 2 External Flag: If EXEN2 = 1, capture or reload caused by a negative transition on T2EX sets EFX2. EXF2 does not cause an interrupt in up/down counter mode (DCEN = 1).
5	RCLK	Receive Clock Bit: Selects timer 2 overflow pulses (RCLK = 1) or timer 1 overflow pulses (RCLK = 0) as the baud rate generator for serial port modes 1 and 3.
4	TCLK	Transmit Clock Bit: Selects timer 2 overflow pulses (TCLK = 1) or timer 1 overflow pulses (TCLK = 0) as the baud rate generator for serial port modes 1 and 3.
3	EXEN2	Timer 2 External Enable Bit: Setting EXEN2 causes a capture or reload to occur as a result of a negative transition on T2EX unless timer 2 is being used as the baud rate generator for the serial port. Clearing EXEN2 causes timer 2 to ignore events at T2EX.
2	TR2	Timer 2 Run Control Bit: Setting this bit starts the timer.
1	C/T2#	Timer 2 Counter/Timer Select: C/T2# = 0 selects timer operation: timer 2 counts the divided-down system clock. C/T2# = 1 selects counter operation: timer 2 counts negative transitions on external pin T2.
0	CP/RL2#	Capture/Reload Bit: When set, captures occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto-reloads occur on timer 2 overflows or negative transitions at T2EX if EXEN2 = 1. The CP/RL2# bit is ignored and timer 2 forced to auto-reload on timer 2 overflow, if RCLK = 1 or TCLK = 1.

Figure 10-12. T2CON: Timer 2 Control Register

11

Serial I/O Port

CHAPTER 11
SERIAL I/O PORT

The serial input/output port supports communication with modems and other external peripheral devices. This chapter provides instructions for programming the serial port and generating the serial I/O baud rates with timer 1 and timer 2.

11.1 OVERVIEW

The serial I/O port provides both synchronous and asynchronous communication modes. It operates as a universal asynchronous receiver and transmitter (UART) in three full-duplex modes (modes 1, 2, and 3). Asynchronous transmission and reception can occur simultaneously and at different baud rates. The UART supports framing-bit error detection, multiprocessor communication, and automatic address recognition. The serial port also operates in a single synchronous mode (mode 0).

The synchronous mode (mode 0) operates at a single baud rate. Mode 2 operates at two baud rates. Modes 1 and 3 operate over a wide range of baud rates, which are generated by timer 1 and timer 2. Baud rates are detailed in “Baud Rates” on page 11-10.

The serial port signals are defined in Table 11-1, and the serial port special function registers are described in Table 11-2. Figure 11-1 is a block diagram of the serial port.

For the three asynchronous modes, the UART transmits on the TXD pin and receives on the RXD pin. For the synchronous mode (mode 0), the UART outputs a clock signal on the TXD pin and sends and receives messages on the RXD pin (Figure 11-1). The SBUF register, which holds received bytes and bytes to be transmitted, actually consists of two physically different registers. To send, firmware writes a byte to SBUF; to receive, firmware reads SBUF. The receive shift register allows reception of a second byte before the first byte has been read from SBUF. However, if firmware has not read the first byte by the time the second byte is received, the second byte will overwrite the first. The UART sets interrupt bits TI and RI on transmission and reception, respectively. These two bits share a single interrupt request and interrupt vector.

The serial port control (SCON) register (Figure 11-2) configures and controls the serial port.

Table 11-1. Serial Port Signals

Function Name	Type	Description	Multiplexed With
TXD	O	**Transmit Data.** In mode 0, TXD transmits the clock signal. In modes 1, 2, and 3, TXD transmits serial data.	P1.7
RXD	I/O	**Receive Data.** In mode 0, RXD transmits and receives serial data. In modes 1, 2, and 3, RXD receives serial data.	P1.6

Table 11-2. Serial Port Special Function Registers

Mnemonic	Description	Address
SBUF	**Serial Buffer**. Two separate registers, accessed with same address comprise the SBUF register. Writing to SBUF loads the transmit buffer; reading SBUF accesses the receive buffer.	99H
SCON	**Serial Port Control**. Selects the serial port operating mode. SCON enables and disables the receiver, framing bit error detection, multiprocessor communication, automatic address recognition, and the serial port interrupt bits.	98H
SADDR	**Serial Address**. Defines the individual address for a slave device.	A9H
SADEN	**Serial Address Enable.** Specifies the mask byte that is used to define the given address for a slave device.	B9H

11.2 MODES OF OPERATION

The serial I/O port can operate in one synchronous and three asynchronous modes.

11.2.1 Synchronous Mode (Mode 0)

Mode 0 is a half-duplex, synchronous mode, which is commonly used to expand the I/O capabilities of a device with shift registers. The transmit data (TXD) pin outputs a set of eight clock pulses while the receive data (RXD) pin transmits or receives a byte of data. The eight data bits are transmitted and received least-significant bit (LSb) first. Shifts occur in the last phase (S6P2) of every peripheral cycle, which corresponds to a baud rate of $F_{OSC}/12$. Figure 11-3 on page 11-6 shows the timing for transmission and reception in mode 0.

11.2.1.1 Transmission (Mode 0)

Follow these steps to begin a transmission:

1. Write to the SCON register, clearing bits SM0, SM1, and REN.
2. Write the byte to be transmitted to the SBUF register. This write starts the transmission.

Hardware executes the write to SBUF in the last phase (S6P2) of a peripheral cycle. At S6P2 of the following cycle, hardware shifts the LSb (D0) onto the RXD pin. At S3P1 of the next cycle, the TXD pin goes low for the first clock-signal pulse. Shifts continue every peripheral cycle. In the ninth cycle after the write to SBUF, the MSB (D7) is on the RXD pin. At the beginning of the tenth cycle, hardware drives the RXD pin high and asserts TI (S1P1) to indicate the end of the transmission.

11.2.1.2 Reception (Mode 0)

To start a reception in mode 0, write to the SCON register. Clear bits SM0, SM1, and RI and set the REN bit.

Hardware executes the write to SCON in the last phase (S6P2) of a peripheral cycle (Figure 11-3). In the second peripheral cycle following the write to SCON, TXD goes low at S3P1 for the first

clock-signal pulse, and the LSb (D0) is sampled on the RXD pin at S5P2. The D0 bit is then shifted into the shift register. After eight shifts at S6P2 of every peripheral cycle, the LSb (D7) is shifted into the shift register, and hardware asserts RI (S1P1) to indicate a completed reception. Firmware can then read the received byte from SBUF.

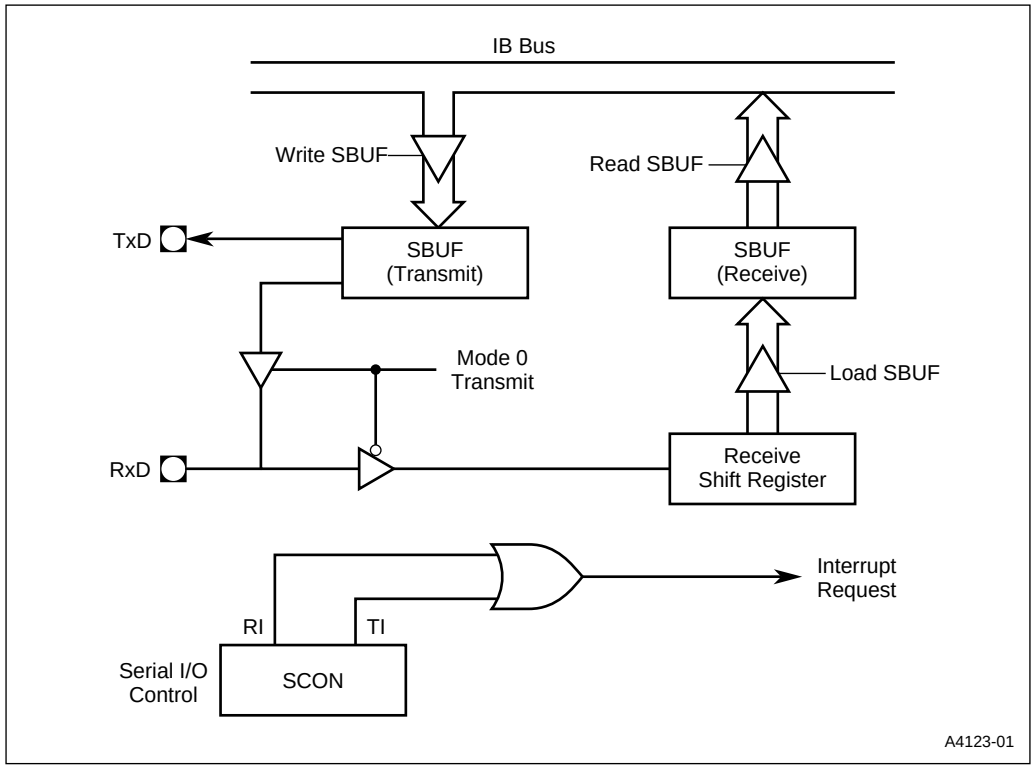


Figure 11-1. Serial Port Block Diagram

SCON Address: 98H
Reset State: 0000 0000B

Serial Port Control Register. SCON contains serial I/O control and status bits, including the mode select bits and the interrupt flag bits.

7							0
FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit Number	Bit Mnemonic	Function
7	FE	Framing Error Bit: To select this function, set the SMOD0 bit in the PCON register. Set by hardware to indicate an invalid stop bit. Cleared by firmware, not by valid frames.
	SM0	Serial Port Mode Bit 0: To select this function, clear the SMOD0 bit in the PCON register. Firmware writes to bits SM0 and SM1 to select the serial port operating mode. Refer to the SM1 bit for the mode selections.
6	SM1	Serial Port Mode Bit 1: Firmware writes to bits SM1 and SM0 (above) to select the serial port operating mode. **SM0 SM1 Mode Description Baud Rate†** 0 0 0 Shift register $F_{OSC}/12$ 0 1 1 8-bit UART Variable 1 0 2 9-bit UART $F_{OSC}/64$†† or $F_{OSC}/32$†† 1 1 3 9-bit UART Variable †F_{OSC} = Oscillator frequency. ††Select by programming the SMOD1 bit in the PCON register (see section "Baud Rates" on page 11-10).
5	SM2	Serial Port Mode Bit 2: Firmware writes to bit SM2 to enable and disable the multiprocessor communication and automatic address recognition features. This allows the serial port to differentiate between data and command frames and to recognize slave and broadcast addresses.
4	REN	Receiver Enable Bit: To enable reception, set this bit. To disable reception, clear this bit.
3	TB8	Transmit Bit 8: In modes 2 and 3, firmware writes the ninth data bit to be transmitted to TB8. Not used in modes 0 and 1.

Figure 11-2. SCON: Serial Port Control Register

SCON (Continued)

Address: 98H
Reset State: 0000 0000B

Serial Port Control Register. SCON contains serial I/O control and status bits, including the mode select bits and the interrupt flag bits.

7							0
FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit Number	Bit Mnemonic	Function
2	RB8	Receiver Bit 8: Mode 0: Not used. Mode 1 (SM2 clear): Set or cleared by hardware to reflect the stop bit received. Modes 2 and 3 (SM2 set): Set or cleared by hardware to reflect the ninth data bit received.
1	TI	Transmit Interrupt Flag Bit: Set by the transmitter after the last data bit is transmitted. Cleared by firmware.
0	RI	Receive Interrupt Flag Bit: Set by the receiver after the last data bit of a frame has been received. Cleared by firmware.

Figure 11-2. SCON: Serial Port Control Register (Continued)

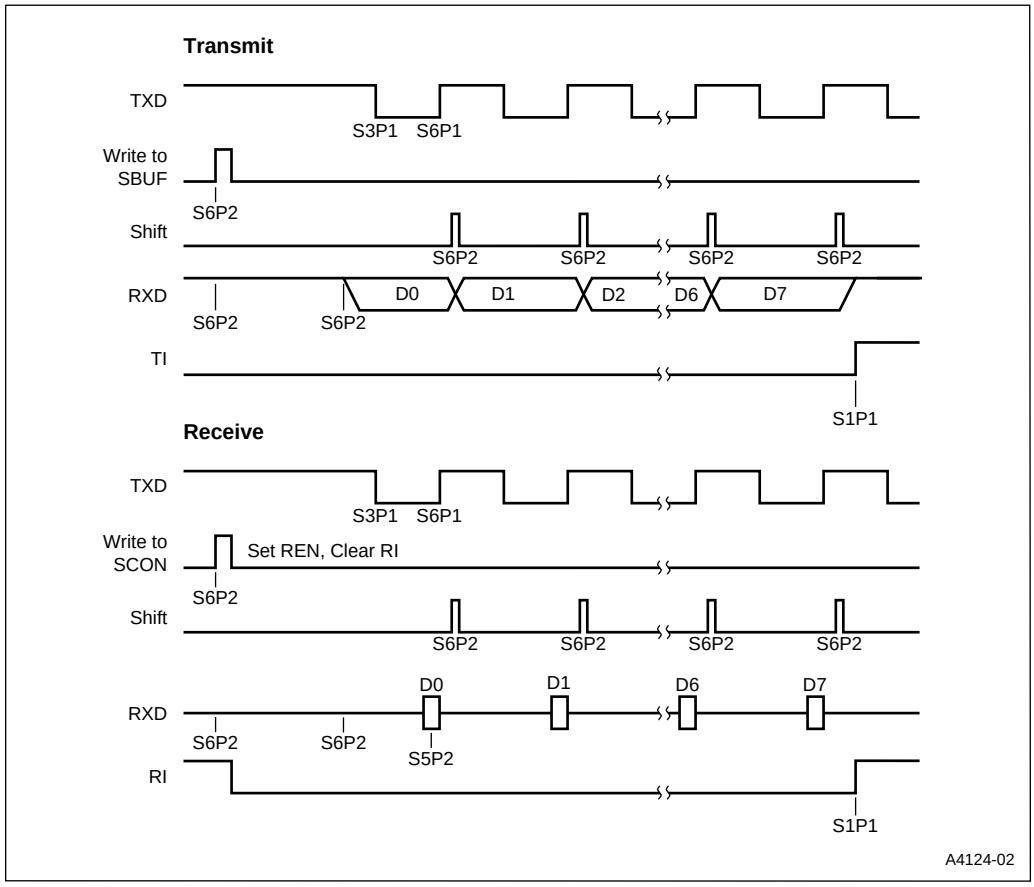


Figure 11-3. Mode 0 Timing

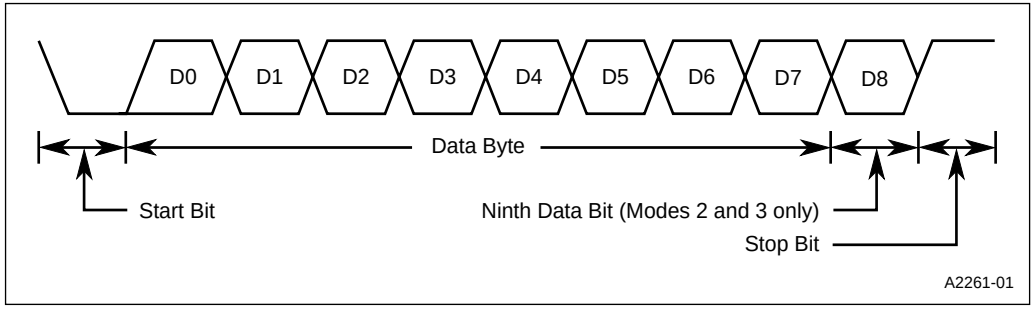


Figure 11-4. Data Frame (Modes 1, 2, and 3)

11.2.2 Asynchronous Modes (Modes 1, 2, and 3)

The serial port has three asynchronous modes of operation:

- **Mode 1.** Mode 1 is a full-duplex, asynchronous mode. The data frame (Figure 11-4) consists of 10 bits: one start bit, eight data bits, and one stop bit. Serial data is transmitted on the TXD pin and received on the RXD pin. When a message is received, the stop bit is read in the RB8 bit in the SCON register. The baud rate is generated by overflow of timer 1 or timer 2 (see "Baud Rates" on page 11-10).
- **Modes 2 and 3.** Modes 2 and 3 are full-duplex, asynchronous modes. The data frame (Figure 11-4) consists of 11 bits: one start bit, eight data bits (transmitted and received LSb first), one programmable ninth data bit, and one stop bit. Serial data is transmitted on the TXD pin and received on the RXD pin. On receive, the ninth bit is read from the RB8 bit in the SCON register. On transmit, the ninth data bit is written to the TB8 bit in the SCON register. Alternatively, you can use the ninth bit as a command/data flag.
  - — In mode 2, the baud rate is programmable to 1/32 or 1/64 of the oscillator frequency, (F_{OSC}).
  - — In mode 3, the baud rate is generated by overflow of timer 1 or timer 2.

11.2.2.1 Transmission (Modes 1, 2, 3)

Follow these steps to initiate a transmission:

1. Write to the SCON register. Select the mode with the SM0 and SM1 bits, and clear the REN bit. For modes 2 and 3, also write the ninth bit to the TB8 bit.
2. Write the byte to be transmitted to the SBUF register. This write starts the transmission.

11.2.2.2 Reception (Modes 1, 2, 3)

To prepare for a reception, set the REN bit in the SCON register. The actual reception is then initiated by a detected high-to-low transition on the RXD pin.

11.3 FRAMING BIT ERROR DETECTION (MODES 1, 2, AND 3)

Framing bit error detection is provided for the three asynchronous modes. To enable the framing bit error detection feature, set the SMOD0 bit in the PCON register (see Figure 15-1 on page 15-3). When this feature is enabled, the receiver checks each incoming data frame for a valid stop bit. An invalid stop bit may result from noise on the serial lines or from simultaneous transmission by two CPUs. If a valid stop bit is not found, the hardware sets the FE bit in the SCON register (see Figure 11-2).

Firmware may examine the FE bit after each reception to check for data errors. Once set, only firmware or a reset can clear the FE bit. Subsequently received frames with valid stop bits cannot clear the FE bit.

11.4 MULTIPROCESSOR COMMUNICATION (MODES 2 AND 3)

Modes 2 and 3 provide a ninth-bit mode to facilitate multiprocessor communication. To enable this feature, set the SM2 bit in the SCON register (see Figure 11-2). When the multiprocessor communication feature is enabled, the serial port can differentiate between data frames (ninth bit clear) and address frames (ninth bit set). This allows the microcontroller to function as a slave processor in an environment where multiple slave processors share a single serial line.

When the multiprocessor communication feature is enabled, the receiver ignores frames with the ninth bit clear. The receiver examines frames with the ninth bit set for an address match. If the received address matches the slave's address, the receiver hardware sets the RB8 bit and the RI bit in the SCON register, generating an interrupt.

NOTE

The ES bit must be set in the IEN0 register to allow the RI bit to generate an interrupt. The IEN0 register is described in Chapter 8, Interrupts.

The addressed slave's firmware then clears the SM2 bit in the SCON register and prepares to receive the data bytes. The other slaves are unaffected by these data bytes because they are waiting to respond to their own addresses.

11.5 AUTOMATIC ADDRESS RECOGNITION

The automatic address recognition feature is enabled when the multiprocessor communication feature is enabled (i.e., the SM2 bit is set in the SCON register).

Implemented in hardware, automatic address recognition enhances the multiprocessor communication feature by allowing the serial port to examine the address of each incoming command frame. Only when the serial port recognizes its own address does the receiver set the RI bit in the SCON register to generate an interrupt. This ensures that the CPU is not interrupted by command frames addressed to other devices.

If desired, you may enable the automatic address recognition feature in mode 1. In this configuration, the stop bit takes the place of the ninth data bit. The RI bit is set only when the received command frame address matches the device's address and is terminated by a valid stop bit.

NOTE

The multiprocessor communication and automatic address recognition features cannot be enabled in mode 0 (i.e., setting the SM2 bit in the SCON register in mode 0 has no effect).

To support automatic address recognition, a device is identified by a *given* address and a *broadcast* address.

11.5.1 Given Address

Each device has an *individual* address that is specified in the SADDR register; the SADEN register is a mask byte that contains don't-care bits (defined by zeros) to form the device's *given* address. These don't-care bits provide the flexibility to address one or more slaves at a time. To address a device by its individual address, the SADEN mask byte must be 1111 1111 The following example illustrates how a given address is formed:

```
SADDR  =  0101 0110
SADEN  =  1111 1100
Given  =  0101 01XX
```

The following is an example of how to use given addresses to address different slaves:

```
Slave A:   SADDR  =  1111 0001        Slave C:   SADDR  =  1111 0010
           SADEN  =  1111 1010                   SADEN  =  1111 1101
           Given  =  1111 0X0X                   Given  =  1111 00X1

                  Slave B:   SADDR  =  1111 0011
                             SADEN  =  1111 1001
                             Given  =  1111 0XX1
```

The SADEN byte is selected so that each slave may be addressed separately. For Slave A, bit 0 (the LSb) is a don't-care bit; for Slaves B and C, bit 0 is a 1. To communicate with Slave A only, the master must send an address where bit 0 is clear (e.g., 1111 0000).

For Slave A, bit 1 is a 0; for Slaves B and C, bit 1 is a don't-care bit. To communicate with Slaves B and C, but not Slave A, the master must send an address with bits 0 and 1 both set (e.g., 1111 0011).

For Slaves A and B, bit 2 is a don't-care bit; for Slave C, bit 2 is a 0. To communicate with Slaves A and B, but not Slave C, the master must send an address with bit 0 set, bit 1 clear, and bit 2 set (e.g., 1111 0101).

To communicate with Slaves A, B, and C, the master must send an address with bit 0 set, bit 1 clear, and bit 2 clear (e.g., 1111 0001).

11.5.2 Broadcast Address

A *broadcast* address is formed from the logical OR of the SADDR and SADEN registers with zeros defined as don't-care bits, e.g.:

```
SADDR                =  0101 0110
SADEN                =  1111 1100
(SADDR) OR (SADEN)   =  1111 111X
```

The use of don't-care bits provides flexibility in defining the broadcast address, however, in most applications, a broadcast address is 0FFH.

The following is an example of using broadcast addresses:

Slave A:	SADDR	=	1111 0001	Slave C:	SADDR	=	1111 0010
	SADEN	=	1111 1010		SADEN	=	1111 1101
	Broadcast	=	1111 1X11		Broadcast	=	1111 1111

Slave B:	SADDR	=	1111 0011
	SADEN	=	1111 1001
	Broadcast	=	1111 1X11

For Slaves A and B, bit 2 is a don't-care bit; for Slave C, bit 2 is set. To communicate with all of the slaves, the master must send an address FFH.

To communicate with Slaves A and B, but not Slave C, the master can send an address FBH.

11.5.3 Reset Addresses

On reset, the SADDR and SADEN registers are initialized to 00H, that is, the given and broadcast addresses are xxxx xxxx (all don't-care bits). This ensures that the serial port is backwards-compatible with MCS® 51 microcontrollers that do not support automatic address recognition.

11.6 BAUD RATES

You must select the baud rate for the serial port transmitter and receiver when operating in modes 1, 2, and 3. (The baud rate is preset for mode 0.) In its asynchronous modes, the serial port can transmit and receive simultaneously. Depending on the mode, the transmission and reception rates can be the same or different. Table 11-3 summarizes the baud rates that can be used for the four serial I/O modes.

NOTE

Setting the low clock (PCON.5) bit forces the internal clock (F_{CLK}) frequency distributed to the CPU and peripherals to 3MHz. This bit is automatically set after a reset. Clearing this bit through firmware returns F_{CLK} to the normal clock frequency (F_{OSC} /2).

11.6.1 Baud Rate for Mode 0

The baud rate for mode 0 is fixed at F_{OSC} /12.

Table 11-3. Summary of Baud Rates

Mode	No. of Baud Rates	Send and Receive at the Same Rate	Send and Receive at Different Rates
0	1	N/A	N/A
1	Many ††	Yes	Yes
2	2	Yes	No
3	Many ††	Yes	Yes

†† Baud rates are determined by overflow of timer 1 and/or timer 2.

11.6.2 Baud Rates for Mode 2

Mode 2 has two baud rates, which are selected by the SMOD1 bit in the PCON register (Figure 15-1 on page 15-3). The following expression defines the baud rate:

$$\text{Serial I/O Mode 2 Baud Rate} = 2^{SMOD1} \times \frac{F_{osc}}{64}$$

11.6.3 Baud Rates for Modes 1 and 3

In modes 1 and 3, the baud rate is generated by overflow of timer 1 (default) and/or timer 2. You may select either or both timer(s) to generate the baud rate(s) for the transmitter and/or the receiver.

11.6.3.1 Timer 1 Generated Baud Rates (Modes 1 and 3)

Timer 1 is the default baud rate generator for the transmitter and the receiver in modes 1 and 3. The baud rate is determined by the timer 1 overflow rate and the value of SMOD, as shown in the following formula:

$$\text{Serial I/O Modes 1 and 3 Baud Rate} = 2^{SMOD1} \times \frac{\text{Timer 1 Overflow Rate}}{32}$$

11.6.3.2 Selecting Timer 1 as the Baud Rate Generator

To select timer 1 as the baud rate generator:

- Disable the timer interrupt by clearing the ET1 bit in the IEN0 register (Figure 6-12 on page 6-25).
- Configure timer 1 as a timer or an event counter (set or clear the C/T# bit in the TMOD register, Figure 11-5 on page 11-7).
- Select timer mode 0–3 by programming the M1 and M0 bits in the TMOD register.

In most applications, timer 1 is configured as a timer in auto-reload mode (high nibble of TMOD = 0010B). The resulting baud rate is defined by the following expression:

$$\text{Serial I/O Modes 1 and 3 Baud Rate} = 2^{SMOD1} \times \frac{F_{osc}}{32 \times 12 \times [256 - (TH1)]}$$

Timer 1 can generate very low baud rates with the following setup:

- Enable the timer 1 interrupt by setting the ET1 bit in the IEN0 register.
- Configure timer 1 to run as a 16-bit timer (high nibble of TMOD = 0001B).
- Use the timer 1 interrupt to initiate a 16-bit firmware reload.

Table 11-4 lists commonly used baud rates and shows how they are generated by timer 1.

Table 11-4. Timer 1 Generated Baud Rates for Serial I/O Modes 1 and 3

Baud Rate	Oscillator Frequency (F_{OSC})	SMOD1	Timer 1		
			C/T#	Mode	Reload Value
62.5 Kbaud (Max)	12.0 MHz	1	0	2	FFH
19.2 Kbaud	11.059 MHz	1	0	2	FDH
9.6 Kbaud	11.059 MHz	0	0	2	FDH
4.8 Kbaud	11.059 MHz	0	0	2	FAH
2.4 Kbaud	11.059 MHz	0	0	2	F4H
1.2 Kbaud	11.059 MHz	0	0	2	E8H
137.5 Baud	11.986 MHz	0	0	2	1DH
110.0 Baud	6.0 MHz	0	0	2	72H
110.0 Baud	12.0 MHz	0	0	1	FEEBH

11.6.3.3 Timer 2 Generated Baud Rates (Modes 1 and 3)

Timer 2 may be selected as the baud rate generator for the transmitter and/or receiver (Figure 11-5). The timer 2 baud rate generator mode is similar to the auto-reload mode. A rollover in the TH2 register reloads registers TH2 and TL2 with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by firmware.

The timer 2 baud rate is expressed by the following formula:

$$\text{Serial I/O Modes 1 and 3 Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

11.6.3.4 Selecting Timer 2 as the Baud Rate Generator

To select timer 2 as the baud rate generator for the transmitter and/or receiver, program the RCLCK and TCLCK bits in the T2CON register as shown in Table 11-5. (You may select different baud rates for the transmitter and receiver.) Setting RCLK and/or TCLK puts timer 2 into its baud rate generator mode (Figure 11-5). In this mode, a rollover in the TH2 register does not set the TF2 bit in the T2CON register. Also, a high-to-low transition at the T2EX pin sets the EXF2 bit in the T2CON register but does not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). You can use the T2EX pin as an additional external interrupt by setting the EXEN2 bit in T2CON.

NOTE

Turn the timer off (clear the TR2 bit in the T2CON register) before accessing registers TH2, TL2, RCAP2H, and RCAP2L.

You may configure timer 2 as a timer or a counter. In most applications, it is configured for timer operation (i.e., the C/T2# bit is clear in the T2CON register).

Table 11-5. Selecting the Baud Rate Generator(s)

RCLCK Bit	TCLCK Bit	Receiver Baud Rate Generator	Transmitter Baud Rate Generator
0	0	Timer 1	Timer 1
0	1	Timer 1	Timer 2
1	0	Timer 2	Timer 1
1	1	Timer 2	Timer 2

Note that timer 2 increments every state time ($2T_{OSC}$) when it is in the baud rate generator mode. In the baud rate formula that follows, “RCAP2H, RCAP2L” denotes the contents of RCAP2H and RCAP2L taken as a 16-bit unsigned integer:

$$\text{Serial I/O Modes 1 and 3 Baud Rate} = \frac{\text{Fosc}}{32 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

NOTE

When timer 2 is configured as a timer and is in baud rate generator mode, do not read or write the TH2 or TL2 registers. The timer is being incremented every state time, and the results of a read or write may not be accurate. In addition, you may read, but not write to, the RCAP2 registers; a write may overlap a reload and cause write and/or reload errors.

Table 11-6 lists commonly used baud rates and shows how they are generated by timer 2.

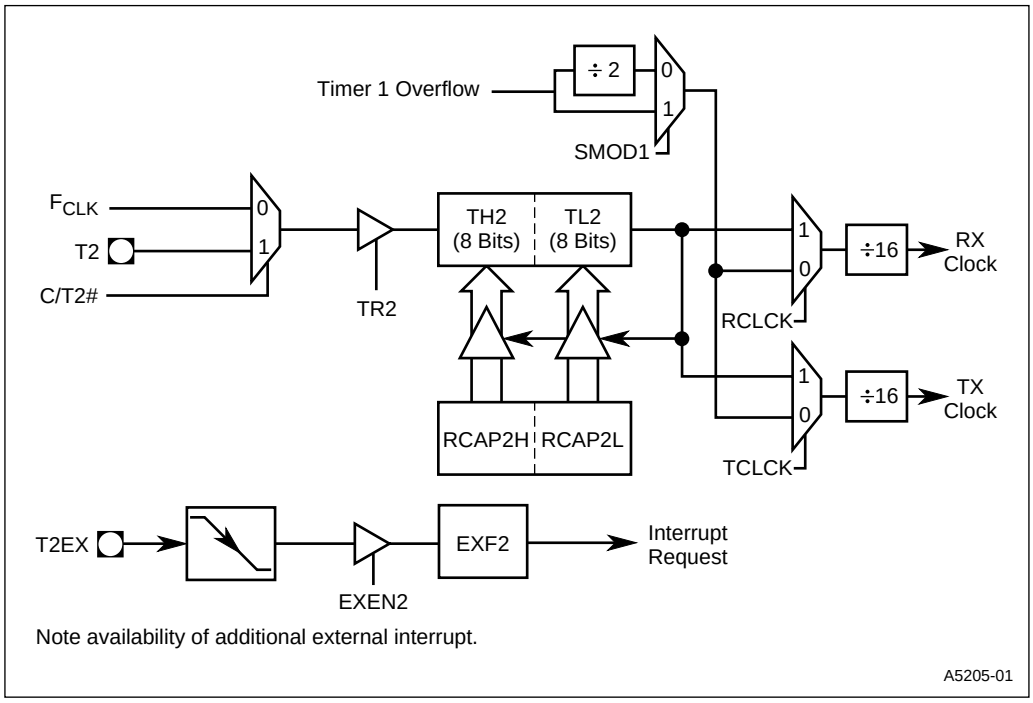


Figure 11-5. Timer 2 in Baud Rate Generator Mode

Table 11-6. Timer 2 Generated Baud Rates

Baud Rate	Oscillator Frequency (F_{OSC})	RCAP2H	RCAP2L
375.0 Kbaud	12 MHz	FFH	FFH
9.6 Kbaud	12 MHz	FFH	D9H
4.8 Kbaud	12 MHz	FFH	B2H
2.4 Kbaud	12 MHz	FFH	64H
1.2 Kbaud	12 MHz	FEH	C8H
300.0 baud	12 MHz	FBH	1EH
110.0 baud	12 MHz	F2H	AFH
300.0 baud	6 MHz	FDH	8FH
110.0 baud	6 MHz	F9H	57H

12

Keyboard Control

CHAPTER 12
KEYBOARD CONTROL

This chapter describes the 8*x*931 keyboard control interface.

12.1 OVERVIEW

The 8*x*931 keyboard control interface consists of 20 keyboard scan output lines, eight keyboard scan input lines, and four LED drivers. The output lines, input lines, and LEDs are controlled by the KBCON SFR, shown in Figure 12-2 below.

KBCON Address: F8H
Reset State: 0xx0 0000B

Keyboard Control Register. This register controls the keyboard scan input and output activity, enables and configures the keyboard scan interrupt, and drives the keyboard LEDs.

7							0
IE2	—	KSEN	IT2	LED3	LED2	LED1	LED0

Bit Number	Bit Mnemonic	Function
7	IE2	Interrupt 2 Flag: Set when external interrupt 2 is detected if the KSEN bit is set. Firmware must clear this bit when the interrupt is serviced.
6	—	Reserved: Write a zero to this bit.
5	KSEN	Keyboard Scan Enable: Setting this bit enables the pullup resistors on the KSI input lines, enables the keyboard scan interrupt (INT2#), and enables the LED drivers. **NOTE:** The EX2 bit in the IEN0 SFR must also be set to enable the KSI external interrupt.
4	IT2	Interrupt 2 Type Control Bit: If set, a negative edge detect on any of the KSI pins causes IE2 to be set. When clear, IE2 acts as a level 0 triggered interrupt.
3:0	LED3:0	LED Driver Control: Clearing one of these bits turns on the associated LED. Setting a bit turns off the associated LED. **NOTE:** The KSEN (Keyboard Scan Enable) bit must be set in order to activate the LED drivers. After reset, the LED driver control bits are cleared. This means that when KSEN is set, the LEDs will turn on. Firmware must set the LED driver control bits to turn off the LEDs.

Figure 12-1. KBCON: Keyboard Control Register

Table 12-1. Keyboard Control Signals

Signal Name	Type	Description	Multiplexed With
KSO19 KSO18 KSO17:16 KSO15:8 KSO7:0	O	**Keyboard Scan Output.** Quasi-bidirectional ports with weak internal pullup resistors used for the output side of the keyboard scan matrix.	P3.7/RD# P3.6/WR# P3.5:4/T1:0 A15:8/P2.7:0 P1.7:0
KSI7:0	I	**Keyboard Scan Input.** Schmitt-trigger inputs with firmware-enabled internal pullup resistors used for the input side of the keyboard scan matrix.	AD7:0/P0.7:0
LED3:0	O	**LED Drivers.** These drive LEDs connected directly to $V_{CC.}$ The current each driver is capable of sinking is given as V_{OL2} in the datasheet.	—

NOTE: Other signals are defined in their respective chapters and in Appendix B, "Pin Descriptions".

12.2 KEYBOARD SCAN IMPLEMENTATION

The keyboard scan matrix supports up to 160 keys using 20 keyboard scan outputs (KSO) and eight keyboard scan inputs (KSI). The KSOs are implemented as quasi-bidirectional ports with weak internal pullup resistors. To reduce overall system cost, each KSI is implemented as a Schmitt-trigger input incorporating an on-chip pullup resistor that may be enabled or disabled through firmware.

A typical implementation of the keyboard scan matrix is shown in Figure 12-2. Note that the pullup resistors are inactive until the pullup enable bit (KSEN in KBCON) is set.

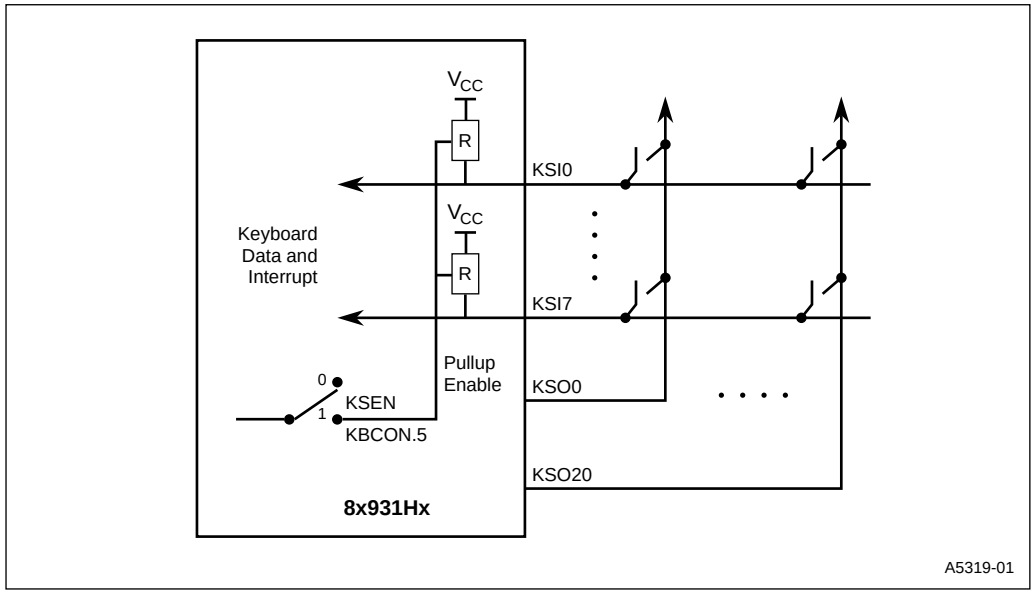


Figure 12-2. Keyboard Scan Matrix Application

12.2.1 Keyboard Interrupt Logic

Firmware must perform the keyboard scan polling operation by generating a running 0 through the KSO outputs. The sampling operation is interrupt-driven using external interrupt 2 (INT2#). All KSI inputs are ANDed together so that a negative edge (or level 0) on any of the KSI inputs causes INT2# to be generated, setting KBCON's IE2 bit. When this interrupt occurs, firmware must read the KSI inputs to determine which one(s) caused the interrupt.

In order for the keyboard scan interrupt to work properly, the following bits must be set in KBCON (Figure 12-1) and IEN1 (Figure 5-11 on page 5-25):

- The global interrupt enable bit must be set (EA of IEN0) — to allow the maskable interrupts to be individually enabled.
- The enable bit for external interrupt 2 (INT2#) must be set (EX2 of IEN1) — to allow an interrupt to be triggered in hardware.
- The keyboard scan enable bit must be set (KSEN of KBCON) — to activate the KSI pullups and enable the keyboard scan interrupt flag (IE2 of KBCON). If KSEN is set, the IE2 flag may be set even if the hardware interrupt is disabled (i.e., EX2 = 0).
- Additionally, the Interrupt 2 Type control bit (IT2 of KBCON) must be set or cleared to specify whether the interrupt will be triggered on negative edge or level 0.

12.3 LED DRIVERS

The LED drivers enable external LEDs to be connected directly between V_{CC} and the LED driver pins without the need for external resistors. The current each driver is capable of sinking is given as V_{OL2} in the datasheet.

NOTE

The KSEN (keyboard scan enable) bit must be set to activate the LED drivers.

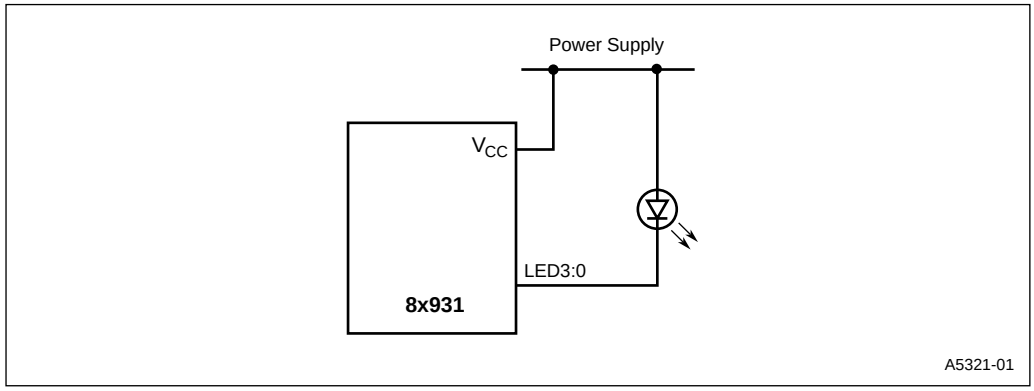


Figure 12-3. LED Driver Application

13

Minimum Hardware Setup

CHAPTER 13
MINIMUM HARDWARE SETUP

This chapter discusses the basic operating requirements of the 8*x*931 and describes a minimum hardware setup. Topics covered include power, ground, clock source, and device reset. For parameter values, refer to the device data sheet.

13.1 MINIMUM HARDWARE SETUP

Figure 13-1 shows a minimum hardware setup that employs the on-chip oscillator for the system clock and provides power-on reset. Control signals, Ports 0–3, and the USB ports are not shown. See "Clock Sources" on page 13-2 and "Power-on Reset" on page 13-7. PLLSEL selects the USB operating rate. Refer to Table 2-4 on page 2-10.

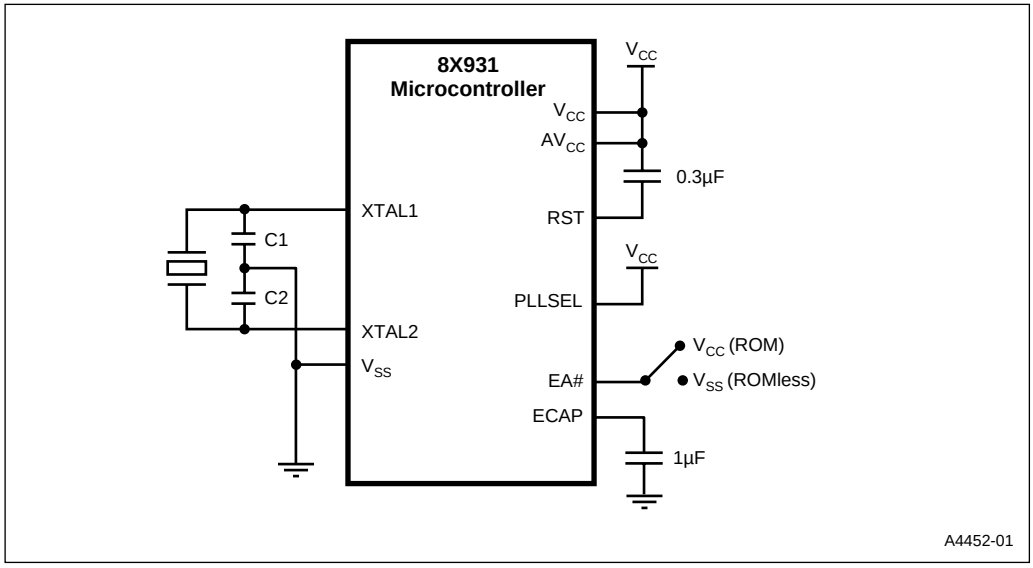


Figure 13-1. Minimum Setup

13.2 ELECTRICAL ENVIRONMENT

The 8*x*931 is a high-speed CHMOS device. To achieve satisfactory performance, its operating environment should accommodate the device signal waveforms without introducing distortion or noise. Design considerations relating to device performance are discussed in this section. See the device data sheet for voltage and current requirements, operating frequency, and waveform timing.

13.2.1 Power and Ground Pins

Power the 8*x*931 from a well-regulated power supply designed for high-speed digital loads. Use short, low impedance connections to the power (V_{CC}) and ground (V_{SS}) pins.

13.2.2 Unused Pins

To provide stable, predictable performance, connect unused input pins to V_{SS} or V_{CC}. Unterminated input pins can float to a mid-voltage level and draw excessive current. Unterminated interrupt inputs may generate spurious interrupts.

For the 8*x*931HA, if the USB downstream ports are not used, the two data lines are still required to be pulled low externally using 15K pulldown resistors so inputs are not floating.

13.2.3 Noise Considerations

The fast rise and fall times of high-speed CHMOS logic may produce noise spikes on the power supply lines and signal outputs. To minimize noise and waveform distortion, follow good board layout techniques. Use sufficient decoupling capacitors and transient absorbers to keep noise within acceptable limits. Connect 0.1 µF bypass capacitors between V_{CC}/AV_{CC} and each V_{SS} pin. Place the capacitors close to the device to minimize path lengths.

Multi-layer printed circuit boards with separate V_{CC} and ground planes help minimize noise. For additional information on noise reduction, see Application Note AP-125, "Designing Microcontroller Systems for Electrically Noisy Environments."

13.3 CLOCK SOURCES

The 8*x*931 can use an external clock (Figure 13-3), an on-chip oscillator with crystal or ceramic resonator (Figure 13-2), or an on-chip phase-locked oscillator (locked to the external clock or the on-chip oscillator) as its clock source. For USB operating rates, see Table 2-3 on page 2-9.

13.3.1 On-chip Oscillator (Crystal)

This clock source uses an external quartz crystal connected from XTAL1 to XTAL2 as the frequency-determining element (Figure 13-2). The crystal operates in its fundamental mode as an inductive reactance in parallel resonance with capacitance external to the crystal. Oscillator design considerations include crystal specifications, operating temperature range, and parasitic board capacitance. Consult the crystal manufacturer's data sheet for parameter values. With high quality components, C1 = C2 = 30 pF is adequate for this application.

Pins XTAL1 and XTAL2 are protected by on-chip electrostatic discharge (ESD) devices, D1 and D2, which are diodes parasitic to the R_F FETs. They serve as clamps to V_{CC} and V_{SS}. Feedback resistor R_F in the inverter circuit, formed from paralleled n- and p- channel FETs, permits the PD bit in the PCON register (Figure 14-1 on page 14-3) to disable the clock during powerdown.

Noise spikes at XTAL1 and XTAL2 can disrupt microcontroller timing. To minimize coupling between other digital circuits and the oscillator, locate the crystal and the capacitors near the chip and connect to XTAL1, XTAL2, and V_{SS} with short, direct traces. To further reduce the effects of noise, place guard rings around the oscillator circuitry and ground the metal crystal case.

For a more in-depth discussion of crystal specifications, ceramic resonators, and the selection of C1 and C2 see Application Note AP-155, "*Oscillators for Microcontrollers*," in the Embedded Applications handbook. See Table 1.3 on page 1-6 for the order number.

13.3.2 On-chip Oscillator (Ceramic Resonator)

In cost-sensitive applications, you may choose a ceramic resonator instead of a crystal. Ceramic resonator applications may require slightly different capacitor values and circuit configuration. Consult the manufacturer's data sheet for specific information.

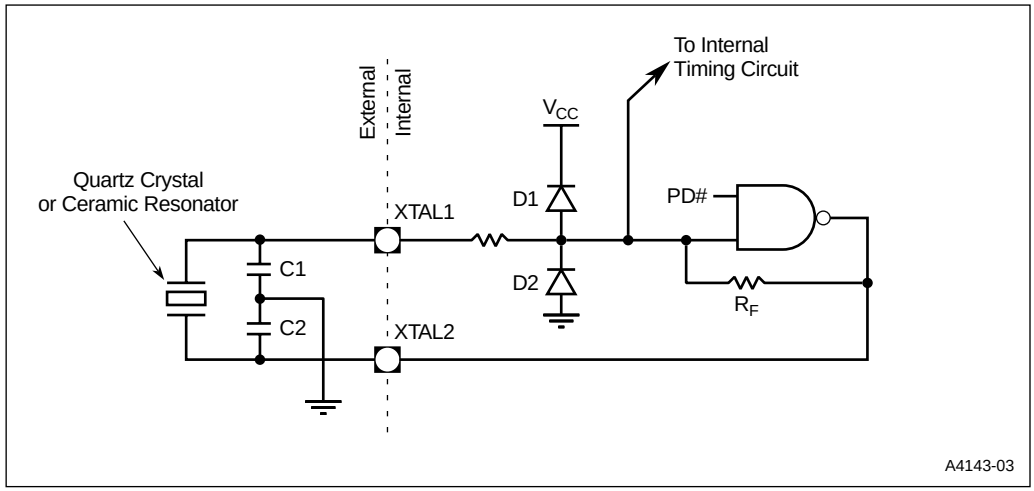


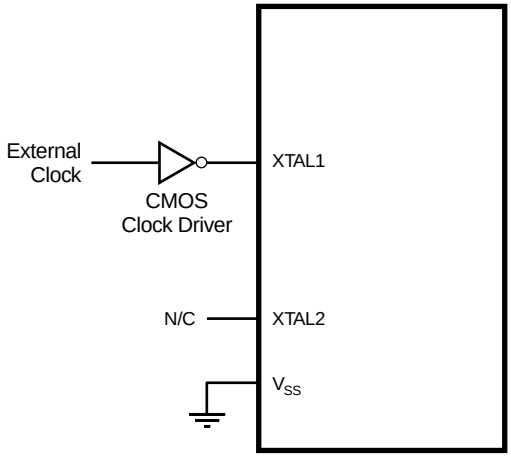
Figure 13-2. CHMOS On-chip Oscillator

13.3.3 External Clock

To operate the 8*x*931 from an external clock, connect the clock source to the XTAL1 pin as shown in Figure 13-3. Leave the XTAL2 pin floating. The external clock driver can be a CMOS gate. If the clock driver is a TTL device, its output must be connected to V_{CC} through a 4.7 kΩ pullup resistor.

For external clock drive requirements, see the device data sheet. Figure 13-4 shows the clock drive waveform. The external clock source must meet the minimum high and low times (T_{CHCX} and T_{CLCX}) and the maximum rise and fall times (T_{CLCH} and T_{CHCL}) to minimize the effect of external noise on the clock generator circuit. Long rise and fall times increase the chance that external noise will affect the clock circuitry and cause unreliable operation.

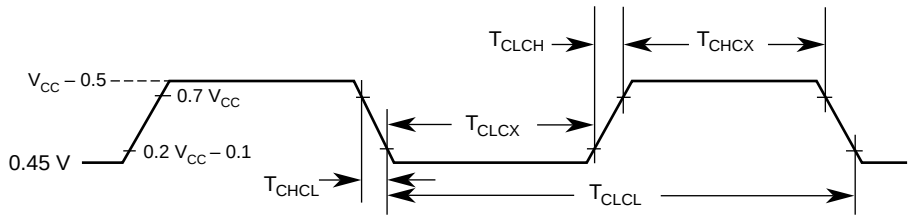
The external clock driver may encounter increased capacitance loading at XTAL1 when power is applied, due to the interaction between the internal amplifier and its feedback capacitance (i.e., the Miller effect). Once the input waveform requirements are met, the input capacitance remains under 20 pF.



Note: If TTL clock driver is used, connect a 4.7kΩ pullup resistor from driver output to V_{CC}.

A4142-03

Figure 13-3. External Clock Connection for the 8*x*931



A4119-01

Figure 13-4. External Clock Drive Waveforms

13.4 RESET

A device reset initializes the 8x931 and vectors the CPU to address 0000H. A reset is a means of exiting the idle and powerdown modes or recovering from firmware malfunctions, and could be a USB reset initiated by the host or upstream hub.

NOTE

A reset is required after applying power.

To achieve a valid reset, V_{CC} must be within its normal operating range (see device data sheet) and the reset signal must be maintained for at least two machine cycles (24 oscillator periods) after the oscillator has stabilized.

Device reset is initiated in two ways:

- externally, by asserting the RST pin
- over the bus, by a USB-initiated reset

These reset mechanisms are ORed to create a single reset signal for the 8*x*931.

The power off flag (POF) in the PCON register indicates whether a reset is a warm start or a cold start. A cold start reset (POF = 1) is a reset that occurs after power has been off or V_{CC} has fallen below 3 V, so the contents of volatile memory are indeterminate. POF is set by hardware when V_{CC} rises from less than 3 V to its normal operating level. See “Power Off Flag” on page 14-1. A warm start reset (POF = 0) is a reset that occurs while the chip is at operating voltage, for example, an external reset used to terminate the idle or powerdown modes.

13.4.1 Externally-initiated Resets

To reset the 8*x*931, hold the RST pin at a logic high for at least two machine cycles (24 oscillator periods) while the oscillator is running. Reset can be accomplished automatically at the time power is applied by capacitively coupling RST to V_{CC} (see Figure 13-1 on page 13-1 and “Power-on Reset” on page 13-7). The RST pin has a Schmitt trigger input and a pulldown resistor.

13.4.2 USB-initiated Resets

The 8*x*931 can be reset by the host or upstream hub if a reset signal is detected by the SIE. This reset signal is defined as an SE0 held longer than 2.5 µs. A USB-initiated reset will reset all of the 8*x*931 hardware, even if the device is suspended (in which case it would first wake-up, then reset). See “USB Power Control” on page 14-7 for additional information about USB-related suspend and resume.

A peripheral that is reset must be re-enumerated. This procedure is given in “Enumeration” on page 8-2.

NOTE

You must ensure that the time from connection of this USB device to the bus until the entire reset process is complete (including firmware initialization of

the 8*x*931) is less than 10 ms. After 10 ms, the host may attempt to communicate with the 8*x*931 to set its device address. If the 8*x*931 firmware cannot respond to the host at this time, the host may disable the device after three attempts to communicate.

13.4.2.1 USB Reset Separation

The 8*x*931 features an optional USB reset that functions independently from the chip reset. When the PCON1 SFR's URDIS bit is set, the 8*x*931 core and peripherals will not reset when a USB reset signal is detected. After an 8*x*931 with URDIS set detects a USB reset signal, it resets all the USB blocks (including the USB SFRs), sets the URST bit in PCON1, and generates a USB reset interrupt. For a complete description of the optional USB reset for the 8*x*931, see "USB Reset Separation" on page 5-17.

13.4.3 Reset Operation

When a reset is initiated, whether externally or over the bus, the port pins are immediately forced to their reset condition as a fail-safe precaution, whether the clock is running or not.

The external reset signal and the USB-initiated reset signals are combined internally. For an external reset the voltage on the RST pin must be held high for at least two machine cycles after the oscillator and on-chip PLL stabilize (approximately 5 ms). For USB-initiated resets, a five-bit counter in the reset logic maintains the signal for the required time. Refer to Table 2-3 on page 2-9.

The external reset signal is asynchronous to the internal clock. The RST pin is sampled during State 5 Phase 2 of every machine cycle. ALE and PSEN will maintain their current activities for 19 oscillator periods after a logic 1 has been sampled at the RST pin; that is, for 19 to 31 oscillator periods after the external reset signal has been applied to the RST pin. The port pins are driven to their reset state as soon as a valid high is detected on the RST pin, regardless of whether the clock is running.

When a reset is detected, the CPU responds by triggering the internal reset routine. The reset routine loads the SFRs, including the ACC, B, stack pointer, and data pointer registers, with their reset values (see Table C-1 on page C-2). Reset does not affect on-chip data RAM or the register file. (However, following a cold start reset, these are indeterminate because V_{CC} has fallen too low or has been off.) Following a synchronizing operation, the CPU vectors to address 0000. Figure 13-5 shows the reset timing sequence.

While the RST pin is high ALE, PSEN#, and the port pins are weakly pulled high. After RST is pulled low, it will take 1 to 2 machine cycles for ALE and PSEN# to start clocking. For this reason, other devices can not be synchronized to the internal timings of the 8*x*931.

NOTE

Externally driving the ALE and/or PSEN# pins to 0 during the reset routine may cause the device to go into an indeterminate state.

Powering up the 8x931 without a reset may improperly initialize the program counter and SFRs and cause the CPU to execute instructions from an undetermined memory location.

13.4.4 Power-on Reset

To automatically generate a reset when power is applied, connect the RST pin to the V_{CC} pin through a 0.3 µF capacitor as shown in Figure 13-1 on page 13-1.

When V_{CC} is applied, the RST pin rises to V_{CC}, then decays exponentially as the capacitor charges. The time constant must be such that RST remains high (above the turn-off threshold of the Schmitt trigger) long enough for the oscillator to start and stabilize, plus two machine cycles. At power up, V_{CC} should rise within approximately 10 ms. Oscillator start-up time is a function of the crystal frequency. During power up, the port pins are in a random state until forced to their reset state by the asynchronous logic. Reducing V_{CC} quickly to 0 causes the RST pin voltage to momentarily fall below 0 V. This voltage is internally limited and does not harm the device.

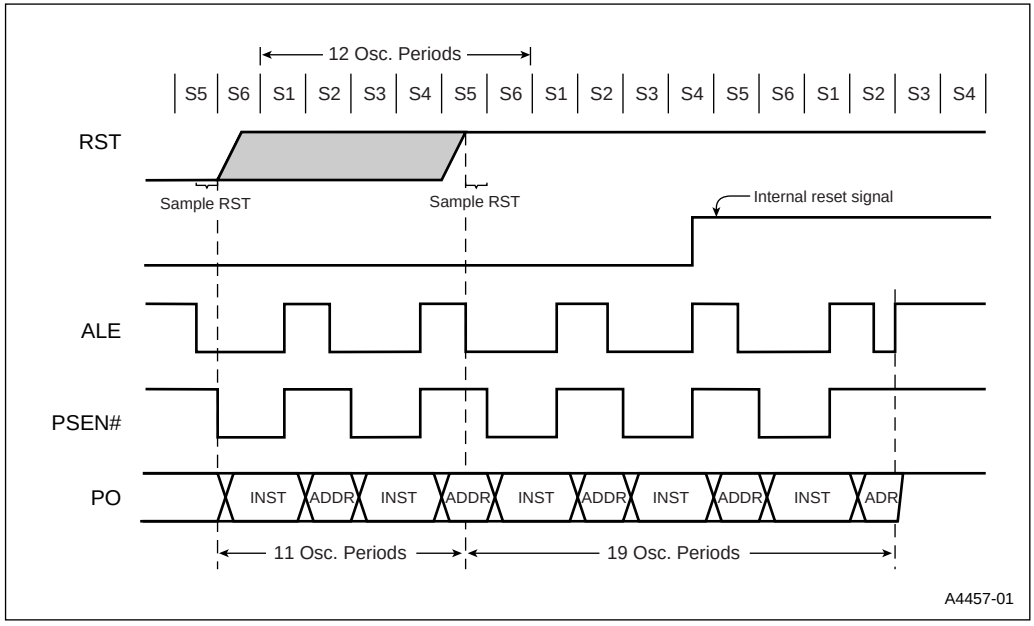


Figure 13-5. Reset Timing Sequence

14

Special Operating Modes

CHAPTER 14
SPECIAL OPERATING MODES

This chapter describes the idle, powerdown, low-clock, and on-circuit emulation (ONCE) device operating modes and the USB function suspend and resume operations. The SFRs associated with these operations (PCON and PCON1) are also described.

14.1 OVERVIEW

The idle, low clock, and powerdown modes are power reduction modes for use in applications where power consumption is a concern. User instructions activate these modes by setting bits in the PCON register. Program execution halts, but resumes when the mode is exited by an interrupt. While in idle or powerdown modes, the V_{CC} pin is the input for backup power.

Following chip reset, the 8*x*931 operates in low-clock mode, wherein the CPU and on-chip peripherals are clocked at a reduced rate until bus enumeration is accomplished. This reduces I_{CC} to meet the 100 mA USB requirement.

Suspend and resume are low current modes used when the USB bus is idle. The 8*x*931 enters suspend when there is a continuous idle state on the bus lines for more than 3.0 msec. When a device is in suspend state, it draws less than 500 µA from the bus. Once a device is in the suspend state, its operation can be resumed by receiving resume signaling on the bus.

ONCE is a test mode that electrically isolates the 8*x*931 from the system in which it operates.

Table 14-1 on page 14-6 lists the condition of the out pins for the various operating modes.

14.2 POWER CONTROL REGISTERS

The PCON special function register (Figure 14-1) provides bits for selecting: the idle, low-clock, and powerdown modes, the power off flag, and two general purpose flags.

The PCON1 SFR (Figure 14-2) provides USB power control, including the USB global suspend/resume and USB function suspend. The PCON1 SFR is discussed further in “USB Power Control” on page 14-7.

14.2.1 Power Off Flag

The Power Off Flag (POF) located at PCON.4, is set by hardware when V_{CC} rises from 0 to 5 Volts. POF can also be set or cleared by software. This allows the user to distinguish between a “cold start” reset and a “warm start” reset.

A cold start reset is one that is coincident with the V_{CC} being turned on to the device after it was turned off. A warm start reset occurs while V_{CC} is still applied to the device and could be generated, for example, by an exit from Power Down.

Immediately after reset, the user's software can check the status of the POF bit. POF = 1 would indicate a cold start. The software then clears POF and commences its tasks. POF = 0 immediately after reset would indicate a warm start.

NOTE

V_{CC} must remain above 3 volts for POF to retain a 0.

PCON Address: 87H
Reset State: 001X 0000B

Power Control Register. Contains the power off flag (POF) and bits for enabling the idle and powerdown modes and two general-purpose flags.

7							0
SMOD1	SMOD0	LC	POF	GF1	GF0	PD	IDL

Bit Number	Bit Mnemonic	Function
7	SMOD1	Double Baud Rate Bit: When set, doubles the baud rate when timer 1 is used and mode 1, 2, or 3 is selected in the SCON register. See “Baud Rates” on page 11-10.
6	SMOD0	SCON.7 Select: When set, read/write accesses to SCON.7 are to the FE bit. When clear, read/write accesses to SCON.7 are to the SM0 bit.See the SCON register (Figure 11-2 on page 11-4).
5	LC	Low-clock Mode Enable: Setting this bit forces the internal clock (F_{CLK}) distributed to the CPU and peripherals (but not the USB module) to 3 MHz. This bit is automatically set after a reset. Clearing this bit through firmware returns F_{CLK} to the normal clock frequency.
4	POF	Power Off Flag: Set by hardware on the rising edge of Vcc. set or cleared by software. This flag allows detection of a reset caused by a power failure. Vcc must remain above 3 volts to retain this bit.
3	GF1	General Purpose Flag: Set or cleared by firmware. One use is to indicate whether an interrupt occurred during normal operation or during idle mode.
2	GF0	General Purpose Flag: Set or cleared by firmware. One use is to indicate whether an interrupt occurred during normal operation or during idle mode.
1	PD	Powerdown Mode Bit: When set, activates powerdown mode. This bit should only be set if the GSUS bit is also set. Cleared by hardware when an interrupt or reset occurs.
0	IDL	Idle Mode Bit: When set, activates idle mode. Cleared by hardware when an interrupt or reset occurs. If IDL and PD are both set, PD takes precedence.

Figure 14-1. PCON: Power Control Register

PCON1 Address: DFH
Reset State: xxxx x000B

USB Power Control Register. Facilitates the control and status relating to global suspend and resume, USB reset separation, and remote wake-up of the 8*x*931.

7							0
—	—	—	URDIS	URST	RWU	GRSM	GSUS

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved: Write zeros to these bits.
4	URDIS	USB Reset Disable: When cleared by firmware, a chip reset occurs upon receiving of a USB reset signal. This resets the MCS® 51 microcontroller core, USB blocks and all peripherals. When set by firmware, the core and peripherals will not reset when a USB reset signal is detected. Upon detecting a USB reset signal, the 8*x*931 resets all the USB blocks (FIFOs, FIU, SIE, and transceiver), sets the URST bit and generates a USB reset interrupt (refer to the description of URST).
3	URST	USB Reset Flag: This flag will be set by hardware when a USB reset occurs, regardless of whether the ESR bit in the IEN1 register is enabled or disabled. The URST also serves as the interrupt bit, ORed with GRSM and GSUS bits to generate an interrupt. Should be cleared by firmware when serving the USB reset interrupt.
2	RWU	Remote Wake-up Bit: 1 = wake-up. This bit is used by the USB function to initiate a remote wake-up. Set by firmware to drive resume signaling on the USB lines to the host or upstream hub. Cleared by hardware when resume signaling is done. **NOTE:** Do not set this bit unless the USB function is suspended (GSUS = 1 and GRSM = 0). See Figure 14-3 on page 14-11.

† Firmware should prioritize GRSM over GSUS if both bits are set simultaneously.

Figure 14-2. PCON1: USB Power Control Register

PCON1 (Continued) Address: DFH
Reset State: xxxx x000B

USB Power Control Register. Facilitates the control and status relating to global suspend and resume, USB reset separation, and remote wake-up of the 8*x*931.

7							0
—	—	—	URDIS	URST	RWU	GRSM	GSUS

1	GRSM	Global Resume Bit: 1 = resume. Set by hardware when a global resume is detected on the USB lines. This bit is ORed with GSUS to generate the interrupt† Cleared by firmware when servicing the global suspend/resume interrupt. (This bit can also be set/cleared by firmware for testability.) This bit is not set if remote wakeup is used (see RWU). See Figure 14-3 on page 14-11.
0	GSUS	Global Suspend Bit: 1 = suspend. This bit is set by hardware when global suspend is detected on the USB lines. This bit is ORed with the GRSM bit to generate the interrupt.† During the global suspend ISR, firmware should set the PD bit to enter the suspend mode. Cleared by hardware when a resume occurs. See Figure 14-3 on page 14-11.

† Firmware should prioritize GRSM over GSUS if both bits are set simultaneously.

Figure 14-2. PCON1: USB Power Control Register (Continued)

14.3 IDLE MODE

Table 14-1. Pin Conditions in Various Modes

Pin	Reset	Idle Mode		Powerdown Mode		Once Mode
	—	Internal Program Memory	External Memory	Internal Program Memory	External Memory	—
ALE	Weak High	1	1	0	0	Float
PSEN#	Weak High	1	1	0	0	Float
Port 0 Pins	Float	Data	Float	Data	Float	Float
Port 1 Pins	Weak High	Data	Data	Data	Data	Float
Port 2 Pins	Weak High	Data	Weak High	Data	Weak High	Float
Port 3 Pins	Weak High	Data	Data	Data	Data	Float
SOF#	Weak High	Data	Data	Data	Data	Weak High
D_{P0}	Float	Data	Data	Float	Float	Weak High
D_{M0}	Float	Data	Data	Float	Float	Float
$D_{P5:2}$	Float	Data	Data	Float	Float	Float
$D_{M5:2}$	Float	Data	Data	Float	Float	Float
UPWEN#	Weak High	Data	Data	Data	Data	Float

Idle mode is a power reduction mode that reduces power consumption to approximately 40% of normal. In this mode, program execution halts. Idle mode freezes the clocks to the CPU at known states while the peripherals continue to be clocked (Figure 2-4 on page 2-9). The CPU status before entering idle mode is preserved. That is, the program counter, program status word register, and register file retain their data for the duration of idle mode. The contents of the SFRs and RAM are also retained. The status of the port pins depends upon the location of the program memory:

- Internal program memory: the ALE and PSEN# pins are pulled high and the ports 0, 1, 2, and 3 pins are driving the port SFR value (Table 14-1).
- External program memory: the ALE and PSEN# pins are pulled high; the port 0 pins are floating; and the pins of parallel ports 1 and 3 are driving the port SFR value (Table 14-1); port 2 pins are weakly pulled high.

14.3.1 Entering Idle Mode

To enter idle mode, set the PCON register IDL bit. The 8*x*931 enters idle mode upon execution of the instruction that sets the IDL bit. The instruction that sets the IDL bit is the last instruction executed.

CAUTION

If the IDL bit and the PD bit are set simultaneously, the 8*x*931 enters powerdown mode.

14.3.2 Exiting Idle Mode

There are two ways to exit idle mode:

- Generate an enabled interrupt. Hardware clears the PCON register IDL bit which restores the clocks to the CPU. Execution resumes with the interrupt service routine. Upon completion of the interrupt service routine, program execution resumes with the instruction immediately following the instruction that activated idle mode. The general purpose flags (GF1 and GF0 in the PCON register) may be used to indicate whether an interrupt occurred during normal operation or during idle mode. When idle mode is exited by an interrupt, the interrupt service routine may examine GF1 and GF0.
- Reset the chip. A logic high on the RST pin clears the IDL bit in the PCON register directly and asynchronously. This restores the clocks to the CPU. Program execution momentarily resumes with the instruction immediately following the instruction that activated the idle mode and may continue for a number of clock cycles before the internal reset algorithm takes control. Reset initializes the 8*x*931 and vectors the CPU to address 0000H.

NOTE

During the time that execution resumes, the internal RAM cannot be accessed; however, it is possible for the port pins to be accessed. To avoid unexpected outputs at the port pins, the instruction immediately following the instruction that activated idle mode should not write to a port pin or to the external RAM.

14.4 USB POWER CONTROL

The 8*x*931 supports USB power control through firmware, including global suspend/resume and remote wake-up. For flow charts of these operations, see Figure 14-3 on page 14-11.

14.4.1 Global Suspend Mode

When a global suspend is detected by the 8*x*931, the global suspend bit (GSUS in PCON1) is set and the global suspend/resume interrupt is generated. Global suspend is defined as bus inactivity for more than 3 ms on the USB root port. A device that is already in suspend mode will not change state. Hardware does not invoke any particular power-saving mode on detection of a global suspend. You must implement power control through firmware within the global suspend/resume ISR.

NOTE

Firmware must set the PD bit (PCON.1 in Figure 14-1 on page 14-3).

For global suspend on a bus powered device, firmware must put the 8*x*931 into powerdown mode to meet the USB limit of 500 µA. For self-powered devices, there is no hard requirement to put the 8*x*931 into powerdown mode. To reduce power consumption, idle mode and low clock mode can be used instead.

14.4.1.1 Powerdown (Suspend) Mode

The powerdown mode places the 8*x*931 in a very low power state. Powerdown mode stops the oscillator and freezes all clocks at known states (Figure 2-4 on page 2-7). The CPU status prior to entering powerdown mode is preserved (i.e., the program counter, program status word register, and register file retain their data for the duration of powerdown mode). In addition, the SFRs and RAM contents are preserved. The status of the port pins depends on the location of the program memory:

- Internal program memory: the ALE and PSEN# pins are pulled low and the pins of parallel ports 0 - 3 are driving the port SFR value (Table 14-1 on page 14-6).
- External program memory: the ALE and PSEN# pins are pulled low; the port 0 pins are floating; and the pins of ports 1 and 3 are reading data, and Port 2 pins are weakly pulled high. (Table 14-1).

NOTE

V_{CC} may be reduced to as low as 2 V during powerdown to further reduce power dissipation. Take care, however, that V_{CC} is not reduced until powerdown is invoked.

14.4.1.2 Entering Powerdown (Suspend) Mode

To enter powerdown mode, set the PCON register PD bit. The 8*x*931 enters powerdown mode upon execution of the instruction that sets the PD bit. The instruction that sets the PD bit is the last instruction executed.

CAUTION

Do not put the 8*x*931 into powerdown mode unless the USB suspend signal is detected on the USB lines (GSUS = 1). Otherwise, the device will not be able to wake up from powerdown mode by a resume signal sent through the USB lines. See "USB Power Control" on page 14-7.

14.4.1.3 Exiting Powerdown (Suspend) Mode

CAUTION

If V_{CC} was reduced during the powerdown mode, do not exit powerdown until V_{CC} is restored to the normal operating level.

There are two ways (other than USB signaling) to exit the powerdown mode:

1. Generate an enabled external interrupt (including the keyboard scan interrupt). The interrupt signal must be held active long enough for the oscillator to restart and stabilize (normally less than 10 ms). Hardware clears the PD bit in the PCON register which starts the oscillator and restores the clocks to the CPU and peripherals. Execution resumes with the interrupt service routine. Upon completion of the interrupt service routine, program execution resumes with the instruction immediately following the instruction that activated powerdown mode.

To enable an external interrupt, set the IEN0 register EX0 and/or EX1 bit[s]. The external interrupt used to exit powerdown mode must be configured as level-sensitive and must be assigned the highest priority. Holding the interrupt pin (INT0#, INT1#, or the keyboard scan interrupt INT2#) low restarts the oscillator and bringing the pin high completes the exit. The duration of the interrupt signal must be long enough to allow the oscillator to stabilize (normally less than 10 ms).

2. Generate a reset. A logic high on the RST pin clears the PD bit in the PCON register directly and asynchronously. This starts the oscillator and restores the clocks to the CPU and peripherals. Program execution momentarily resumes with the instruction immediately following the instruction that activated powerdown and may continue for a number of clock cycles before the internal reset algorithm takes control. Reset initializes the 8*x*931 and vectors the CPU to address 0000H.

NOTE

During the time that execution resumes, the internal RAM cannot be accessed; however, it is possible for the port pins to be accessed. To avoid unexpected outputs at the port pins, the instruction immediately following the instruction that activated the powerdown mode should not write to a port pin or to the external RAM.

14.4.2 Global Resume Mode

Upon detection of a global resume, the 8*x*931 sets the global resume bit (GRSM of PCON1), clears the global suspend bit (GSUS of PCON1), and generates the global suspend/resume interrupt. The 8*x*931 restarts the oscillator as soon as resume signaling is detected on the USB lines. A resume condition is defined as a “J to anything” transition. This could be a K transition, or reset signaling on the root port. A resume condition could be an enabled downstream port or connect/disconnect of a downstream port in the disconnected, disabled, or suspended states.

NOTE

Since the 8*x*931AA microcontroller does not support a hub interface, there are no downstream ports to signal a resume condition. A resume condition can still be caused by any of the other conditions mentioned above, however. Specific details of the 8*x*931AA are covered in Appendix E, “8x931AA Design Considerations”.

Upon detection of a resume condition, the 8*x*931 applies power to the USB transceivers, the crystal oscillator, and the PLL (although the PLL output is still gated-off). The device begins timing two different time points, T1 and T2, as described in Chapter 11 of the *Universal Serial Bus Specification*.

After the clocks are restarted, the CPU program continues execution from where it was when the device was put into powerdown mode. The device then services the resume interrupt service routine. After executing the resume ISR, the 8*x*931 continues operation from the point where it was interrupted by the suspend interrupt.

14.4.3 USB Remote Wake-up

The 8*x*931 can initiate resume signaling to the USB lines through remote wake-up of the USB function while it is in powerdown mode. While in powerdown mode, remote wake-up has to be initiated through assertion of an enabled external interrupt. The external interrupt has to be enabled and it must be configured with level trigger and with higher priority than a suspend/resume interrupt. A function resume restarts the clocks to the 8*x*931 and program execution branches to an external interrupt service routine.

Within this external interrupt service routine, you must ensure GRSM = 0. If GRSM is clear, set the remote wake-up bit (RWU in PCON1 — Figure 14-2) to drive resume signaling on the USB lines to the host or upstream hub and to the enabled downstream ports). After executing the external ISR, the program continues execution from where it was put into powerdown mode and the 8*x*931 resumes normal operation.

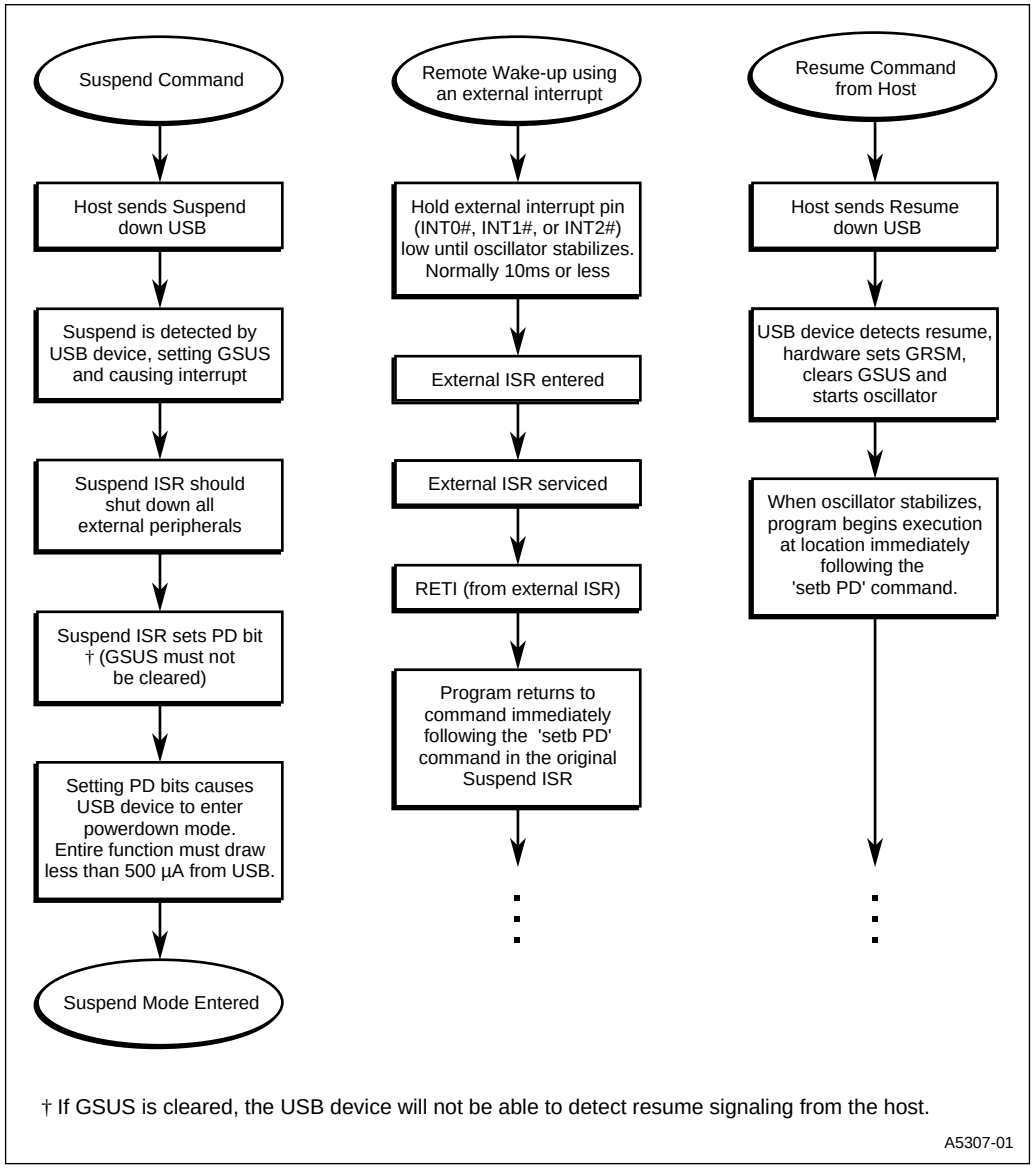


Figure 14-3. Suspend/Resume Program with/without Remote Wake-up

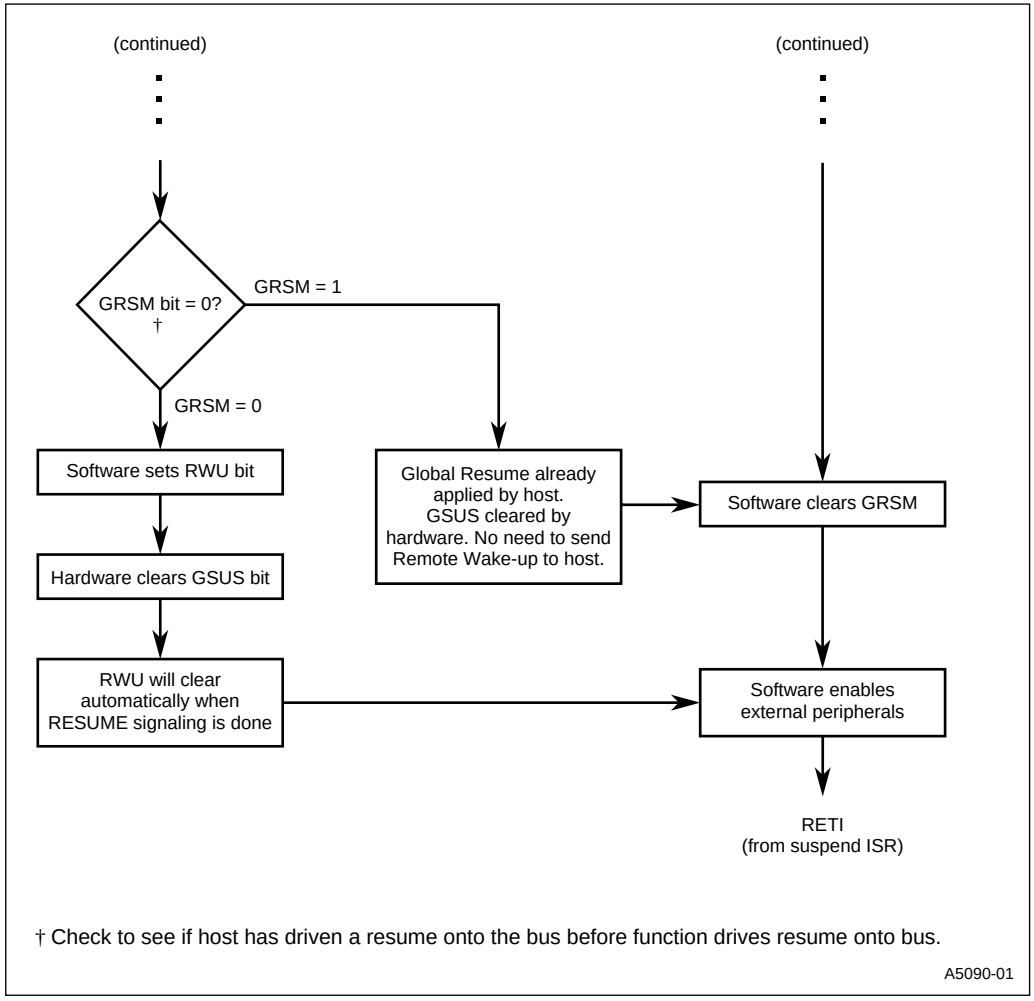


Figure 14-4. Suspend/Resume Program with/without Remote Wake-up (Continued)

14.5 LOW-CLOCK MODE

Low-clock mode is the default operation mode for the 8*x*931 upon reset. After reset, the CPU and peripherals (excluding the USB module) default to a 3 MHz clock rate. The USB module always runs at full speed. Low-clock mode ensures that the I_{CC} drawn by the 8*x*931, while in the unenumerated state following chip reset, is less than one unit load (100 mA).

After USB enumeration (and given that the request for more than one unit load of I_{CC} is granted), firmware can clear the LC bit in PCON to clock the CPU and on-chip peripherals at the normal rate.

14.5.1 Entering Low-clock Mode

Low-clock mode can be invoked through firmware anytime the device is unconfigured by the host PC. To invoke low-clock mode, set the LC bit in the PCON register (Figure 14-1).

NOTE

The device reset routine sets the LC bit placing the 8*x*931 in low-clock mode.

14.5.2 Exiting Low-clock Mode

To switch the clock of the CPU and the peripherals to the hardware-selected clock rate, clear the LC bit in the PCON register (Figure 14-1). The hardware clock rate selection determines the highest operating clock rate for the 8*x*931.

14.6 ON-CIRCUIT EMULATION (ONCE) MODE

The on-circuit emulation (ONCE) mode permits external testers to test and debug 8*x*931-based systems without removing the chip from the circuit board. A clamp-on emulator or test CPU is used in place of the 8*x*931 which is electrically isolated from the system.

14.6.1 Entering ONCE Mode

To enter the ONCE mode:

1. Assert RST to initiate a device reset.
2. While holding RST asserted, apply and hold logic levels to I/O pins as follows: PSEN# = high, ALE = low, and EA# = high.
3. Deassert RST, then remove the logic levels from PSEN# and ALE.

These actions cause the 8*x*931 to enter the ONCE mode. The pins of parallel ports 0 - 3, ALE, and PSEN# pins are floating (Table 14-1 on page 14-6). Thus the device is electrically isolated from the remainder of the system which can then be tested by an emulator or test CPU. Note that in the ONCE mode the device oscillator remains active.

14.6.2 Exiting ONCE Mode

To exit ONCE mode, reset the device.

15

External Memory Interface

CHAPTER 15
EXTERNAL MEMORY INTERFACE

This chapter covers various aspects of the external memory interface. It describes the signals associated with external memory operations and external bus cycle timing. This chapter also gives the status of the pins for ports P0 and P2 during bus cycles and bus idle, and includes several external memory design examples.

15.1 OVERVIEW

The 8*x*931 interfaces with a variety of external memory devices. Data transfer operations (8 bits) are multiplexed on the lower address bits (A7:0).

The external memory interface comprises the external bus (ports 0 and 2) and the bus control signals described in Table 15-1. Figure 15-1 shows the structure of the external address bus.

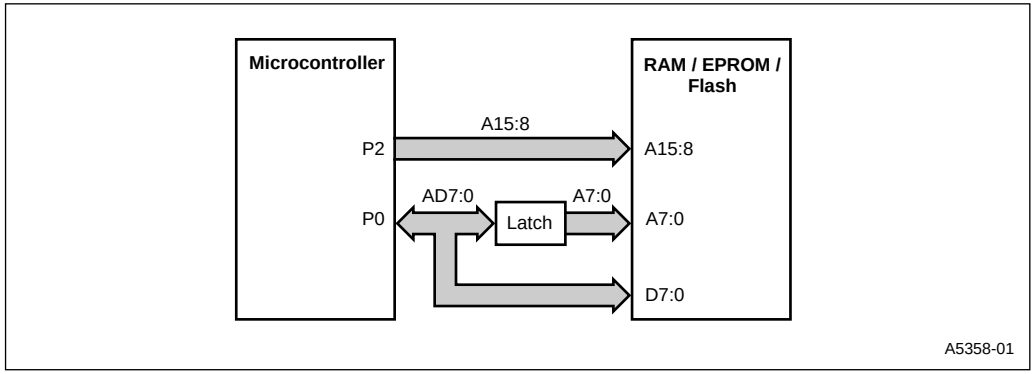


Figure 15-1. Bus Structure

Table 15-1. External Memory Interface Signals

Signal Name	Type	Description	Alternate Function
A15:8	O	**Address Lines**. Upper byte of external memory address.	P2.7:0/KSO15:8
AD7:0	I/O	**Address/Data Lines**. Lower byte of external memory address multiplexed with data	P0.7:0/KSI7:0
ALE	O	**Address Latch Enable**. ALE signals the start of an external bus cycle and indicates that valid address information is available on lines A15:8 and AD7:0. An external latch can use ALE to demultiplex the address from the address/data bus (AD7:0).	—
EA#	I	**External Access**. Directs program memory accesses to on-chip or off-chip code memory. For EA# strapped to ground, all program memory accesses are off-chip. For EA# strapped to V_{CC}, program accesses on-chip ROM if the address is within the range of the on-chip ROM; otherwise the access is off-chip. The value of EA# is latched at reset. For devices without on-chip ROM, EA# must be strapped to ground.	—
PSEN#	O	**Program Store Enable**. Read signal output. Asserted for read accesses to external program memory.	—
RD#	O	**Read.** Read signal output. Asserted for read accesses to external data memory.	P3.7/KSO19
WR#	O	**Write.** Write signal output to external memory.	P3.6/KSO18

15.2 EXTERNAL BUS CYCLES

This section describes the bus cycles the 8*x*931 executes to fetch code, read data, and write data in external memory.

NOTE

For simplicity, the accompanying figures depict the bus cycle waveforms in idealized form and do not provide precise timing information. For bus cycle timing parameters refer to the 8*x*931AA, 8*x*931HA datasheet (order number: 273108-01).

An "inactive external bus" exists when the 8*x*931 is not executing external bus cycles. This occurs under any of the three following conditions:

- Bus Idle (The chip is in normal operating mode but no external bus cycles are executing.)
- The chip is in idle mode
- The chip is in powerdown mode

15.2.1 Bus Cycle Definitions

There are three types of bus cycles: code fetch, data read, and data write. The external bus structure is the same as for MCS® 51 microcontrollers. The upper address bits (A15:8) are on port 2, and the lower address bits (A7:0) are multiplexed with the data (D7:0) on port 0.

Normally, two program fetches are generated during each machine cycle, even if the instruction being executed doesn't need more code bytes, the CPU simply ignores the extra fetch and the Program Counter is not incremented.

If the Program Memory is external, then the Program Memory read strobe PSEN# is normally activated twice per machine cycle. If access to external Data Memory occurs, two PSEN#'s are skipped. This is a result of the address bus and data bus being used for Data Memory access.

NOTE

A Data Memory bus cycle takes twice as much time as a Program Memory bus cycle.

When the CPU is executing from internal Program Memory, PSEN# is not activated, and program addresses are not emitted. However, ALE continues to be activated twice per machine cycle and is available as a clock output signal. Note that one ALE is skipped during the execution of the MOVX instruction.

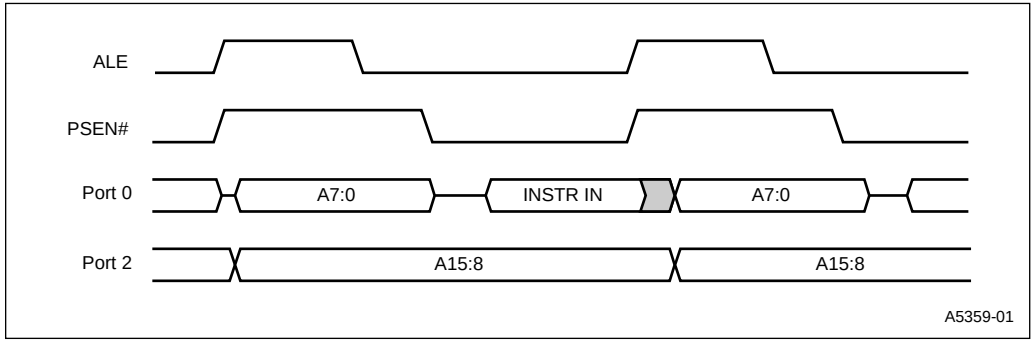


Figure 15-2. External Code Fetch

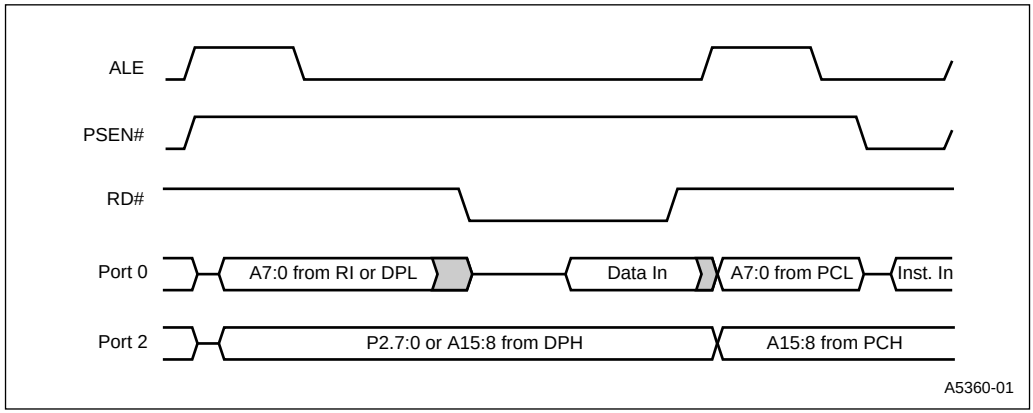


Figure 15-3. External Data Read

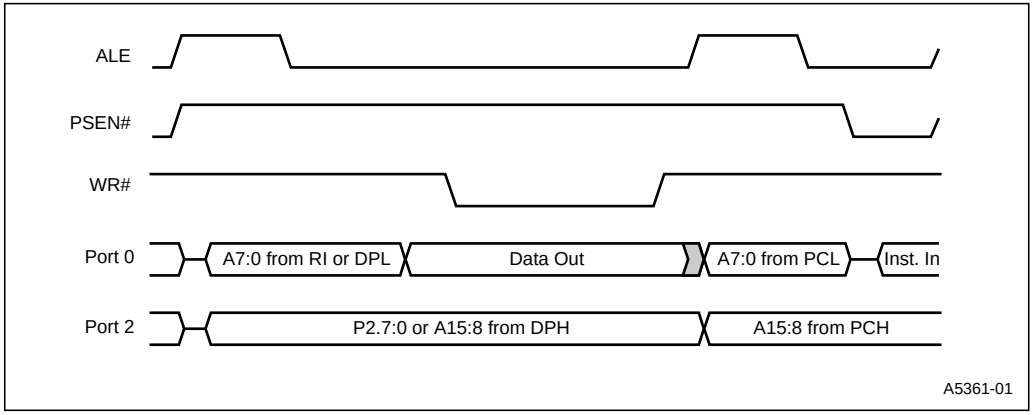


Figure 15-4. External Data Write

15.3 PORT 0 AND PORT 2 STATUS

This section summarizes the status of the port 0 and port 2 pins when these ports are used as the external bus. A more comprehensive description of the ports and their use is given in Chapter 9, “Input/Output Ports”.

When port 0 and port 2 are used as the external memory bus, the signals on the port pins can originate from three sources:

- the 8*x*931 CPU (address bits, data bits)
- the port SFRs: P0 and P2 (logic levels)
- an external device (data bits)

The port 0 pins (but not the port 2 pins) can also be held in a high-impedance state. Table 15-2 lists the status of the port 0 and port 2 pins when the chip in is the normal operating mode and the external bus is idle or executing a bus cycle.

Table 15-2. Port 0 and Port 2 Pin Status In Normal Operating Mode

Port	8-bit/16-bit Addressing	Bus Cycle	Bus Idle
Port 0	8 or 16	AD7:0 (1)	High Impedance
Port 2	8	P2 (2)	P2
	16	A15:8	P2

NOTES:
1. During external memory accesses, the CPU writes FFH to the P0 register and the register contents are lost.
2. The P2 register can be used to select 256-byte pages in external memory.

15.3.1 Port 0 and Port 2 Pin Status

The port pins have the same signals as those on the 8XC51FX. For an external memory instruction using a 16-bit address, the port pins carry address and data bits during the bus cycle. However, if the instruction uses an 8-bit address (e.g., MOVX @Ri), the contents of P2 are driven onto the pins. These pin signals can be used to select 256-bit pages in external memory.

During a bus cycle, the CPU always writes FFH to P0, and the former contents of P0 are lost. A bus cycle does not change the contents of P2. When the bus is idle, the port 0 pins are held at high impedance, and the contents of P2 are driven onto the port 2 pins.

15.4 EXTERNAL MEMORY DESIGN EXAMPLES

This section presents several external memory designs for 8*x*931 systems. Many designs are possible. The examples apply for both 8*x*931AA and 8*x*931HA devices.

15.4.1 Example 1: 11-bit Bus, External RAM

Figure 15-5 shows a hardware configuration for accessing up to 2K bytes of external RAM. The CPU in this case is executing from internal ROM. Port 0 serves as a multiplexed address/data bus to the RAM, 3 lines of port 2 are being used to page the RAM. If the Program Memory is internal, the other bits of P2 are available as input/output. The CPU generates RD# and WR# signals as needed during external RAM accesses.

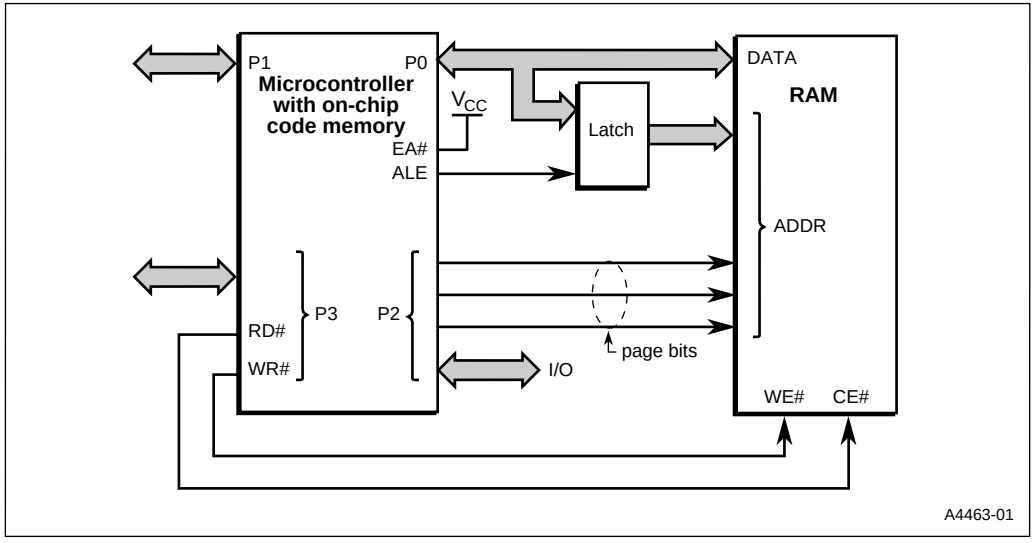


Figure 15-5. Bus Diagram for Example 1: 8*x*931AA/HA

15.4.2 Example 2: 16-bit Bus, External ROM

The hardware configuration for external program execution is shown in Figure 15-6 below. Note that the 16 I/O lines (ports 0 and 2) are dedicated to bus functions during external Program Memory fetches. Port 0 (P0 in Figure 15-6) serves as a multiplexed address/data bus. It emits the low byte of the Program Counter (PCL) as an address, and then goes into a float state awaiting the arrival of the code byte from the Program Memory. During the time that the low byte of the Program Counter is valid on P0, the signal ALE (Address Latch Enable) clocks this byte into an address latch. Meanwhile, port 2 (P2 in Figure 15-6) emits the high byte of the Program Counter (PCH). Then PSEN# strobes the EPROM and the code byte is read into the microcontroller.

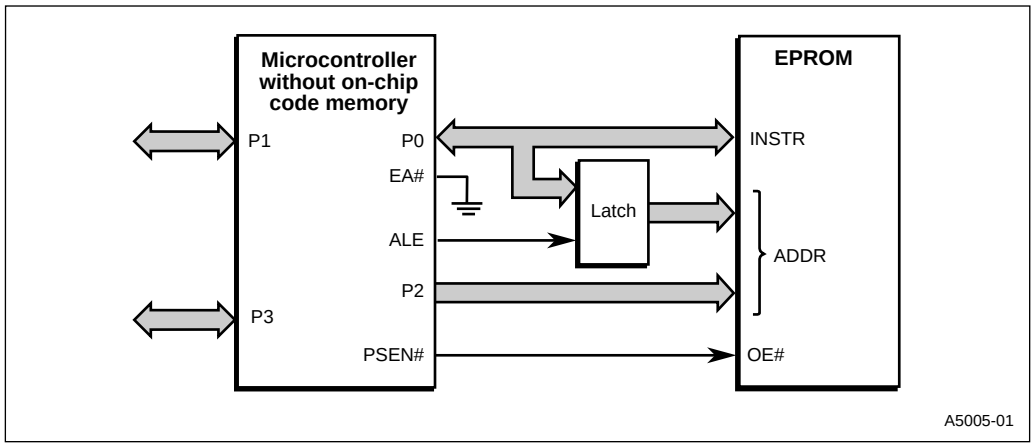


Figure 15-6. Bus Diagram for Example 2: 8*x*931AA/HA

15.4.3 Example 3: 16-bit Bus, External EPROM and RAM

In this example, an 8*x*931AA/HA operates with a 16-bit external address bus interfaced to 64 Kbytes of EPROM and 64 Kbytes of RAM (Figure 15-7). The 8*x*931AA/HA will assert RD# and WR# signals as needed during external RAM accesses. The read strobe to external EPROM, PSEN#, is used for external program fetches.

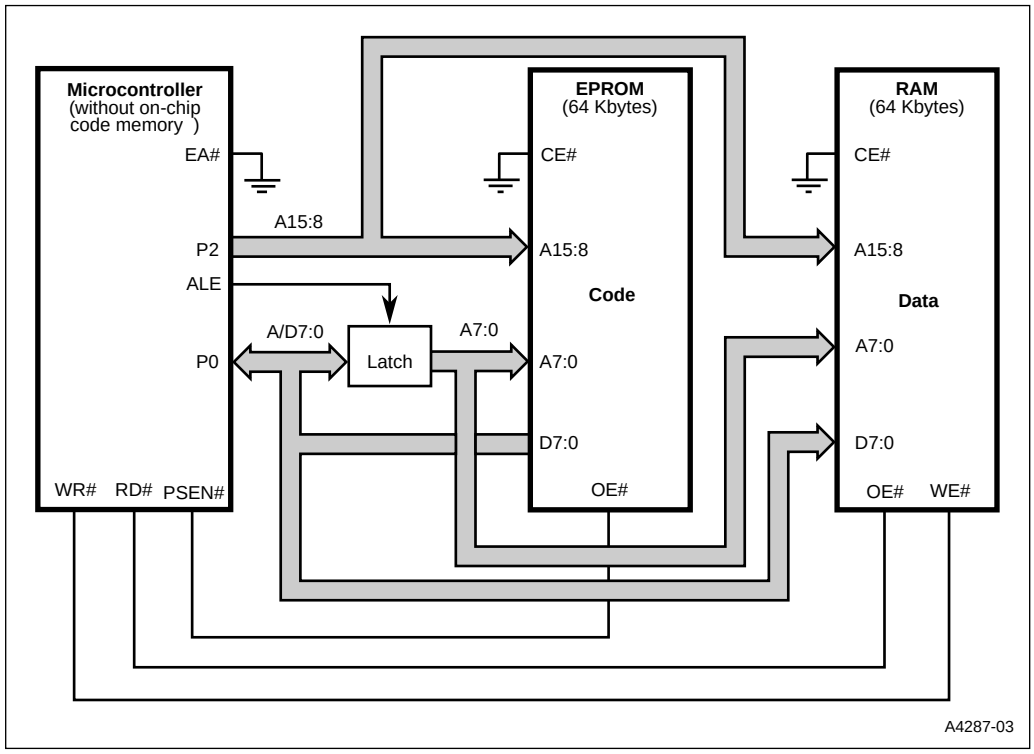


Figure 15-7. Bus Diagram for Example 3: 8*x*931AA/HA

16

Verifying Nonvolatile Memory

CHAPTER 16
VERIFYING NONVOLATILE MEMORY

This chapter discusses the 83931 on-chip memory and provides the procedure for verifying on-chip nonvolatile memory.

16.1 83931 MEMORY

The MCS® 51 architecture provides separate 64-Kbyte address spaces for program memory and data memory (see "8x931 Memory" on page 2-11). Table 2-1 on page 2-3 lists the available on-chip ROM and RAM memory options for the 83931.

16.2 NONVOLATILE MEMORY

For ROM devices (83931), on-chip program memory is located at the lowest addresses of the program memory address space. ROM devices also make provision for storing signature bytes, an encryption array, and lock bits in on-chip nonvolatile memory outside the program and data memory address spaces.

In some applications, it is desirable that program code be secure from unauthorized access. The 83931 offers two types of protection for on-chip program code. On-chip program code is encrypted when read out for verification if the encryption array is programmed. Lock bits restrict external access to on-chip program memory.

16.3 VERIFYING ON-CHIP NONVOLATILE MEMORY

This section provides instructions for verifying the contents of the following nonvolatile memory functions on the 83931 ROM device:

- on-chip program memory (8 Kbytes)
- lock bits (3 bits)
- signature bytes (3 bytes)

The 83931 is verified in the same manner as MCS 51 microcontrollers. Verify operations differ from normal device operation. Verify operations are performed with the device installed in a ROM or EPROM programmer. The CPU does not execute instructions. Memory accesses are made one byte at a time using addresses externally applied to ports P3:4-5, P2:0-5 and P1. See Table 16-2 on page 16-3 for pin usage during verify operations. For a complete list of device signal descriptions, see Appendix B.

To preserve the security of on-chip program code, the encryption array cannot be verified.

Table 16-1. Signal Descriptions (Verify Mode)

Signal Name	Type	Description	Alternate Function
P0.7:0	O	**Port 0**. Eight-bit, open-drain, bidirectional I/O port. For verify operations, use as the data port. See Table 16-2 and Figure 16-1.	—
P1.7:0	I	**Port 1**. Eight-bit, bidirectional I/O port with internal pullups. For verify operations, use for low byte of address. See Table 16-2 and Figure 16-1.	—
P2.7:0	I	**Port 2**. Eight-bit, bidirectional I/O port with internal pullups. For verify operations, use P2.5:0 as A8-13 and P2.6 and P2.7 to ground. See Table 16-2 and Figure 16-1.	—
P3.7:0	I	**Port 3**. Eight-bit, bidirectional I/O port with internal pullups. For verify operations, use P3.4 and P3.5 as A14 and A15, P3.3 to ground. For P3.6 and P3.7, see Table 16-2 and Figure 16-1.	—
ALE	I	**Address Latch Enable.** For verify operations, connect this pin to V_{CC}	—
EA#	I	**External Enable.** For verify operations, connect this pin to $V_{CC.}$	—
PSEN#	I	**Program Store Enable**. For verify operations, connect this pin to V_{SS}	—
RST	I	Reset. For verify operations, connect this pin to $V_{CC.}$	—

16.3.1 Verify Modes

Table 16-2 lists the verify modes and provides details about the setup. The encryption array, lock bits, and signature bytes reside in nonvolatile memory outside the program and data memory address spaces.

16.3.2 General Setup

Figure 16-1 shows the general setup for verifying nonvolatile memory on the 83931. The controller must be running with an oscillator frequency of 4 MHz to 6 MHz. Set up the controller as shown in Table 16-2 to verify on-chip program memory, signature bytes and lockbits. Data appears on port 0. Connect RST, ALE, and EA# to V_{CC} and PSEN# to ground.

Table 16-2. Verify Modes

Verify Modes	RST	PSEN#	EA#	ALE	P3[7:3]	P2[7:0]	P1[7:0]
Verify On-chip Program Memory	1	0	1	1	11AA0	00AAAAAA	AAAAAAAA
Verify Signature Bytes	1	0	1	1	00 . . 0	00.	AAAAAAAA
Verify Lock bit	1	0	1	1	10 . . 0	00.	. . . . .

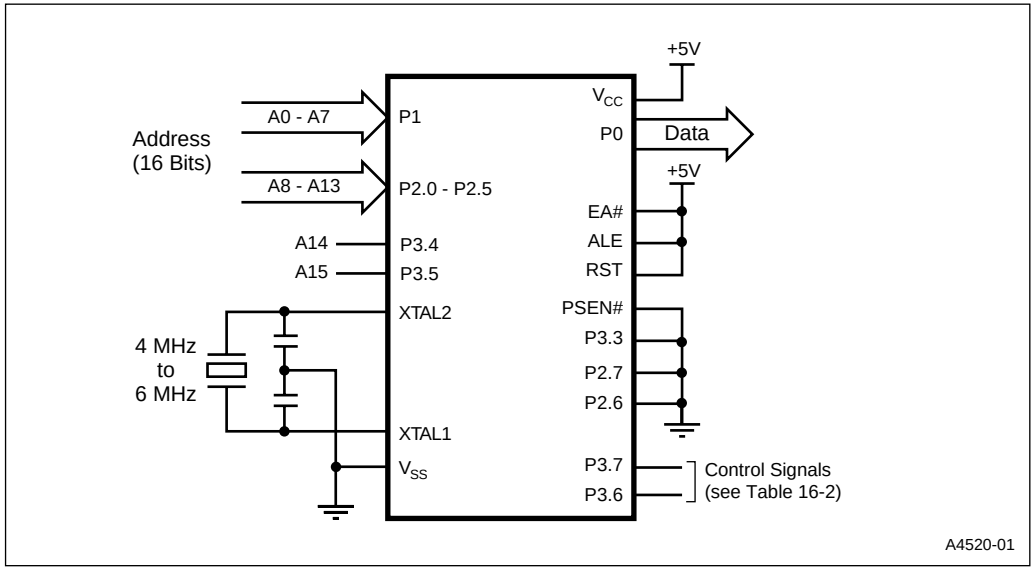


Figure 16-1. Setup for Verifying Nonvolatile Memory

16.3.3 Verify Algorithm

Use this procedure to verify program code, signature bytes, and lock bits stored in nonvolatile memory on the 83931. To preserve the secrecy of the encryption key byte sequence, the encryption array cannot be verified. Verification can be performed on a block of bytes. The procedure for verifying the 83931 is as follows:

1. Set up the microcontroller for operation in the appropriate mode according to Table 16-2.
2. Input the 16-bit address on ports P1 and P2.0 - P2.5 and P3.4:5.
3. Wait for the data on port P0 to become valid (T_{AVQV} = 48 clock cycles), then compare the data with the expected value.
4. Repeat steps 1 through 3 until all memory locations are verified.

16.3.4 Verifying On-chip Program Memory

To verify that on-chip program memory is correctly programmed, perform the procedure described in "Verify Algorithm" on page 16-3 using the verify on-chip program memory mode (Table 16-2). For information about using on-chip program memory, see "Considerations for On-chip Program Code Memory" on page 16-5.

16.3.5 Verifying the Lock Bits

The 8*x*931 provides lock bits for protecting program code stored in the on-chip program memory from unauthorized access. To verify that the lock bits are correctly programmed, perform the procedure described in "Verify Algorithm" on page 16-3 using the verify lock bit mode (Table 16-2).

Table 16-3. Lock Bit Function

Program Lock Bits				Protection Type
	LB1	LB2	LB3	
1	U	U	U	No program lock features enabled. (Code verify will still be encrypted by the encryption array if programmed.)
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, EA is sampled and latched upon reset.
3	P	P	U	Same as 2, verify disabled.
4	P	P	P	Same as 3, external execution is disabled.

P = Programmed
U = Unprogrammed
Any other combination of the Lock Bits is undefined.

16.3.6 Verifying the Signature Bytes

The 83931 contains factory-programmed signature bytes. These bytes are located in nonvolatile memory outside the program and data memory address space at 30H, 31H, 60H. To read the signature bytes, perform the procedure described in "Verify Algorithm" on page 16-3 using the verify signature mode (Table 16-2). Signature byte values are listed in Table 16-4.

Table 16-4. Contents of the Signature Bytes

ADDRESS	CONTENTS	DEVICE TYPE
30H	89H	Indicates Intel devices
31H	59H	Indicates 51F***x*** + USB core product
60H	1AH	Indicates 83931AA/HA device

16.4 ENCRYPTION ARRAY

The 83931 includes a 64-byte encryption array located in nonvolatile memory outside the program and data memory address spaces. To preserve the secrecy of the encryption key byte sequence, the encryption array cannot be verified.

Program code verification is performed as usual, except as each byte of program code is read, it is exclusive-NORed (XNOR) with the corresponding key byte from the encryption array. If the encryption array is programmed with key bytes, the program code is encrypted during verification and can not be used without knowledge of the key byte sequence. If the encryption array is not programmed (still all 1s), the program code is placed on the data bus in its original, unencrypted form.

CAUTION

If the encryption feature is implemented, the portion of the on-chip program code memory that does not contain program code should be filled with "random" byte values other than FFH to prevent the encryption key sequence from being revealed.

16.5 CONSIDERATIONS FOR ON-CHIP PROGRAM CODE MEMORY

On-chip, nonvolatile code memory is located at the lowest addresses of program memory address space. The first instruction following device reset is fetched from 0000H. It is recommended that user program code start at address 0100H. Use a jump instruction to 0100H to begin execution of the program. For information on address spaces, see "8x931 Memory" on page 2-11.

Addresses outside the range of on-chip code memory access external memory. With EA# = 1 and both on-chip and external code memory implemented, you can place program code at the highest on-chip memory addresses. When the highest on-chip address is exceeded during execution, program code fetches automatically rollover from on-chip memory to external memory.

With EA# = 1 and only on-chip program code memory, multi-byte instructions and instructions that result in call returns or prefetches should be located a few bytes below the maximum address to avoid inadvertently exceeding the top address.

CAUTION

Execution of program code located in the top few bytes of the on-chip memory may cause prefetches from the next higher addresses (i.e., external memory). External memory fetches make use of port 0 and port 2 and may disrupt program execution if the program uses port 0 or port 2 for a different purpose.

A

Instruction Set Reference

APPENDIX A
INSTRUCTION SET REFERENCE

This appendix contains reference material for the 8*x*931 instruction set, which is identical to instruction set for the MCS® 51 architecture. The appendix includes an opcode map, a detailed description of each instruction, and the following tables that summarize notation, addressing, instructions types, instruction lengths and execution times:

- Tables A-1 through A-4 describe the notation used for the instruction operands. Table A-5 describes the notation used for control instruction destinations.
- Table A-6 on page A-3 contains the opcode map for the instruction set.
- The following tables list the instructions giving length (in bytes) and execution time:

  Add and Subtract Instructions, Table A-7 on page A-4

  Increment and Decrement Instructions, Table A-8 on page A-4

  Multiply, Divide, and Decimal-adjust Instructions, Table A-9 on page A-5

  Logical Instructions, Table A-10 on page A-5

  Move Instructions, Table A-11 on page A-6

  Exchange, Push, and Pop Instructions, Table A-12 on page A-7

  Bit Instructions, Table A-13 on page A-7

  Control Instructions, Table A-14 on page A-8

"Instruction Descriptions" on page A-9 contains a detailed description of each instruction.

A.1 NOTATION FOR INSTRUCTION OPERANDS

Table A-1. Notation for Register Operands

Register Notation	
@Ri	8-bit internal data RAM location (00H–FFH) addressed indirectly via byte register R0 or R1
Rn	Byte register R0–R7 of the currently selected register bank
n	Byte register index: n = 0–7
r r r	Binary representation of n

Table A-2. Notation for Direct Addresses

Direct Address.	Description
dir8	An 8-bit internal data address. This can be internal data RAM (00H–7FH) or an SFR address (80H - FFH).

Table A-3. Notation for Immediate Addressing

Immediate Data	Description
#data	An 8-bit constant that is immediately addressed in an instruction.
#data16	A 16-bit constant that is immediately addressed in an instruction.

Table A-4. Notation for Bit Addressing

Bit Address	Description
bit	A directly addressed bit (bit number = 00H–FFH) in internal data RAM or an SFR. Bits 00H–7FH are the 128 bits in byte locations 20H–2FH in the on-chip RAM. Bits 80H–FFH are the 128 bits in the 16 SFR's with addresses that end in 0H or 8H: 80H, 88H, 90H, . . . , F0H, F8H.

Table A-5. Notation for Destinations in Control Instructions

Destination Address	Description
rel	A signed (two's complement) 8-bit relative address. The destination is -128 to +127 bytes relative to first byte of the next instruction.
addr11	An 11-bit destination address. The destination is in the same 2-Kbyte block of memory as the first byte of the next instruction.
addr16	A 16-bit destination address. A destination can be anywhere within the same 64-Kbyte region as the first byte of the next instruction.

A.2 OPCODE MAP

Table A-6. Instructions for 8*x*931 Peripheral Controllers

Bin	0	1	2	3	4	5	6-7	8-F
0	NOP	AJMP addr11	LJMP addr16	RR A	INC A	INC dir8	INC @Ri	INC Rn
1	JBC bit,rel	ACALL addr11	LCALL addr16	RRC A	DEC A	DEC dir8	DEC @Ri	DEC Rn
2	JB bit,rel	AJMP addr11	RET	RL A	ADD A,#data	ADD A,dir8	ADD A,@Ri	ADD A,Rn
3	JNB bit,rel	ACALL addr11	RETI	RLC A	ADDC A,#data	ADDC A,dir8	ADDC A,@Ri	ADDC A,Rn
4	JC rel	AJMP addr11	ORL dir8,A	ORL dir8,#data	ORL A,#data	ORL A,dir8	ORL A,@Ri	ORL A,Rn
5	JNC rel	ACALL addr11	ANL dir8,A	ANL dir8,#data	ANL A,#data	ANL A,dir8	ANL A,@Ri	ANL A,Rn
6	JZ rel	AJMP addr11	XRL dir8,A	XRL dir8,#data	XRL A,#data	XRL A,dir8	XRL A,@Ri	XRL A,Rn
7	JNZ rel	ACALL addr11	ORL CY,bit	JMP @A+DPTR	MOV A,#data	MOV dir8, #data	MOV @Ri,#data	MOV Rn,#data
8	SJMP rel	AJMP addr11	ANL CY,bit	MOVC A,@A+PC	DIV AB	MOV dir8,dir8	MOV dir8,@Ri	MOV dir8,Rn
9	MOV DPTR, #data16	ACALL addr11	MOV bit,CY	MOVC A,@A+DPTR	SUBB A,#data	SUBB A,dir8	SUBB A,@Ri	SUBB A,Rn
A	ORL CY,/bit	AJMP addr11	MOV CY,bit	INC DPTR	MUL AB	Reserved	MOV @Ri,dir8	MOV Rn,dir8
B	ANL CY,/bit	ACALL addr11	CPL bit	CPL CY	CJNE A,#data,rel	CJNE A,dir8,rel	CJNE @Ri,#data, rel	CJNE Rn,#data, rel
C	PUSH dir8	AJMP addr11	CLR bit	CLR CY	SWAP A	XCH A,dir8	XCH A,@Ri	XCH A,Rn
D	POP dir8	ACALL addr11	SETB bit	SETB CY	DA A	DJNZ dir8,rel	XCHD A,@Ri	DJNZ Rn,rel
E	MOVX A,@DPTR	AJMP addr11	MOVX A,@Ri		CLR A	MOV A,dir8	MOV A,@Ri	MOV A,Rn
F	MOVX @DPTR,A	ACALL addr11	MOVX @Ri,A		CPL A	MOV dir8,A	MOV @Ri,A	MOV Rn,A

A.3 INSTRUCTION SET SUMMARY

This section contains tables that summarize the instruction set. For each instruction there is a short description, its length in bytes, and its execution time in states and machine cycles.

A.3.1 Instruction Summaries

Table A-7. Summary of Add and Subtract Instructions

Add	**ADD <dest>,<src>**	**dest opnd ← dest opnd + src opnd**
Add with Carry	**ADDC <dest>,<src>**	**(A) ← (A) + src opnd + carry bit**
Subtract with Borrow	**SUBB <dest>,<src>**	**(A) ← (A) - src opnd - carry bit**

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
ADD	A,Rn	Reg to acc	1	6	1
	A,dir8	Dir byte to acc	2	6	1
	A,@Ri	Indir addr to acc	1	6	1
	A,#data	Immediate data to acc	2	6	1
ADDC; SUBB	A,Rn	Reg to/from acc with carry	1	6	1
	A,dir8	Dir byte to/from acc with carry	2	6	1
	A,@Ri	Indir RAM to/from acc with carry	1	6	1
	A,#data	Immediate data to/from acc with carry	2	6	1

Table A-8. Summary of Increment and Decrement Instructions

Increment	**INC DPTR**	**(DPTR) ← (DPTR) + 1**
Increment	**INC byte**	**byte ← byte + 1**
Decrement	**DEC byte**	**byte ← byte – 1**

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
INC; DEC	A	acc	1	6	1
	Rn	Reg	1	6	1
	dir8	Dir byte	2	6	1
	@Ri	Indir RAM	1	6	1
	DPTR	Data pointer	1	12	2

Table A-9. Summary of Multiply, Divide, and Decimal-adjust Instructions

Multiply	**MUL AB**	**(B:A) = AxB**
Divide	**DIV AB**	**(A) = Quotient; (B) = Remainder**
Decimal-adjust ACC for Addition (BCD)	**DA A**	**(1)**

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
MUL	AB	Multiply A and B	1	24	4
DIV	AB	Divide A by B	1	24	4
DA	A	Decimal adjust acc	1	6	1

NOTES:
1. See “Instruction Descriptions” on page A-9.

Table A-10. Summary of Logical Instructions

Logical AND	**ANL <dest>,<src>**	**dest opnd ←dest opnd Λ src opnd**
Logical OR	**ORL <dest>,<src>**	**dest opnd ← dest opnd V src opnd**
Logical Exclusive OR	**XRL <dest>,<src>**	**dest opnd ← dest opnd ∀ src opnd**
Clear	**CLR A**	**(A) ← 0**
Complement	**CPL A**	**(Ai) ← Ø(Ai)**
Rotate	**RXX A**	**(1)**
SWAP	**A**	**A3:0 ↔ A7:4**

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
ANL; ORL; XRL;	A,Rn	Reg to acc	1	6	1
	A,dir8	Dir byte to acc	2	6	1
	A,@Ri	Indir addr to acc	1	6	1
	A,#data	Immediate data to acc	2	6	1
	dir8,A	Acc to dir byte	2	6	1
	dir8,#data	Immediate data to dir byte	3	12	2
CLR	A	Clear acc	1	6	1
CPL	A	Complement acc	1	6	1
RL	A	Rotate acc left	1	6	1
RLC	A	Rotate acc left through the carry	1	6	1
RR	A	Rotate acc right	1	6	1
RRC	A	Rotate acc right through the carry	1	6	1
SWAP	A	Swap nibbles within the acc	1	6	1

NOTES:
1. See “Instruction Descriptions” on page A-9.

Table A-11. Summary of Move Instructions

Move (1)	**MOV <dest>,<src>**	**destination ← src opnd**
Move Code Byte	**MOVC <dest>,<src>**	**A ← code byte**
Move to External Mem	**MOVX <dest>,<src>**	**external mem ← (A)**
Move from External Mem	**MOVX <dest>,<src>**	**A ← source opnd in external mem**

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
MOV	A,Rn	Reg to acc	1	6	1
	A,dir8	Dir byte to acc	2	6	1
	A,@Ri	Indir RAM to acc	1	6	1
	A,#data	Immediate data to acc	2	6	1
	Rn,A	Acc to reg	1	6	1
	Rn,dir8	Dir byte to reg	2	12	2
	Rn,#data	Immediate data to reg	2	6	1
	dir8,A	Acc to dir byte	2	6	1
	dir8,Rn	Reg to dir byte	2	12	2
	dir8,dir8	Dir byte to dir byte	3	12	2
	dir8,@Ri	Indir RAM to dir byte	2	12	2
	dir8,#data	Immediate data to dir byte	3	12	2
	@Ri,A	Acc to indir RAM	1	6	1
	@Ri,dir8	Dir byte to indir RAM	2	12	2
	@Ri,#data	Immediate data to indir RAM	2	6	1
	DPTR,#data16	Load Data Pointer with a 16-bit const	3	12	2
MOVC	A,@A+DPTR	Code byte relative to DPTR to acc	1	12	2
	A,@A+PC	Code byte relative to PC to acc	1	12	2
MOVX	A,@Ri	External mem (8-bit addr) to acc	1	12	2
	A,@DPTR	External mem (16-bit addr) to acc	1	12	2
	@Ri,A	Acc to external mem (8-bit addr)	1	12	2
	@DPTR,A	Acc to external mem (16-bit addr)	1	12	2

NOTES:
1. Instructions that move bits are in Table A-13.

Table A-12. Summary of Exchange, Push, and Pop Instructions

Exchange Contents	**XCH <dest>,<src>**	**A ↔ src opnd**
Exchange Digit	**XCHD <dest>,<src>**	**A3:0 ↔ on-chip RAM bits 3:0**
Push	**PUSH <src>**	**SP ← SP + 1; (SP) ← src**
Pop	**POP <dest>**	**dest ← (SP); SP ← SP – 1**

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
XCH	A,Rn	Acc and reg	1	6	1
	A,dir8	Acc and dir addr	2	6	1
	A,@Ri	Acc and on-chip RAM (8-bit addr)	1	6	1
XCHD	A,@Ri	Acc and low nibble in on-chip RAM (8-bit addr)	1	6	1
PUSH	dir8	Push dir byte onto stack	2	12	2
POP	Dir8	Pop dir byte from stack	2	12	2

Table A-13. Summary of Bit Instructions

Clear Bit	**CLR bit**	**bit ← 0**
Set Bit	**SETB bit**	**bit ← 1**
Complement Bit	**CPL bit**	**bit← Øbit**
AND Carry with Bit	**ANL CY,bit**	**CY ← CY Λ bit**
AND Carry with Complement of Bit	**ANL CY,/bit**	**CY ← CY Λ Øbit**
OR Carry with Bit	**ORL CY,bit**	**CY ← CY V bit**
ORL Carry with Complement of Bit	**ORL CY,/bit**	**CY ← CY V Øbit**
Move Bit to Carry	**MOV CY,bit**	**CY ← bit**
Move Bit from Carry	**MOV bit,CY**	**bit ← CY**

Mnemonic	<src>,<dest>	Notes	Bytes	States	Machine Cycles
CLR	CY	Clear carry	1	6	1
	bit	Clear dir bit	2	6	1
SETB	CY	Set carry	1	6	1
	bit	Set dir bit	2	6	1
CPL	CY	Complement carry	1	6	1
	bit	Complement dir bit	2	6	1
ANL	CY,bit	AND dir bit to carry	2	12	2
ANL	CY,/bit	AND complemented dir bit to carry	2	12	2
ORL	CY,bit	OR dir bit to carry	2	12	2
ORL	CY,/bit	OR complemented dir bit to carry	2	12	2
MOV	CY,bit	Move dir bit to carry	2	12	1
	bit,CY	Move carry to dir bit	2	12	2

Table A-14. Summary of Control Instructions

Mnemonic	<dest>,<src>	Notes	Bytes	States	Machine Cycles
ACALL	addr11	Absolute subroutine call	2	12	2
LCALL	addr16	Long subroutine call	3	12	2
RET		Return from subroutine	1	12	2
RETI		Return from interrupt	1	12	2
AJMP	addr11	Absolute jump	2	12	2
LJMP	addr16	Long jump	3	12	2
SJMP	rel	Short jump (relative addr)	2	12	2
JMP	@A+DPTR	Jump indir relative to the DPTR	1	12	2
JC	rel	Jump if carry is set	2	12	2
JNC	rel	Jump if carry not set	2	12	2
JB	bit,rel	Jump if dir bit is set	3	12	2
JNB	bit,rel	Jump if dir bit is not set	3	12	2
JBC	bit,rel	Jump if dir bit is set & clear bit	3	12	2
JZ	rel	Jump if acc is zero	2	12	2
JNZ	rel	Jump if acc is not zero	2	12	2
CJNE	A,dir8,rel	Compare dir byte to acc and jump if not equal	3	12	2
	A,#data,rel	Compare immediate to acc and jump if not equal	3	12	2
	Rn,#data,rel	Compare immediate to reg and jump if not equal	3	12	2
	@Ri,#data,rel	Compare immediate to indir and jump if not equal	3	12	2
DJNZ	Rn,rel	Decrement reg and jump if not zero	2	12	2
	dir8,rel	Decrement dir byte and jump if not zero	3	12	2
NOP	—	No operation	1	6	1

A.4 INSTRUCTION DESCRIPTIONS

This section describes each instruction in the 8*x*931 architecture.

Table A-15 defines the symbols (—, 3, 1, 0,?) used to indicate the effect of the instruction on the flags in the PSW register. For a conditional jump instruction, "!" indicates that a flag influences the decision to jump.

Table A-15. Flag Symbols

Symbol	Description
—	The instruction does not modify the flag.
3	The instruction sets or clears the flag, as appropriate.
1	The instruction sets the flag.
0	The instruction clears the flag.
?	The instruction leaves the flag in an indeterminate state.
!	For a conditional jump instruction: The state of the flag before the instruction executes influences the decision to jump or not jump.

ACALL <addr11>

Function: Absolute call

Description: Unconditionally calls a subroutine at the specified address. The instruction increments the 3-byte PC twice to obtain the address of the following instruction, then pushes bytes 0 and 1 of the result onto the stack (byte 0 first) and increments the stack pointer twice. The destination address is obtained by successively concatenating bits 15–11 of the incremented PC, opcode bits 7–5, and the second byte of the instruction. The subroutine called must therefore start within the same 2-Kbyte "page" of the program memory as the first byte of the instruction following ACALL.

Flags:

CY	AC	OV
—	—	—

Example: The stack pointer (SP) contains 07H and the label "SUBRTN" is at program memory location 0345H. After executing the instruction

ACALL SUBRTN

at location 0123H, SP contains 09H; on-chip RAM locations 08H and 09H contain 01H and 25H, respectively; and the PC contains 0345H.

Bytes: 2
States: 12
Cycles: 2

[Encoding]

a10 a9 a8 1	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL
(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC.7:0)
(SP) ← (SP) + 1
((SP)) ← (PC.15:8)
(PC.10:0) ← page address

ADD <A>,<src-byte>

Function: Add

Description: Adds the source operand to the accumulator, leaving the result in the accumulator. If there is a carry out of bit 7 (CY), the CY flag is set. If byte variables are added, and if there is a carry out of bit 3 (AC), the AC flag is set. For addition of unsigned integers, the CY flag indicates that an overflow occurred.

If there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not bit 6, the OV flag is set. When adding signed integers, the OV flag indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bit 6 and bit 7 in this description refer to the most significant byte of the operand (8, 16, or 32 bit).

Four source operand addressing modes are allowed: register, direct, register-indirect, and immediate.

Flags:

CY	AC	OV
3	3	3

Example: The accumulator hods 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

Variations

ADD A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 0 1 0	0 1 0 0	immed. data

Operation: ADD
(A) ← (A) + #data

ADD A,dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 0 1 0	0 1 0 1	direct addr

Operation: ADD
(A) ← (A) + (dir8)

ADD A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 1 0	0 1 1 i

Operation: ADD
(A) ← (A) + ((Ri))

ADD A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 1 0	1 r r r

Operation: ADD
(A) ← (A) + (Rn)

ADDC A,<src>

Function: Add with carry

Description: Simultaneously adds the specified byte variable, the CY flag, and the accumulator contents, leaving the result in the accumulator. If there is a carry out of bit 7 (CY), the CY flag is set; if there is a carry out of bit 3 (AC), the AC flag is set. When adding unsigned integers, the CY flag indicates that an overflow occurred.

If there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not bit 6, the OV flag is set. When adding signed integers, the OV flag indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Bit 6 and bit 7 in this description refer to the most significant byte of the operand (8, 16, or 32 bit)

Four source operand addressing modes are allowed: register, direct, register-indirect, and immediate.

Flags:

CY	AC	OV
3	3	3

Example: The accumulator contains 0C3H (11000011B), register 0 contains 0AAH (10101010B), and the CY flag is set. After executing the instruction

ADDC A,R0

the accumulator contains 6EH (01101110B), the AC flag is clear, and the CY and OV flags are set.

Variations

ADDC A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 0 1 1	0 1 0 0	immed. data

Operation: ADDC
(A) ← (A) + (CY) + #data

ADDC A,dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 0 1 1	0 1 0 1	direct addr

Operation: ADDC
(A) ← (A) + (CY) + (dir8)

ADDC A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 1 1	0 1 1 i

Operation: ADDC
(A) ← (A) + (CY) + ((Ri))

ADDC A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 1 1	1 r r r

Operation: ADDC
(A) ← (A) + (CY) + (Rn)

AJMP addr11

Function: Absolute jump

Description: Transfers program execution to the specified address, which is formed at run time by concatenating the upper five bits of the PC (after incrementing the PC twice), opcode bits 7–5, and the second byte of the instruction. The destination must therefore be within the same 2-Kbyte “page” of program memory as the first byte of the instruction following AJMP.

Flags:

CY	AC	OV
—	—	—

Example: The label "JMPADR" is at program memory location 0123H. After executing the instruction

AJMP JMPADR

at location 0345H, the PC contains 0123H.

Bytes: 2
States: 12
Cycles: 2

[Encoding]

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP
(PC) ← (PC) + 2
(PC.10:0) ← page address

ANL <dest>,<src>

Function: Logical-AND

Description: Performs the bitwise logical-AND (Λ) operation between the specified variables and stores the results in the destination variable.

The two operands allow 10 addressing mode combinations. When the destination is the register or accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, not the input pins.

Flags:

CY	AC	OV
—	—	—

Example: If the accumulator contains 0C3H (11000011B) and register 0 contains 55H (01010101B). After executing the instruction

ANL A,R0

Accumulator 1 contains 41H (01000001B).

When the destination is a directly addressed byte, this instruction clears combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be an immediate constant contained in the instruction or a value computed in the register or accumulator at run time. The instruction

ANL P1,#01110011B

clears bits 7, 3, and 2 of output port 1.

Variations

ANL dir8,A

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 0 1	0 0 1 0	direct addr

Operation: ANL
(dir8) ← (dir8) Λ (A)

ANL dir8,#data

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 1 0 1	0 0 1 1	direct addr	immed. data

Operation: ANL
(dir8) ← (dir8) Λ #data

ANL A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 0 1	0 1 0 0	immed. data

Operation: ANL
(A) ← (A) Λ #data

ANL A,dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 0 1	0 1 0 1	direct addr

Operation: ANL
(A) ← (A) Λ (dir8)

ANL A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 1 0 1	0 1 1 i

Operation: ANL
(A) ← (A) Λ ((Ri))

ANL A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 1 0 1	1 r r r

Operation: ANL
(A) ← (A) Λ (Rn)

ANL CY,<src–bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0, clear the CY flag; otherwise leave the CY flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected.

Only direct addressing is allowed for the source operand.

Flags:

CY	AC	OV
3	—	—

Example: Set the CY flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV CY,P1.0    ;Load carry with input pin state
ANL CY,ACC.7 ;AND carry with accumulator bit 7
ANL CY,/OV     ;AND with inverse of overflow flag
```

ANL CY,bit

Bytes: 2

States: 12

Cycle: 2

[Encoding]

1 0 0 0	0 0 1 0	bit addr

Operation: ANL
(CY) ← (CY) Λ (bit51)

ANL CY,/bit

Bytes: 2

States: 12

Cycles: 2

[Encoding]

1 0 1 1	0 0 0 0	bit addr

Operation: ANL
(CY) ← (CY) Λ Ø (bit)

CJNE <dest>,<src>,rel

Function: Compare and jump if not equal.

Description: Compares the magnitudes of the first two operands and branches if their values are not equal. The branch destination is computed by adding the signed relative displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. If the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>, the CY flag is set. Neither operand is affected.

The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Flags:

CY	AC	OV
3	—	—

Example: The accumulator contains 34H and R7 contains 56H. After executing the first instruction in the sequence

```
                CJNE    R7,#60H,NOT_EQ
;               . . .   . . .           ;R7 = 60H
NOT_EQ:         JC      REQ_LOW         ; IF R7 < 60H
;               . . .   . . .           ;R7 > 60H
```

the CY flag is set and program execution continues at label NOT_EQ. By testing the CY flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then executing the instruction,

```
WAIT: CJNE A,P1,WAIT
```

clears the CY flag and continues with the next instruction in the sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program loops at this point until the P1 data changes to 34H.)

Variations

CJNE A,#data,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 0 1 1	0 1 0 0	immed. data	rel. addr

Operation:

```
(PC) ← (PC) + 3
IF (A) ≠ #data
THEN
        (PC) ← (PC) + relative offset
IF (A) < #data
THEN
        (CY) ← 1
ELSE
        (CY) ← 0
```

CJNE A,dir8,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 0 1 1	0 1 0 1	direct addr	rel. addr

Operation:
(PC) ← (PC) + 3
IF (A) ≠ dir8
THEN
    (PC) ← (PC) + relative offset
IF (A) < dir8
THEN
    (CY) ← 1
ELSE
    (CY) ← 0

CJNE @Ri,#data,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 0 1 1	0 1 1 i	immed. data	rel. addr

Operation:
(PC) ← (PC) + 3
IF ((Ri)) ≠ #data
THEN
    (PC) ← (PC) + relative offset
IF ((Ri)) < #data
THEN
    (CY) ← 1
ELSE
    (CY) ← 0

CJNE Rn,#data,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 01 1	1 r r r	immed. data	rel. addr

Operation:
(PC) ← (PC) + 3
IF (Rn) ≠ #data
THEN
    (PC) ← (PC) + relative offset
IF (Rn) < #data
THEN
    (CY) ← 1
ELSE
    (CY) ← 0

CLR A

Function: Clear accumulator

Description: Clears the accumulator (i.e., resets all bits to zero).

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 5CH (01011100B). The instruction

CLR A

clears the accumulator to 00H (00000000B).

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 1 0	0 1 0 0

Operation: CLR
(A) ← 0

CLR bit

Function: Clear bit

Description: Clears the specified bit. CLR can operate on the CY flag or any directly addressable bit.

Flags: Only for instructions with CY as the operand.

CY	AC	OV
3	—	—

Example: Port 1 contains 5DH (01011101B). After executing the instruction

CLR P1.2

port 1 contains 59H (01011001B).

Variations

CLR bit

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 1 0 0	0 0 1 0	Bit addr

Operation: CLR
(bit51) ← 0

CLR CY

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 0 0	0 0 1 1

Operation: CLR
(CY) ← 0

CPL A

Function: Complement accumulator

Description: Logically complements (Ø) each bit of the accumulator (one's complement). Clear bits are set and set bits are cleared.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 5CH (01011100B). After executing the instruction

CPL A

the accumulator contains 0A3H (10100011B).

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 1 1	0 1 0 0

Operation: CPL
(A) ← Ø(A)

CPL bit

Function: Complement bit
Description: Complements (Ø) the specified bit variable. A clear bit is set, and a set bit is cleared. CPL can operate on the CY or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data is read from the output data latch, not the input pin.

Flags: Only for instructions with CY as the operand.

CY	AC	OV
3	—	—

Example: Port 1 contains 5BH (01011101B). After executing the instruction sequence

CPL P1.1
CPL P1.2

port 1 contains 5BH (01011011B).

Variations

CPL bit

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 0 1 1	0 0 1 0	bit addr

Operation: CPL
(bit) ← Ø(bit)

CPL CY

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 0 1 1	0 0 1 1

Operation: CPL
(CY) ← Ø(CY)

DA A

Function: Decimal-adjust accumulator for addition

Description: Adjusts the 8-bit value in the accumulator that resulted from the earlier addition of two variables (each in packed-BCD format), producing two 4-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3:0 are greater than nine (XXXX1010–XXXX1111), or if the AC flag is set, six is added to the accumulator, producing the proper BCD digit in the low nibble. This internal addition sets the CY flag if a carry out of the lowest 4 bits propagated through all higher bits, but it does not clear the CY flag otherwise.

If the CY flag is now set, or if the upper four bits now exceed nine (1010XXXX–1111XXXX), these four bits are incremented by six, producing the proper BCD digit in the high nibble. Again, this sets the CY flag if there was a carry out of the upper four bits, but does not clear the carry. The CY flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple-precision decimal addition. The OV flag is not affected.

All of this occurs during one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Flags:

CY	AC	OV
3	—	—

Example: The accumulator contains 56H (01010110B), which represents the packed BCD digits of the decimal number 56. Register 3 contains 67H (01100111B), which represents the packed BCD digits of the decimal number 67. The CY flag is set. After executing the instruction sequence

```
ADDC A,R3
DA A
```

the accumulator contains 0BEH (10111110) and the CY and AC flags are clear. The Decimal Adjust instruction then alters the accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the lower two digits of the decimal sum of 56, 67, and the carry-in. The CY flag is set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum of 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator contains 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD A,#99H
DA A
```

leaves the CY flag set and 29H in the accumulator, since 30 + 99 = 129. The low byte of the sum can be interpreted to mean 30 – 1 = 29.

Bytes: 1
States: 6
Cycles: 1

[Encoding]	1 1 0 1	0 1 0 0

Operation:

```
DA
(Contents of accumulator are BCD)
IF      [[(A.3:0) > 9] V [(AC) = 1]]
          THEN (A.3:0) ← (A.3:0) + 6
                  AND
IF      [[(A.7:4) > 9] V [(CY) = 1]]
          THEN (A.7:4) ← (A.7:4) + 6
```

DEC byte

Function: Decrement

Description: Decrements the specified byte variable by 1. An original value of 00H underflows to 0FFH. Four operands addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, not the input pins.

Flags:

CY	AC	OV
—	—	—

Example: Register 0 contains 7FH (01111111B). On-chip RAM locations 7EH and 7FH contain 00H and 40H, respectively. After executing the instruction sequence

```
DEC @R0
DEC R0
DEC @R0
```

register 0 contains 7EH and on-chip RAM locations 7EH and 7FH are set to 0FFH and 3FH, respectively.

Variations

DEC A

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 0 1	0 1 0 0

Operation: DEC
(A) ← (A) – 1

DEC dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 0 0 1	0 1 0 1	dir addr

Operation: DEC
(dir8) ← (dir8) – 1

DEC @Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 0 1	0 1 1 i

Operation: DEC
((Ri)) ← ((Ri)) – 1

DEC Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 0 1	1 r r r

Operation: DEC
(Rn) ← (Rn) – 1

DIV AB

Function: Divide

Description: Divides the unsigned 8-bit integer in the accumulator by the unsigned 8-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The CY and OV flags are cleared.

Exception: if register B contains 00H, the values returned in the accumulator and register B are undefined; the CY flag is cleared and the OV flag is set.

Flags:

CY	AC	OV
0	—	3

For division by zero**:**

CY	AC	OV
0	—	1

Example: The accumulator contains 251 (0FBH or 11111011B) and register B contains 18 (12H or 00010010B). After executing the instruction

DIV AB

the accumulator contains 13 (0DH or 00001101B); register B contains 17 (11H or 00010001B), since 251 = (13 X 18) + 17; and the CY and OV flags are clear.

Bytes: 1
States: 24
Cycles: 4

[Encoding]

1 0 0 0	0 1 0 0

Operation: DIV
(A) ← quotient (A)/(B)
(B) ← remainder (A)/(B)

DJNZ <byte>,<rel–addr>

Function: Decrement and jump if not zero

Description: Decrements the specified location by 1 and branches to the address specified by the second operand if the resulting value is not zero. An original value of 00H underflows to 0FFH. The branch destination is computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, not the input pins.

Flags:

CY	AC	OV
—	—	—

Example: The on-chip RAM locations 40H, 50H, and 60H contain 01H, 70H, and 15H, respectively. After executing the following instruction sequence

```
DJNZ 40H,LABEL1
DJNZ 50H,LABEL2
DJNZ 60H,LABEL3
```

on-chip RAM locations 40H, 50H, and 60H contain 00H, 6FH, and 14H, respectively, and program execution continues at label LABEL2. (The first jump was not taken because the result was zero.)

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction.

The instruction sequence,

```
        MOV R2,#8
TOGGLE: CPL P1.7
        DJNZ R2,TOGGLE
```

toggles P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse lasts three machine cycles: two for DJNZ and one to alter the pin.

Variations

DJNZ dir8,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 1 0 1	0 1 0 1	direct addr	rel. addr

Operation:
```
DJNZ
(PC) ← (PC) + 2
(dir8) ← (dir8) – 1
IF (dir8) > 0 or (dir8) < 0
        THEN
          (PC) ← (PC) + rel
```

DJNZ Rn,rel

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 1 0 1	1 r r r	rel. addr

Operation:
```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) – 1
IF (Rn) > 0 or (Rn) < 0
        THEN
          (PC) ← (PC) + rel
```

INC <Byte>

Function: Increment

Description: Increments the specified byte variable by 1. An original value of FFH overflows to 00H. Three addressing modes are allowed for 8-bit operands: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, not the input pins.

Flags:

CY	AC	OV
—	—	—

Example: Register 0 contains 7EH (011111110B) and on-chip RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. After executing the instruction sequence

```
INC @R0
INC R0
INC @R0
```

register 0 contains 7FH and on-chip RAM locations 7EH and 7FH contain 00H and 41H, respectively.

Variations

INC A

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 0 0	0 1 0 0

Operation: INC
(A) ← (A) + 1

INC dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 0 0 0	0 1 0 1	direct addr

Operation: INC
(dir8) ← (dir8) + 1

INC @Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 0 0	0 1 1 i

Operation: INC
((Ri) ← ((Ri)) + 1

INC Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 0 0	1 r r r

Operation: INC
(Rn) ← (Rn) + 1

INC DPTR

Function: Increment data pointer

Description: Increments the 16-bit data pointer by one. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low byte of the data pointer (DPL) from 0FFH to 00H increments the high byte of the data pointer (DPH) by one. An overflow of the high byte (DPH) does not increment the high word of the extended data pointer (DPX = DR56).

Flags:

CY	AC	OV
—	—	—

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. After the instruction sequence

```
INC DPTR
INC DPTR
INC DPTR
```

DPH and DPL contain 13H and 01H, respectively.

Bytes: 1
States: 12
Cycles: 2

[Encoding]

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1

JB bit,rel

Function: Jump if bit set

Description: If the specified bit is a one, jump to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified.

Flags:

CY	AC	OV
—	—	—

Example: Input port 1 contains 11001010B and the accumulator contains 56 (01010110B). After the instruction sequence

```
JB P1.2,LABEL1
JB ACC.2,LABEL2
```

program execution continues at label LABEL2.

Variations

JB bit,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 0 1 0	0 0 0 0	bit addr	rel. addr

Hex Code in: **Binary Mode = [Encoding]**
Source Mode = [Encoding]

Operation:
```
JB
(PC) ← (PC) + 3
IF (bit51) = 1
        THEN
          (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if bit is set and clear bit

Description: If the specified bit is one, branch to the specified address; otherwise proceed with the next instruction. The bit is not cleared if it is already a zero. The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction.

Note: When this instruction is used to test an output pin, the value used as the original data is read from the output data latch, not the input pin.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 56H (01010110B). After the instruction sequence

```
JBC ACC.3,LABEL1
JBC ACC.2,LABEL2
```

the accumulator contains 52H (01010010B) and program execution continues at label LABEL2.

Variations

JBC bit,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 0 0 1	0 0 0 0	bit addr	rel. addr

Operation:
```
JBC
(PC) ← (PC) + 3
IF (bit) = 1
        THEN
          (bit) ← 0
          (PC) ← (PC) + rel
```

JC rel

Function: Jump if carry is set

Description: If the CY flag is set, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Flags:

CY	AC	OV
!	—	—

Example: The CY flag is clear. After the instruction sequence

```
JC          LABEL1
CPL CY
JC LABEL 2
```

the CY flag is set and program execution continues at label LABEL2.

Bytes: 2
States: 12
Cycles: 2

[Encoding]	0 1 0 0	0 0 0 0	rel. addr

Operation:
```
JC
(PC) ← (PC) + 2
IF (CY) = 1
      THEN
         (PC) ← (PC) + rel
```

JMP @A+DPTR

Function: Jump indirect

Description: Add the 8-bit unsigned contents of the accumulator with the 16-bit data pointer and load the resulting sum into the lower 16 bits of the program counter. This is the address for subsequent instruction fetches. The contents of the accumulator and the data pointer are not affected.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains an even number from 0 to 6. The following sequence of instructions branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
            MOV     DPTR,#JMP_TBL
            JMP     @A+DPTR
JMP_TBL:    AJMP    LABEL0
            AJMP    LABEL1
            AJMP    LABEL2
            AJMP    LABEL3
```

If the accumulator contains 04H at the start this sequence, execution jumps to LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1
States: 12
Cycles: 2

[Encoding]

0 1 1 1	0 0 1 1

Operation: JMP
(PC.15:0) ← (A) + (DPTR)

JNB bit,rel

Function: Jump if bit not set

Description: If the specified bit is clear, branch to the specified address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified.

Flags:

CY	AC	OV
—	—	—

Example: Input port 1 contains 11001010B and the accumulator contains 56H (01010110B). After executing the instruction sequence

```
JNB P1.3,LABEL1
JNB ACC.3,LABEL2
```

program execution continues at label LABEL2.

Variations

JNB bit,rel

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 0 1 1	0 0 0 0	bit addr	rel. addr

Operation: JNB
(PC) ← (PC) + 3
IF (bit) = 0
THEN (PC) ← (PC) + rel

JNC rel

Function: Jump if carry not set

Description: If the CY flag is clear, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The CY flag is not modified.

Flags:

CY	AC	OV
!	—	—

Example: The CY flag is set. The instruction sequence

```
JNC LABEL1
CPL CY
JNC LABEL2
```

clears the CY flag and causes program execution to continue at label LABEL2.

Bytes: 2
States: 12
Cycles: 2

Operation:
```
JNC
(PC) ← (PC) + 2
IF (CY) = 0
    THEN (PC) ← (PC) + rel
```

JNZ rel

Function: Jump if accumulator not zero

Description: If any bit of the accumulator is set, branch to the specified address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 00H. After executing the instruction sequence

```
JNZ LABEL1
INC A
JNZ LABEL2
```

the accumulator contains 01H and program execution continues at label LABEL2.

Bytes: 2
States: 12
Cycles: 2

[Encoding]

0 1 1 1	0 0 0 0	rel. addr

Operation: JNZ
(PC) ← (PC) + 2
IF (A) ≠ 0
THEN (PC) ← (PC) + rel

JZ rel

Function: Jump if accumulator zero

Description: If all bits of the accumulator are clear (zero), branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 01H. After executing the instruction sequence

```
JZ LABEL1
DEC A
JZ LABEL2
```

the accumulator contains 00H and program execution continues at label LABEL2.

Bytes: 2
States: 12
Cycles: 2

[Encoding]

0 1 1 0	0 0 0 0	rel. addr

Operation: JZ
(PC) ← (PC) + 2
IF (A) = 0
THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long call

Description: Calls a subroutine located at the specified address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first). The stack pointer is incremented by two. The high and low bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the 64-Kbyte region of memory where the next instruction is located.

Flags:

CY	AC	OV
—	—	—

Example: The stack pointer contains 07H and the label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction

LCALL SUBRTN

at location 0123H, the stack pointer contains 09H, on-chip RAM locations 08H and 09H contain 01H and 26H, and the PC contains 1234H.

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 0 0 1	0 0 1 0	addr15–addr8	addr7–addr0

Operation: LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC.7:0)
(SP) ← (SP) + 1
((SP)) ← (PC.15:8)
(PC) ← (addr.15:0)

LJMP addr16

Function: Long Jump

Description: Causes an unconditional branch to the specified address, by loading the high and low bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the 64-Kbyte memory region where the next instruction is located.

Flags:

CY	AC	OV
—	—	—

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. After executing the instruction

LJMP JMPADR

at location 0123H, the program counter contains 1234H.

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 0 0 0	0 0 1 0	addr15–addr8	addr7–addr0

Operation: LJMP
(PC) ← (addr.15:0)

MOV <dest>,<src>

Function: Move byte variable

Description: Copies the byte variable specified by the second operand into the location specified by the first operand. The source byte is not affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Flags:

CY	AC	OV
—	—	—

Example: On-chip RAM location 30H contains 40H, on-chip RAM location 40H contains 10H, and input port 1 contains 11001010B (0CAH). After executing the instruction sequence

```
MOV     R0,#30H         ;R0 < = 30H
MOV     A,@R0           ;A < = 40H
MOV     R1,A            ;R1 < = 40H
MOV     B,@R1           ;B < = 10H
MOV     @R1,P1          ;RAM (40H) < = 0CAH
MOV     P2,P1           ;P2 #0CAH
```

register 0 contains 30H, the accumulator and register 1 contain 40H, register B contains 10H, and on-chip RAM location 40H and output port 2 contain 0CAH (11001010B).

Variations

MOV A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 1 1	0 1 0 0	immed. data

Operation: MOV
(A) ← #data

MOV dir8,#data

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 1 1 1	0 1 0 1	direct addr	immed. data

Operation: MOV
(dir8) ← #data

MOV @Ri,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 1 1	0 1 1 i	immed. data

Operation: MOV
((Ri)) ← #data

MOV Rn,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 1 1	1 r r r r	immed. data

Operation: MOV
(Rn) ← #data

MOV dir8,dir8

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 0 0 0	0 1 0 1	direct addr	direct addr

Operation: MOV
(dir8) ← (dir8)

MOV dir8,@Ri

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 0 0 0	0 1 1 i	direct addr

Operation: MOV
(dir8) ← ((Ri))

MOV dir8,Rn

Bytes: 2
States: 12

Cycles: 2

[Encoding]

1 0 0 0	1 r r r	direct addr

Operation: MOV
(dir8) ← (Rn)

MOV @Ri,dir8

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1010	011i	direct addr

Operation: MOV
((Ri)) ← (dir8)

MOV Rn,dir8

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 0 1 0	1 r r r	direct addr

Operation: MOV
(Rn) ← (dir8)

MOV A,dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 1 1 0	0 1 0 1	direct addr

Operation: MOV
(A) ← (dir8)

MOV A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

MOV dir8,A

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 1 1 1	0 1 0 1	direct addr

Operation: MOV
(dir8) ← (A)

MOV @Ri,A

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← (A)

MOV Rn,A

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 1 1	1 1 1 r

Operation: MOV
(Rn) ← (A)

MOV <dest–bit>,<src–bit>

Function: Move bit data

Description: Copies the Boolean variable specified by the second operand into the location specified by the first operand. One of the operands must be the CY flag; the other may be any directly addressable bit. Does not affect any other register.

Flags:

CY	AC	OV
3	—	—

Example: The CY flag is set, input Port 3 contains 11000101B, and output Port 1 contains 35H (00110101B). After executing the instruction sequence

```
MOV P1.3,CY
MOV CY,P3.3
MOV P1.2,CY
```

the CY flag is clear and Port 1 contains 39H (00111001B).

Variations

MOV bit,CY

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 0 0 1	0 0 1 0	bit addr

Operation: MOV
(bit51) ← (CY)

MOV CY,bit

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 0 1 0	0 0 1 0	bit addr

Operation: MOV
(CY) ← (bit51)

MOV DPTR,#data16

Function: Load data pointer with a 16-bit constant

Description: Loads the 16-bit data pointer (DPTR) with the specified 16-bit constant. The high byte of the constant is loaded into the high byte of the data pointer (DPH). The low byte of the constant is loaded into the low byte of the data pointer (DPL).

Flags:

CY	AC	OV

—	—	—

Example: After executing the instruction

MOV DPTR,#1234H

DPTR contains 1234H (DPH contains 12H and DPL contains 34H).

Bytes: 3
States: 12
Cycles: 2

[Encoding]

1 0 0 1	0 0 0 0	data hi	data low

Operation: MOV
(DPTR) ← #data16

MOVC A,@A+<base–reg>

Function: Move code byte

Description: Loads the accumulator with a code byte or constant from program memory. The address of the byte fetched is the sum of the original unsigned 8-bit accumulator contents and the contents of a 16-bit base register, which may be the 16 LSBs of the data pointer or PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains a number between 0 and 3. The following instruction sequence translates the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
RELPC:  INC     A
        MOVC    A,@A+PC
        RET
        DB      66H
        DB      77H
        DB      88H
        DB      99H
```

If the subroutine is called with the accumulator equal to 01H, it returns with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

Variations

MOVC A,@A+PC

Bytes: 1
States: 12

Cycles: 2

[Encoding]

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

MOVC A,@A+DPTR

Bytes: 1
States: 12
Cycles: 2

[Encoding]

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A) + (DPTR))

MOVX <dest>,<src>

Function: Move external

Description: Transfers data between the accumulator and a byte in external data RAM. There are two types of instructions. One provides an 8-bit indirect address to external data RAM; the second provides a 16-bit indirect address to external data RAM.

In the first type of MOVX instruction, the contents of R0 or R1 in the current register bank provides an 8-bit address on port 0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For larger arrays, any port pins can be used to output higher address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a 16-bit address. Port 2 outputs the upper eight address bits (from DPH) while port 0 outputs the lower eight address bits (from DPL).

For both types of moves, the data is multiplexed with the lower address bits on port 0.

It is possible in some situations to mix the two MOVX types. A large RAM array with its upper address lines driven by P2 can be addressed via the data pointer, or with code to output upper address bits to P2 followed by a MOVX instruction using R0 or R1.

Flags:

CY	AC	OV
—	—	—

Example: An external 256-byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to port 0. Port 3 provides control lines for the external RAM. ports 1 and 2 are used for normal I/O. R0 and R1 contain 12H and 34H. Location 34H of the external RAM contains 56H. After executing the instruction sequence

```
MOVX A,@R1
MOVX @R0,A
```

the accumulator and external RAM location 12H contain 56H.

Variations

MOVX A,@DPTR

Bytes: 1
States: 12
Cycles: 2

[Encoding]

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX A,@Ri

Bytes: 1
States: 12
Cycles: 2

[Encoding]

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX @DPTR,A

Bytes: 1
States: 12
Cycles: 2

[Encoding]

1 1 1 1	0 0 0 0

Operation: MOVX
((DPTR)) ← (A)

MOVX @Ri,A

Bytes: 1
States: 12
Cycles: 2

[Encoding]

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri)) ← (A)

MUL AB

Function: Multiply

Description: Multiplies the unsigned 8-bit integers in the accumulator and register B. The low byte of the 16-bit product is left in the accumulator, and the high byte is left in register B. If the product is greater than 255 (0FFH) the OV flag is set; otherwise it is clear. The CY flag is always clear.

Flags:

CY	AC	OV
0	—	3

Example: The accumulator contains 80 (50H) and register B contains 160 (0A0H). After executing the instruction

MUL AB

which gives the product 12,800 (3200H), register B contains 32H (00110010B), the accumulator contains 00H, the OV flag is set, and the CY flag is clear.

Bytes: 1

States: 24

Cycles: 4

[Encoding]	1 0 1 0	0 1 0 0

Operation: MUL
(A) ← low byte of (A) X (B)
(B) ← high byte of (A) X (B)

NOP

Function: No operation

Description: Execution continues at the following instruction. Affects the PC register only.

Flags:

CY	AC	OV
—	—	—

Example: You want to produce a low-going output pulse on bit 7 of Port 2 that lasts exactly 11 states. A simple CLR-SETB sequence generates an one-cycle pulse, so four additional cycles must be inserted. You can insert the four additional cycles (if no interrupts are enabled) with the following instruction sequence:

```
CLR  P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

Bytes: 1

States: 6

Cycles: 1

[Encoding]

0 0 0 0	0 0 0 0

Operation: NOP
(PC) ← (PC) + 1

ORL <dest> <src>

Function: Logical-OR for byte variables

Description: Performs the bitwise logical-OR operation (V) between the specified variables, storing the results in the destination operand.

The destination operand can be an accumulator or direct address.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can be register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, not the input pins.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 0C3H (11000011B) and R0 contains 55H (01010101B). After executing the instruction

ORL A,R0

the accumulator contains 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be a constant data value in the instruction or a variable computed in the accumulator at run time. After executing the instruction

ORL P1,#00110010B

sets bits 5, 4, and 1 of output Port 1.

ORL dir8,A

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 0 0	0 0 1 0	direct addr

Operation: ORL
(dir8) ← (dir8) V (A)

ORL dir8,#data

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 1 0 0	0 0 1 1	direct addr	immed. data

Operation: ORL
(dir8) ← (dir8) V #data

ORL A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 0 0	0 1 0 0	immed. data

Operation: ORL
(A) ← (A) V #data

ORL A,dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 0 0	0 1 0 1	direct addr

Operation: ORL
(A) ← (A) V (dir8)

ORL A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 1 0 0	0 1 1 i

Operation: ORL
(A) ← (A) V ((Ri))

ORL A,Rn

Bytes: 1
States: 6

Cycles: 1

[Encoding]

0 1 0 0	1 r r r

Operation: ORL
(A) ← (A) V (Rn)

ORL CY,<src–bit>

Function: Logical-OR for bit variables

Description: Sets the CY flag if the Boolean value is a logical 1; leaves the CY flag in its current state otherwise . A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected.

Flags:

CY	AC	OV
3	—	—

Example: Set the CY flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV CY,P1.0          ;LOAD CARRY WITH INPUT PIN P10
ORL CY,ACC.7 ;OR CARRY WITH THE ACC. BIT 7
ORL CY,/OV           ;OR CARRY WITH THE INVERSE OF OV.
```

Variations

ORL CY,bit

Bytes: 2
States: 12
Cycles: 2

[Encoding]

0 1 1 1	0 0 1 0	bit addr

Operation: ORL
(CY) ← (CY) V (bit51)

ORL CY,/bit

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 0 1 0	0 0 0 0	bit addr

Operation: ORL
(CY) ← (CY) V¬ (bit51)

POP dir8
Function: Pop from stack

Description: Reads the contents of the on-chip RAM location addressed by the stack pointer, then decrements the stack pointer by one. The value read at the original RAM location is transferred to the newly addressed location, which can be 8-bit or 16-bit.

Flags:

CY	AC	OV
—	—	—

Example: The stack pointer contains 32H and on-chip RAM locations 30H through 32H contain 01H, 23H, and 20H, respectively. After executing the instruction sequence

```
POP DPH
POP DPL
```

the stack pointer contains 30H and the data pointer contains 0123H. After executing the instruction

```
POP SP
```

the stack pointer contains 20H. Note that in this special case the stack pointer was decremented to 2FH before it was loaded with the value popped (20H).

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 1 0 1	0 0 0 0	direct addr

Operation:
```
POP
(dir8) ← ((SP))
(SP) ← (SP) – 1
```

PUSH dir8
Function: Push onto stack

Description: Increments the stack pointer by one. The contents of the specified variable are then copied into the on-chip RAM location addressed by the stack pointer.

Flags:

CY	AC	OV
—	—	—

Example: On entering an interrupt routine, the stack pointer contains 09H and the data pointer contains 0123H. After executing the instruction sequence

```
PUSH DPL
PUSH DPH
```

the stack pointer contains 0BH and on-chip RAM locations 0AH and 0BH contain 01H and 23H, respectively.

Bytes: 2
States: 12

Cycles: 2

[Encoding]

1 1 0 0	0 0 0 0	direct addr

Operation: PUSH
(SP) ← (SP) + 1
((SP)) ← (dir8)

RET
Function: Return from subroutine

Description: Pops the high and low bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, which normally is the instruction immediately following ACALL or LCALL.

Flags:

CY	AC	OV
—	—	—

Example: The stack pointer contains 0BH and on-chip RAM locations 0AH and 0BH contain 01H and 23H, respectively. After executing the instruction,

RET

the stack pointer contains 09H and program execution continues at location 0123H.

Bytes: 1
States: 12
Cycles: 2

[Encoding]

0 0 1 0	0 0 1 0

Operation: RET
(PC).15:8 ← ((SP))
(SP) ← (SP) – 1
(PC).7:0 ← ((SP))
(SP) ← (SP) – 1

RETI

Function: Return from interrupt

Description: RETI pops the high and low bytes of the PC successively from the stack and uses them as the 16-bit return address. The stack pointer is decremented by two. No other registers are affected, the PSW is not automatically restored to its pre-interrupt status.

Hardware restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. Program execution continues at the return address, which normally is the instruction immediately after the point at which the interrupt request was detected. If an interrupt of the same or lower priority is pending when the RETI instruction is executed, that one instruction is executed before the pending interrupt is processed.

Flags:

CY	AC	OV

—	—	—

Example: The stack pointer contains 0BH. An interrupt was detected during the instruction ending at location 0122H. On-chip RAM locations 0AH and 0BH contain 01H and 23H, respectively. After executing the instruction

RETI

the stack pointer contains 09H and program execution continues at location 0123H.

Bytes: 1
States 12
Cycles: 2

[Encoding]

0 0 1 1	0 0 1 0

Operation:

RETI
(PC).15:8 ←((SP))
(SP) ← (SP) – 1
(PC).7:0 ¨ ((SP))
(SP) ←(SP) – 1

RL A

Function: Rotate accumulator left

Description: Rotates the eight bits in the accumulator one bit to the left. Bit 7 is rotated into the bit 0 position.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 0C5H (11000101B). After executing the instruction,

RL A

the accumulator contains 8BH (10001011B); the CY flag is unaffected.

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 0 1 0	0 0 1 1

Operation: RL
(A).a+1 ← (A).a
(A).0 ← (A).7

RLC A

Function: Rotate accumulator left through the carry flag

Description: Rotates the eight bits in the accumulator and the CY flag one bit to the left. Bit 7 moves into the CY flag position and the original state of the CY flag moves into bit 0 position.

Flags:

CY	AC	OV
3	—	—

Example: The accumulator contains 0C5H (11000101B) and the CY flag is clear. After executing the instruction

RLC A

the accumulator contains 8AH (10001010B) and the CY flag is set.

Bytes: 1
States: 6
Cycles: 1

[Encoding]	0 0 1 1	0 0 1 1

Operation: RLC
(A).a+1 ← (A).a
(A).0 ← (CY)
(CY) ← (A).7

RR A

Function: Rotate accumulator right

Description: Rotates the 8 bits in the accumulator one bit to the right. Bit 0 is moved into the bit 7 position.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 0C5H (11000101B). After executing the instruction

RR A

the accumulator contains 0E2H (11100010B) and the CY flag is unaffected.

Bytes: 1
States: 6
Cycles: 1

[Encoding]	0 0 0 0	0 0 1 1

Operation: RR
(A).a ← (A).a+1
(A).7 ← (A) .0

RRC A

Function: Rotate accumulator right through carry flag

Description: Rotates the eight bits in the accumulator and the CY flag one bit to the right. Bit 0 moves into the CY flag position; the original value of the CY flag moves into the bit 7 position.

Flags:

CY	AC	OV
3	—	—

Example: The accumulator contains 0C5H (11000101B) and the CY flag is clear. After executing the instruction

RRC A

the accumulator contains 62 (01100010B) and the CY flag is set.

Bytes: 1
States: 6
Cycles: 1

[Encoding]	0 0 0 1	0 0 1 1

Operation: RRC
(A).a ← (A).a+1
(A).7 ← (CY)
(CY) ← (A).0

SETB <bit>

Function: Set bit

Description: Sets the specified bit to one. SETB can operate on the CY flag or any directly addressable bit.

Flags: No flags are affected except the CY flag for instruction with CY as the operand.

CY	AC	OV
3	—	—

Example: The CY flag is clear and output Port 1 contains 34H (00110100B). After executing the instruction sequence

SETB CY
SETB P1.0

the CY flag is set and output Port 1 contains 35H (00110101B).

SETB bit

Bytes: 2
States: 6

Cycles: 1

[Encoding]

1 1 0 1	0 0 1 0	bit addr

Operation: SETB
(bit51) ← 1

SETB CY

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 0 1	0 0 1 1

Operation: SETB
(CY) ← 1

SJMP rel

Function: Short jump

Description: Program control branches unconditionally to the specified address. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Flags:

CY	AC	OV
—	—	—

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction

SJMP RELADR

assembles into location 0100H. After executing the instruction, the PC contains 0123H.

(Note: In the above example, the instruction following SJMP is located at 102H. Therefore, the displacement byte of the instruction is the relative offset (0123H–0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2
States: 12
Cycles: 2

[Encoding]

1 0 0 0	0 0 0 0	rel. addr

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

SUBB A,<src–byte>

Function: Subtract with borrow

Description: SUBB subtracts the specified variable and the CY flag together from the accumulator, leaving the result in the accumulator. SUBB sets the CY (borrow) flag if a borrow is needed for bit 7, and clears CY otherwise. (If CY was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the CY flag is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.
When subtracting signed integers the OV flag indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

Bit 6 and bit 7 in this description refer to the most significant byte of the operand (8, 16, or 32 bit).

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Flags:

CY	AC	OV
3	3	3

Example: The accumulator contains 0C9H (11001001B), register 2 contains 54H (01010100B), and the CY flag is set. After executing the instruction

SUBB A,R2

the accumulator contains 74H (01110100B), the CY and AC flags are clear, and the OV flag is set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the CY (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR CY instruction.

Variations

SUBB A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 0 0 1	0 1 0 0	immed. data

Operation: SUBB
(A) ← (A) – (CY) – #data

SUBB A,dir8

Bytes: 2

States: 6
Cycles: 1

[Encoding]

1 0 0 1	0 1 0 1	direct addr

Operation: SUBB
(A) ← (A) – (CY) – (dir8)

SUBB A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 0 0 1	0 1 1 i

Operation: SUBB
(A) ← (A) – (CY) – ((Ri))

SUBB A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 0 0 1	1 r r r

Operation: SUBB
(A) ← (A) – (CY) – (Rn)

SWAP A

Function: Swap nibbles within the accumulator

Description: Interchanges the low and high nibbles (4-bit fields) of the accumulator (bits 3–0 and bits 7–4). This operation can also be thought of as a 4-bit rotate instruction.

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 0C5H (11000101B). After executing the instruction

SWAP A

the accumulator contains 5CH (01011100B).

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 0 0	0 1 0 0

Operation: SWAP
(A).3:0 → ← (A).7:4

XCH A,<byte>

Function: Exchange accumulator with byte variable

Description: Loads the accumulator with the contents of the specified variable, at the same time writing the original accumulator contents to the specified variable. The source/destination operand can use register, direct, or register-indirect addressing.

Flags:

CY	AC	OV
—	—	—

Example: R0 contains the address 20H, the accumulator contains 3FH (00111111B) and on-chip RAM location 20H contains 75H (01110101B). After executing the instruction

XCH A,@R0

RAM location 20H contains 3FH (00111111B) and the accumulator contains 75H (01110101B).

Variations

XCH A,dir8

Bytes: 2
States: 6
Cycles: 1

[Encoding]

1 1 0 0	0 1 0 1	direct addr

Operation: XCH
(A) → ← (dir8)

XCH A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 0 0	0 1 1 i

Operation: XCH
(A) → ← ((Ri))

XCH A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

1 1 0 0	1 r r r

Operation: XCH
(A) → ← (Rn)

Variations

XCHD A,@Ri

Function: Exchange digit

Description: Exchanges the low nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the on-chip RAM location indirectly addressed by the specified register. Does not affect the high nibble (bits 7-4) of either register.

Flags:

CY	AC	OV
—	—	—

Example: R0 contains the address 20H, the accumulator contains 36H (00110110B), and on-chip RAM location 20H contains 75H (01110101B). After executing the instruction

XCHD A,@R0

on-chip RAM location 20H contains 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

States: 6

Cycles: 1

[Encoding]	1 1 0 1	0 1 1 i

Operation: XCHD
(A).3:0 → ← ((Ri)).3:0

XRL <dest>,<src>

Function: Logical Exclusive-OR for byte variables

Description: Performs the bitwise logical Exclusive-OR operation (∀) between the specified variables, storing the results in the destination. The destination operand can be the accumulator, a register, or a direct address.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source addressing can be register, direct, register-indirect, or immediate; when the destination is a direct address, the source can be the accumulator or immediate data.

(Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, not the input pins.)

Flags:

CY	AC	OV
—	—	—

Example: The accumulator contains 0C3H (11000011B) and R0 contains 0AAH (10101010B). After executing the instruction

XRL A,R0

the accumulator contains 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run time. The instruction

XRL P1,#00110001B

complements bits 5, 4, and 0 of output Port 1.

Variations

XRL dir8,A

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 1 0	0 0 1 0	direct addr

Operation: XRL
(dir8) ← (dir8) ∀ (A)

XRL dir8,#data

Bytes: 3
States: 12
Cycles: 2

[Encoding]

0 1 1 0	0 0 1 1	direct addr	immed. data

Operation: XRL
(dir8) ← (dir8) ∀ #data

XRL A,#data

Bytes: 2
States: 6
Cycles: 1

[Encoding]

0 1 1 0	0 1 0 0	immed. data

Operation: XRL
(A) ← (A) ∀ #data

XRL A,dir8

Bytes: 2

States: 6
Cycles: 1

[Encoding]

0 1 1 0	0 1 0 1	direct addr

Operation: XRL
(A) ← (A) ∀ (dir8)

XRL A,@Ri

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 1 1 0	0 1 1 i

Operation: XRL
(A) ← (A) ∀ ((Ri))

XRL A,Rn

Bytes: 1
States: 6
Cycles: 1

[Encoding]

0 1 1 0	1 r r r

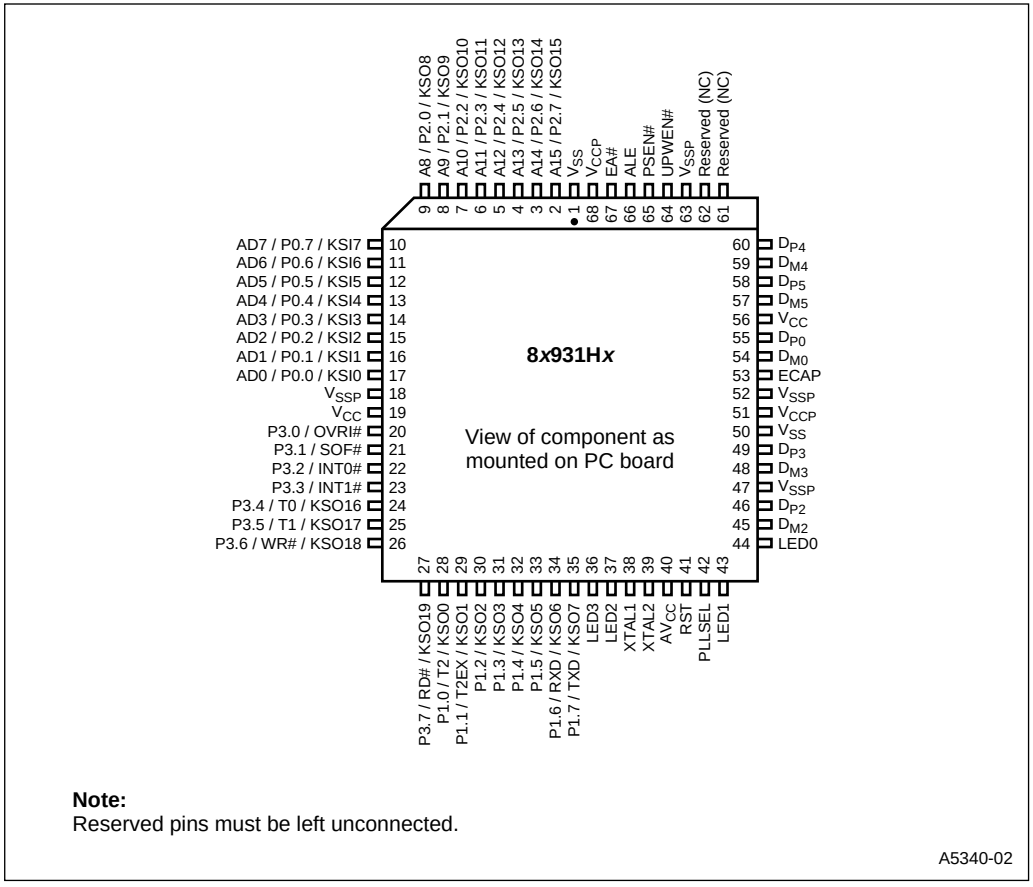
Operation: XRL
(A) ← (A) ∀ (Rn)

B

Pin Descriptions

APPENDIX B
PIN DESCRIPTIONS

This appendix provides reference information regarding the external signals of the 8*x*931. The 8*x*931 is available in dual in-line (64-pin S-DIP, 8*x*931HA only), quad flatpack (64-pin QFP), and plastic leaded chip carrier (68-pin PLCC) packages. See Figures B-1 through B-5. Tables B-4 through B-6 list the signals by functional category. Table B-7 describes each of the signals. It lists the signal type (input, output, power, or ground) and the alternative functions of multi-function pins.



Note:
Reserved pins must be left unconnected.

Figure B-1. 8*x*931HA 68-pin PLCC Package

Figure B-2 illustrates a diagram of the 8*x*931HA SDIP package. Table B-2 and Table B-5 contain indexes of the pin arrangement. Table B-7 contains the signal descriptions for all pins.

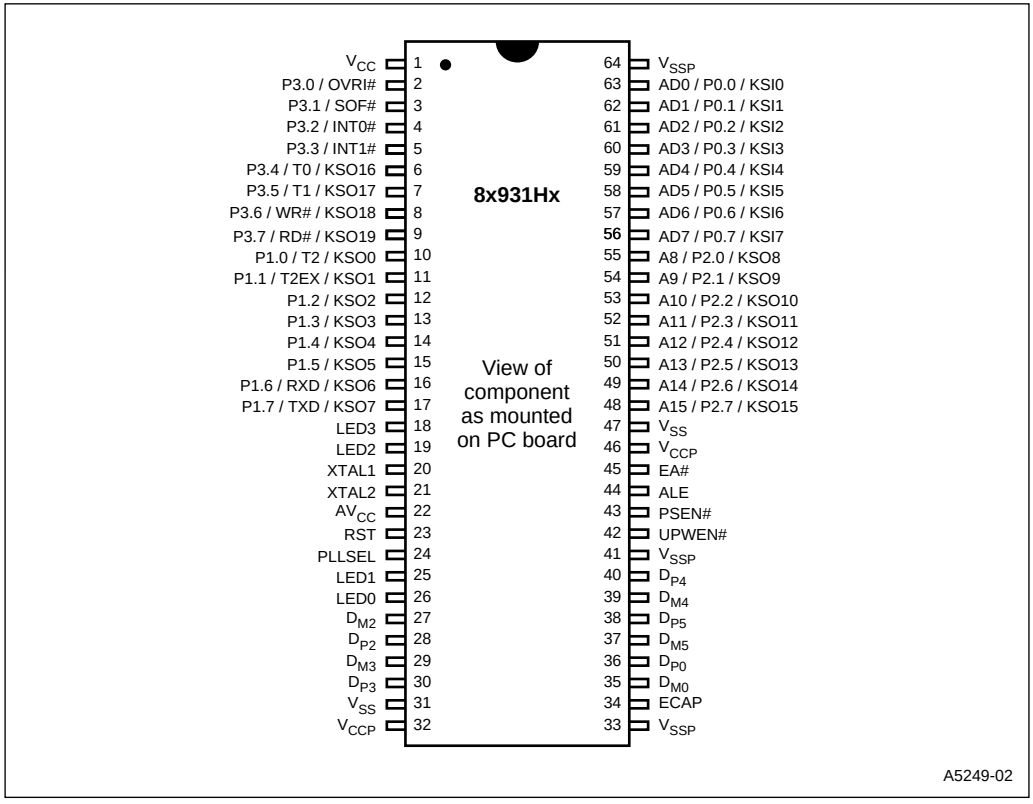


Figure B-2. 8*x*931HA 64-pin SDIP Package

Figure B-3 illustrates a diagram of the 8*x*931HA QFP package. Table B-3 and Table B-6 contain indexes of the pin arrangement. Table B-7 contains the signal descriptions for all pins.

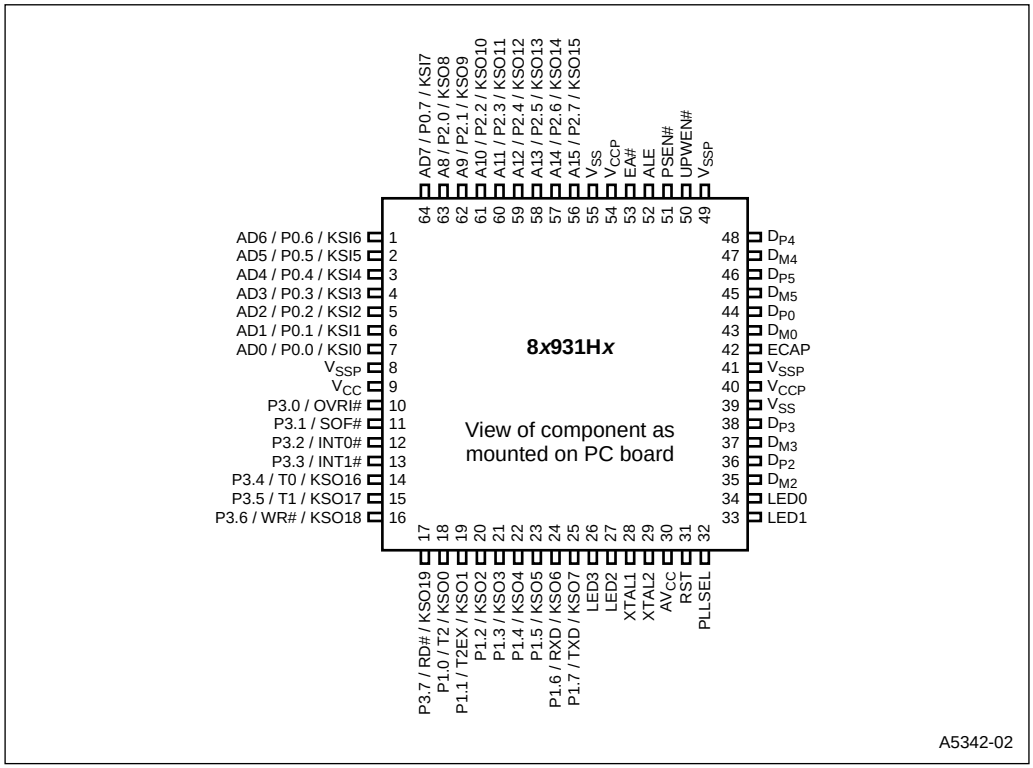


Figure B-3. 8*x*931HA 64-pin QFP Package

Figure B-4 illustrates a diagram of the 8*x*931AA QFP package. Table B-3 and Table B-6 contain indexes of the pin arrangement. Table B-7 contains the signal descriptions for all pins.

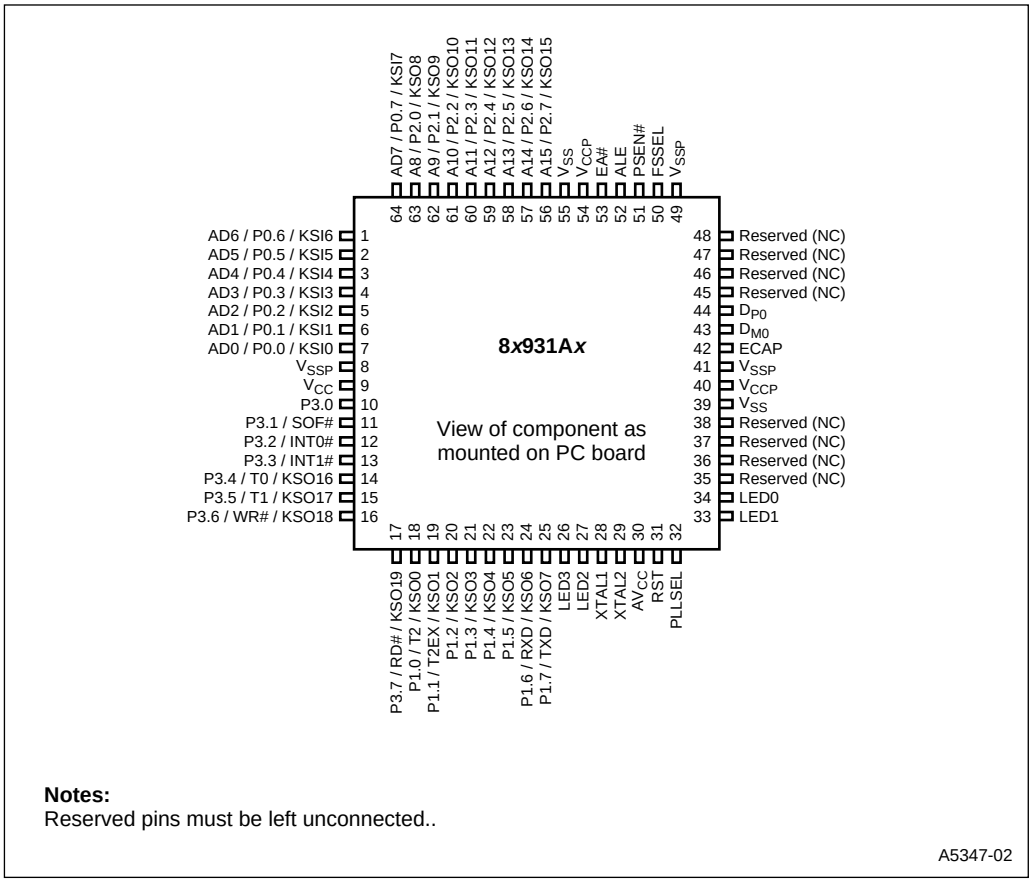
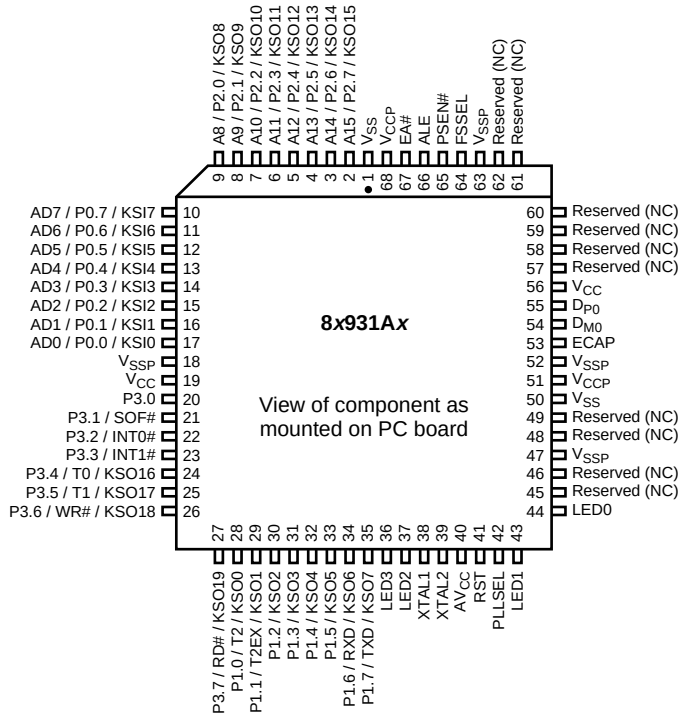


Figure B-4. 8*x*931AA 64-pin QFP Package

Figure B-5 illustrates a diagram of the 8*x*931AA PLCC package. Table B-1 and Table B-4 contain indexes of the pin arrangement. Table B-7 contains the signal descriptions for all pins.



Note:
Reserved pins must be left unconnected.

A5348-02

Figure B-5. 8*x*931AA 68-pin PLCC Package

Table B-1. 68-pin PLCC Pin Assignment

Pin	Name	Pin	Name	Pin	Name
1	V_{SS}	24	P3.4/T0/KSO16	47	V_{SSP}
2	A15/P2.7/KSO15	25	P3.5/T1/KSO17	48	Reserved†/ D_{M3}††
3	A14/P2.6/KSO14	26	P3.6/WR#/KSO18	49	Reserved†/ D_{P3}††
4	A13/P2.5/KSO13	27	P3.7/RD#/KSO19	50	V_{SS}
5	A12/P2.4/KSO12	28	P1.0/T2/KSO0	51	V_{CCP}
6	A11/P2.3/KSO11	29	P1.1/T2EX/KSO1	52	V_{SSP}
7	A10/P2.2/KSO10	30	P1.2/KSO2	53	ECAP
8	A9/P2.1/KSO9	31	P1.3/KSO3	54	D_{M0}
9	A8/P2.0/KSO8	32	P1.4/KSO4	55	D_{P0}
10	AD7/P0.7/KSI7	33	P1.5/KSO5	56	V_{CC}
11	AD6/P0.6/KSI6	34	P1.6/KSO6/RXD	57	Reserved†/ D_{M5}††
12	AD5/P0.5/KSI5	35	P1.7/KSO7/TXD	58	Reserved†/ D_{P5}††
13	AD4/P0.4/KSI4	36	LED3	59	Reserved†/ D_{M4}††
14	AD3/P0.3/KSI3	37	LED2	60	Reserved†/ D_{P4}††
15	AD2/P0.2/KSI2	38	XTAL1	61	Reserved (NC)
16	AD1/P0.1/KSI1	39	XTAL2	62	Reserved (NC)
17	AD0/P0.0/KSI0	40	AV_{CC}	63	V_{SSP}
18	V_{SSP}	41	RST	64	FSSEL†/ UPWEN#††
19	V_{CC}	42	PLLSEL	65	PSEN#
20	P3.0/ OVRI#††	43	LED1	66	ALE
21	P3.1/SOF#	44	LED0	67	EA#
22	P3.2/INT0#	45	Reserved†/ D_{M2}††	68	V_{CCP}
23	P3.3/INT1#	46	Reserved†/ D_{P2}††		

† Specific to the 8*x*931AA

†† Specific to the 8*x*931HA

Table B-2. 64-pin SDIP Pin Assignment

Pin	Name
1	V_{CC}
2	P3.0/OVRI#
3	P3.1/SOF#
4	P3.2/INT0#
5	P3.3/INT1#
6	P3.4/T0/KSO16
7	P3.5/T1/KSO17
8	P3.6/WR#/KSO18
9	P3.7/RD#/KSO19
10	P1.0/T2/KSO0
11	P1.1/T2EX/KSO1
12	P1.2/KSO2
13	P1.3/KSO3
14	P1.4/KSO4
15	P1.5/KSO5
16	P1.6/RXD/KSO6
17	P1.7/TXD/KSO7
18	LED3
19	LED2
20	XTAL1
21	XTAL2
22	AVCC

Pin	Name
23	RST
24	PLLSEL
25	LED1
26	LED0
27	DM2
28	DP2
29	DM3
30	DP3
31	V_{SS}
32	V_{CCP}
33	V_{SSP}
34	ECAP
35	D_{M0}
36	D_{P0}
37	D_{M5}
38	D_{P5}
39	D_{M4}
40	D_{P4}
41	V_{SSP}
42	UPWEN#
43	PSEN#
44	ALE

Pin	Name
45	EA#
46	V_{CCP}
47	V_{SS}
48	A15/P2.7/KSO15
49	A14/P2.6/KSO14
50	A13/P2.5/KSO13
51	A12/P2.4/KSO12
52	A11/P2.3/KSO11
53	A10/P2.2/KSO10
54	A9/P2.1/KSO9
55	A8/P2.0/KSO8
56	AD7/P0.7/KSI7
57	AD6/P0.6/KSI6
58	AD5/P0.5/KSI5
59	AD4/P0.4/KSI4
60	AD3/P0.3/KSI3
61	AD2/P0.2/KSI2
62	AD1/P0.1/KSI1
63	AD0/P0.0/KSI0
64	V_{SSP}

Table B-3. 64-pin QFP Pin Assignment

Pin	Name
1	AD6/P0.6/KSI6
2	AD5/P0.5/KSI5
3	AD4/P0.4/KSI4
4	AD3/P0.3/KSI3
5	AD2/P0.2/KSI2
6	AD1/P0.1/KSI1
7	AD0/P0.0/KSI0
8	V_{SSP}
9	V_{CC}
10	P3.0/OVRI#††
11	P3.1/SOF#
12	P3.2/INT0#
13	P3.3/INT1#
14	P3.4/T0/KSO16
15	P3.5/T1/KSO17
16	P3.6/WR#/KSO18
17	P3.7/RD#/KSO19
18	P1.0/T2/KSO0
19	P1.1/T2EX/KSO1
20	P1.2/KSO2
21	P1.3/KSO3
22	P1.4/KSO4

Pin	Name
23	P1.5/KSO5
24	P1.6/RXD/KSO6
25	P1.7/TXD/KSO7
26	LED3
27	LED2
28	XTAL1
29	XTAL2
30	AV_{CC}
31	RST
32	PLLSEL
33	LED1
34	LED0
35	Reserved (NC)†/D_{M2}††
36	Reserved (NC)†/D_{P2}††
37	Reserved (NC)†/D_{M3}††
38	Reserved (NC)†/D_{P3}††
39	V_{SS}
40	V_{CCP}
41	V_{SSP}
42	ECAP
43	D_{M0}
44	D_{P0}

Pin	Name
45	Reserved (NC)†/D_{M5}††
46	Reserved (NC)†/D_{P5}††
47	Reserved (NC)†/D_{M4}††
48	Reserved (NC)†/D_{P4}††
49	V_{SSP}
50	FSSEL†/UPWEN#††
51	PSEN#
52	ALE
53	EA#
54	V_{CCP}
55	V_{SS}
56	A15/P2.7/KSO15
57	A14/P2.6/KSO14
58	A13/P2.5/KSO13
59	A12/P2.4/KSO12
60	A11/P2.3/KSO11
61	A10/P2.2/KSO10
62	A9/P2.1/KSO9
63	A8/P2.0/KSO8
64	AD7/P0.7/KSI7

† Specific to the 8*x*931AA

†† Specific to the 8*x*931HA

Table B-4. 68-pin PLCC Signal Assignments Arranged by Functional Category

Address & Data	
Name	**Pin**
A15/P2.7/KSO15	2
A14/P2.6/KSO14	3
A13/P2.5/KSO13	4
A12/P2.4/KSO12	5
A11/P2.3/KSO11	6
A10/P2.2/KSO10	7
A9/P2.1/KSO9	8
A8/P2.0/KSO8	9
AD7/P0.7/KSI7	10
AD6/P0.6/KSI6	11
AD5/P0.5/KSI5	12
AD4/P0.4/KSI4	13
AD3/P0.3/KSI3	14
AD2/P0.2/KSI2	15
AD1/P0.1/KSI1	16
AD0/P0.0/KSI0	17

Input/Output	
Name	**Pin**
P1.0/T2/KSO0	28
P1.1/T2EX/KSO1	29
P1.2/KSO2	30
P1.3/KSO3	31
P1.4/KSO4	32
P1.5/KSO5	33
P1.6/KSO6	34
P1.7/KSO7	35
P3.0/ OVRI#††	20
P3.1/SOF#	21
P3.2/INT0#	22
P3.3/INT1#	23
P3.4/T0/KSO16	24
P3.5/T1/KSO17	25
P3.6/WR#/KSO18	26
P3.7/RD#/KSO19	27

USB	
Name	**Pin**
PLLSEL	42
D_{M0}	54
D_{P0}	55
Reserved†/ D_{M5}††	57
Reserved†/ D_{P5}††	58
Reserved†/ D_{M2}††	45
Reserved†/ D_{P2}††	46
Reserved†/ D_{M3}††	48
Reserved†/ D_{P3}††	49
ECAP	53
Reserved†/ D_{M4}††	59
Reserved†/ D_{P4}††	60
FSSEL†/UPWEN#††	64
OVRI#††	20

Processor Control	
Name	**Pin**
P3.2/INT0#	22
P3.3/INT1#	23
RST	41
XTAL1	38
XTAL2	39

Power & Ground	
Name	**Pin**
V_{CCP}	51, 68
V_{CC}	19,56
AV_{CC}	40
V_{SS}	1,50
V_{SSP}	18,47, 52,63

Bus Control & Status	
Name	**Pin**
P3.6/WR#/KSO18	26
P3.7/RD#/KSO19	27
PSEN#	65
ALE	66
EA#	67

† Specific to the 8***x***931AA

†† Specific to the 8***x***931HA

Table B-5. 64-pin SDIP Signal Assignments Arranged by Functional Category

Address & Data	
Name	**Pin**
A15/P2.7/KSO15	48
A14/P2.6/KSO14	49
A13/P2.5/KSO13	50
A12/P2.4/KSO12	51
A11/P2.3/KSO11	52
A10/P2.2/KSO10	53
A9/P2.1/KSO9	54
A8/P2.0/KSO8	55
AD7/P0.7/KSI7	56
AD6/P0.6/KSI6	57
AD5/P0.5/KSI5	58
AD4/P0.4/KSI4	59
AD3/P0.3/KSI3	60
AD2/P0.2/KSI2	61
AD1/P0.1/KSI1	62
AD0/P0.0/KSI0	63

Input/Output	
Name	**Pin**
P1.0/T2/KSO0	10
P1.1/T2EX/KSO1	11
P1.2/KSO2	12
P1.3/KSO3	13
P1.4/KSO4	14
P1.5/KSO5	15
P1.6/RXD/KSO6	16
P1.7/TXD/KSO7	17
P3.0/OVRI#	2
P3.1/SOF#	3
P3.2/INT0#	4
P3.3/ INT1#	5
P3.4/T0/KSO16	6
P3.5/T1/KSO17	7
P3.6/WR#/KSO18	8
P3.7/RD#/KSO19	9

USB	
Name	**Pin**
PLLSEL	24
D_{M0}	35
D_{P0}	36
D_{M5}	37
D_{P5}	38
D_{M2}	27
D_{P2}	28
D_{M3}	29
D_{P3}	30
ECAP	34
UPWEN#	42
OVRI#	2
D_{M4}	39
D_{P4}	40

Processor Control	
Name	**Pin**
P3.2/ INT0#	4
P3.3/ INT1#	5
RST	23
XTAL1	20
XTAL2	21

Power & Ground	
Name	**Pin**
V_{CC}	1
V_{CCP}	32,46
AV_{CC}	22
V_{SS}	31,47
V_{SSP}	33,41, 64

Bus Control & Status	
Name	**Pin**
P3.6/WR#/KSO18	8
P3.7/RD#/KSO19	9
PSEN#	43
ALE	44
EA#	45

Table B-6. 64-pin QFP Signal Assignments Arranged by Functional Category

Address & Data	
Name	**Pin**
AD6/P0.6/KSI6	1
AD5/P0.5/KSI5	2
AD4/P0.4/KSI4	3
AD3/P0.3/KSI3	4
AD2/P0.2/KSI2	5
AD1/P0.1/KSI1	6
AD0/P0.0/KSI0	7
A15/P2.7/KSO15	56
A14/P2.6/KSO14	57
A13/P2.5/KSO13	58
A12/P2.4/KSO12	59
A11/P2.3/KSO11	60
A10/P2.2/KSO10	61
A9/P2.1/KSO9	62
A8/P2.0/KSO8	63
AD7/P0.7/KSI7	64

Input/Output	
Name	**Pin**
P3.0/OVRI#††	10
P3.1/SOF#	11
P3.2/INT0#	12
P3.3/INT1#	13
P3.4/T0/KSO16	14
P3.5/T1/KSO17	15
P3.6/WR#/KSO18	16
P3.7/RD#/KSO19	17
P1.0/T2/KSO0	18
P1.1/T2EX/KSO1	19
P1.2/KSO2	20
P1.3/KSO3	21
P1.4/KSO4	22
P1.5/KSO5	23
P1.6/RXD/KSO6	24
P1.7/TXD/KSO7	25

USB	
Name	**Pin**
PLLSEL	32
ECAP	42
D_{M0}	43
D_{P0}	44
FSSEL†/UPWEN#††	50
OVRI#††	10
D_{P2}††	36
D_{M2}††	35
D_{P3}††	38
D_{M3}††	37
D_{P4}††	48
D_{M4}††	47
D_{P5}††	46
D_{M5}††	45

Processor Control	
Name	**Pin**
XTAL1	28
XTAL2	29
RST	31
P3.2/INT0#	12
P3.3/INT1#	13

Power & Ground	
Name	**Pin**
V_{CC}	9
AV_{CC}	30
V_{SS}	39,55
V_{CCP}	40,54
V_{SSP}	8, 41,49

Bus Control & Status	
Name	**Pin**
PSEN#	51
ALE	52
EA#	53
WR#	16
RD#	17

† Specific to the 8***x***931AA

†† Specific to the 8***x***931HA

Table B-7. Signal Description (Sheet 1 of 3)

Signal Name	Type	Description	Alternate Function
A15:8	O	**Address Lines**. Upper byte of external memory address.	P2.7:0/KS08:15
AD7:0	I/O	**Address/Data Lines**. Lower byte of external memory address multiplexed with data	P0.7:0/KSI0:7
ALE	O	**Address Latch Enable**. ALE signals the start of an external bus cycle and indicates that valid address information is available on lines A15:8 and AD7:0. An external latch can use ALE to demultiplex the address from the address/data bus.	—
AV_{CC}	PWR	**Analog V_{CC}**. A separate V_{CC} input for the phase-locked loop circuitry.	—
D_{M0}, D_{P0}	I/O	**USB Port 0**. Root port. Upstream port to the host PC. D_{P0} and D_{M0} are the differential data plus and data minus signals of USB port 0. These lines do not have internal pullup resistors. Provide an external 1.5 KΩ pullup resistor at D_{P0} so the device indicates to the host that it is a full-speed device. **NOTE:** D_{P0} low AND D_{M0} low signals an SE0 (USB reset), causing the 8*x*931 to stay in reset.	—
D_{M2}, D_{P2} D_{M3}, D_{P3} D_{M4}, D_{P4} D_{M5}, D_{P5}	I/O	**USB External Downstream Ports 2, 3, 4, 5.** These pins are the differential data plus and data minus lines for the four USB external downstream ports. These lines do not have internal pulldown resistors. Provide an external 15 KΩ pulldown resistor at each of these pins. If a port is not used, it is still required to pull these 2 pins low externally (similar to a disconnect) so that the inputs are not floated.	—
EA#	I	**External Access**. Directs program memory accesses to on-chip or off-chip code memory. For EA# strapped to ground, all program memory accesses are off-chip. For EA# strapped to V_{CC}, program accesses on-chip ROM if the address is within the range of the on-chip ROM; otherwise the access is off-chip. The value of EA# is latched at reset. For devices without on-chip ROM, EA# must be strapped to ground.	—
ECAP	I	**External Capacitor**. Connect a 1 µF or larger capacitor between this pin and V_{SS} to ensure proper operation of the differential line drivers.	—
FSSEL		**Full Speed Select.** Applies to the 8***x*931AA** only. If this pin is high, full speed USB data rate is selected (12Mbps). If pin is low, low speed USB data rate is selected (1.5 Mbps). Refer to **Table E-3 on page E-9**.	—
INT1:0#	I	**External Interrupts 0 and 1**. These inputs set the IE1:0 interrupt flags in the TCON register. Bits IT1:0 in TCON select the triggering method: edge-triggered (high-to-low) or level triggered (active low). INT1:0 also serves as external run control for timer1:0 when selected by GATE1:0# in TCON.	P3.3:2
KSI7:0	I	**Keyboard Scan Input.** Schmitt-trigger inputs with firmware-enabled internal pullup resistors used for the input side of the keyboard scan matrix.	AD7:0/P0.7:0

Table B-7. Signal Description (Sheet 2 of 3)

Signal Name	Type	Description	Alternate Function
KSO19 KSO18 KSO17:16 KSO15:8 KSO7:0	O	**Keyboard Scan Output.** Quasi-bidirectional ports with weak internal pullup resistors used for the output side of the keyboard scan matrix.	P3.7/RD# P3.6/WR# P3.5:4/T1:0 A15:8/P2.7:0 P1.7:0
LED3:0	O	**LED Drivers.** Designed to drive LEDs connected directly to V_{CC}. The current each driver is capable of sinking is given as V_{OL2} in the datasheet.	—
OVRI#	I	**Overcurrent Sense**. Sense input to indicate an overcurrent condition for a bus-powered USB device on an external down-stream port. Active low with an internal pullup.	P3.0
P0.7:0	I/O	**Port 0**. Eight-bit, open-drain, bidirectional I/O port. Port 0 pins have Schmitt trigger inputs.	AD7:0/KSI7:0
P1.7:0	I/O	**Port 1**. Eight-bit quasi-bidirectional I/O port with internal pullups.	KSO7:0
P2.7:0	I/O	**Port 2**. Eight-bit quasi-bidirectional I/O port with internal pullups.	A15:8/KSO15:8
P3.0 P3.1 P3.2 P3.3 P3.4 P3.5 P3.6 P3.7	I/O	**Port 3**. Eight-bit quasi-bidirectional I/O port with internal pullups.	OVRI# SOF# INT0# INT1# T0/KSO16 T1/KSO17 WR#/KSO18 RD#/KSO19
PLLSEL	I	**Phase-locked Loop Select**. For normal operation using the 8***x***931HA, connect PLLSEL to logic high. PLLSEL = 0 is used for factory test (see Table 2-3 on page 2-9). For 8***x***931AA operation, see Table E-3 on page E-9.	—
PSEN#	O	**Program Store Enable**. Read signal output. Asserted for read accesses to external program memory.	—
RD#	O	**Read.** Read signal output. Asserted for read accesses to external data memory.	P3.7/KSO19
RXD	I/O	**Receive Serial Data**. RXD sends and receives data in serial I/O mode 0 and receives data in serial I/O modes 1, 2, and 3.	P1.6
RST	I	**Reset**. Reset input to the chip. Holding this pin high for 64 oscillator periods while the oscillator is running resets the device. The port pins are driven to their reset conditions when a voltage greater than V_{IH1} is applied, whether or not the oscillator is running. This pin has an internal pulldown resistor which allows the device to be reset by connecting a capacitor between this pin and V_{CC}. Asserting RST when the chip is in idle mode or powerdown mode returns the chip to normal operation.	—

Table B-7. Signal Description (Sheet 3 of 3)

Signal Name	Type	Description	Alternate Function
SOF#	O	**Start of Frame**. Start of frame pulse. Active low. Asserted for 8 states when frame timer is locked to USB frame timing and SOF token or artificial SOF is detected.	P3.1
T1:0	I	**Timer 1:0 External Clock Input**. When timer 1:0 operates as a counter, a falling edge on the T1:0 pin increments the count.	P3.5:4/KSO17:16
T2	I/O	**Timer 2 Clock Input/Output**. For the timer 2 capture mode, this signal is the external clock input. For the clock-out mode, it is the timer 2 clock output.	P1.0
T2EX	I	**Timer 2 External Input**. In timer 2 capture mode, a falling edge initiates a capture of the timer 2 registers. In auto-reload mode, a falling edge causes the timer 2 registers to be reloaded. In the up-down counter mode, this signal determines the count direction: 1 = up, 0 = down.	P1.1
TXD	O	**Transmit Serial Data**. TXD outputs the shift clock in serial I/O mode 0 and transmits serial data in serial I/O modes 1, 2, and 3.	P1.7
UPWEN#	O	**USB Power Enable**. A low signal on this pin applies power to the external downstream ports.	—
V_{CC}	PWR	**Supply Voltage**. Connect this pin to the +5V supply voltage.	—
V_{CCP}	PWR	**Supply Voltage for I/O buffers**. Connect this pin to the +5V supply voltage.	—
V_{SS}	GND	**Circuit Ground**. Connect this pin to ground.	—
V_{SSP}	GND	**Circuit Ground for I/O buffers**. Connect this pin to ground.	—
WR#	O	**Write.** Write signal output to external memory.	P3.6/KSO19
XTAL1	I	**Oscillator Amplifier Input**. When implementing the on-chip oscillator, connect the external crystal or ceramic resonator across XTAL1 and XTAL2. If an external clock source is used, connect it to this pin.	—
XTAL2	O	**Oscillator Amplifier Output**. When implementing the on-chip oscillator, connect the external crystal or ceramic resonator across XTAL1 and XTAL2. If an external oscillator is used, leave XTAL2 unconnected.	—

C

Registers

APPENDIX C
REGISTERS

This appendix contains reference information regarding the 8*x*931 special function registers (SFRs). The SFR memory map in Table C-1 gives the address of each SFR and its contents following chip reset. An "x" indicates the bit value following chip reset is indeterminate.

Blank locations in Table C-1 are not implemented, i.e., no register exists. If an instruction attempts to write to an unimplemented SFR location, the instruction executes, but nothing is actually written. If an unimplemented SFR location is read, it returns an unspecified value.

SFRs shown with double borders are endpoint-indexed. Endpoint-indexed SFRs are implemented as banks of registers similar to registers R0-R7. There is a set or bank of registers for each endpoint pair. Endpoint-indexed SFRs are accessed by means of the SFR address and an index value. The EPINDEX register specifies hub/function and the endpoint number (which serves as the index value). See "Endpoint-indexed SFRs" on page 6-5 and "Hub Endpoint Indexing Using EPINDEX" on page 7-11.

SFRs shown with bold borders are port-indexed. Port-indexed SFRs are implemented as banks of registers similar to registers R0-R7. There is a set or bank of port-indexed SFRs for each USB downstream port. Port-indexed SFRs are accessed by means of the SFR address and an index value. The HPINDEX register contains the port number which serves as the index value. See "Hub Port Indexing Using HPINDEX" on page 7-23.

NOTE

The 8*x*931HA uses a different SFR map than the 8*x*931AA. See Table E-4 on page E-10 for the 8*x*931AA SFR map.

Tables C-2 through C-6 list the SFRs by functional category. Register definition tables which describe the SFRs and define the bits can be found arranged alphabetically in "SFR Descriptions" on page C-6.

Table C-1. 8*x*931HA SFR Map

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
F8	KBCON 0xx00000								FF
F0	B 00000000	EPINDEX 1xxxxx00	TXSTAT 0xx00000	TXDAT†† xxxxxxxx†	TXCON 0xxx 0100†	TXFLG 00xx1000	TXCNTL xxxx xxxx†		F7
E8	HIFLG xxxxxx00								EF
E0	ACC 00000000	EPCON 00d1 0d0d†	RXSTAT 00000000	RXDAT xxxxxxxx	RXCON 0xx0 0100	RXFLG 00xx1000	RXCNTL xxxx xxxx†		E7
D8								PCON1 xxxx x000	DF
D0	PSW 00000000		SOFL 00000000	SOFH 00001000	HPINDEX xxxxx000	HPSC xxx00000		HPSTAT 100d 0000†	D7
C8	T2CON 00000000	T2MOD xxxx xx00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000		HPCON xxxxx000	CF
C0	FIFLG xx00 0000								C7
B8	IPL0 x0000000	SADEN 00000000							BF
B0	P3 11111111	IEN1 0000 0000	IPL1 0000 0000	IPH1 0000 0000				IPH0 x0000000	B7
A8	IEN0 00000000	SADDR 00000000					HSTAT 0000 0000		AF
A0	P2 11111111	HIE xxxxxx00	FIE xx00 0000						A7
98	SCON 0000 0000	SBUF xxxxxxxx	HPPWR xx00001x						9F
90	P1 11111111							HADDR 00000000	97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		FADDR 00000000	8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 001d 0000	87
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

MCS 51 microcontroller SFRs

Port-indexed SFRs

Endpoint-indexed SFRs

† For EPCON, TXCON, TXDAT, TXCNTL, and RXCNTL the reset value depends on the endpoint pair selected. Refer to the register definition tables in Appendix C or Chapter 5, "USB Function."

†† For hub endpoint 1 (EPINDEX = 1000 0001), the only endpoint SFR implemented is TXDAT. A separate TXDAT register definition table is provided for this endpoint (see Figure 6-8 on page 6-16).

NOTE: "d" in the SFR reset value denotes configuration/operation dependence. Refer to specific SFR descriptions for more details.

C.1 SFRS BY FUNCTIONAL CATEGORY

Table C-2. Core SFRs

Mnemonic	Name	Address
ACC	Accumulator	E0H
B	B Register	F0H
DPTR	Data Pointer (2 bytes)	—
DPL	Low Byte of DPTR	82H
DPH	High Byte of DPTR	83H
KBCON	Keyboard Control	F8H
PCON	Power Control	87H
PCON1	USB Power Control.	DFH
PSW	Program Status Word	D0H
SP	Stack Pointer	81H

Table C-3. Interrupt System SFRs

Mnemonic	Description	Address
FIE	USB Function Interrupt Enable Register.	A2H
FIFLG	USB Function Interrupt Flag Register.	C0H
HIE	Hub Interrupt Enable Register.	A1H
HIFLG	Hub Interrupt Flag Register.	E8H
IEN0	Interrupt Enable Register 0.	A8H
IEN1	Interrupt Enable Register1.	B1H
IPL0	Interrupt Priority Low Register 0.	B8H
IPH0	Interrupt Priority High Register 0.	B7H
IPL1	Interrupt Priority Low Register 1.	B2H
IPH1	Interrupt Priority High Register 1.	B3H
KBCON	Keyboard Control Register.	F8H
SOFH	Start of Frame High Register.	D3H
SOFL	Start of Frame Low Register.	D2H

Table C-4. I/O Port SFRs

Mnemonic	Name	Address
P0	Port 0	80H
P1	Port 1	90H
P2	Port 2	A0H
P3	Port 3	B0H

Table C-5. Serial I/O SFRs

Mnemonic	Name	Address
SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
SADEN	Slave Address Mask	B9H
SADDR	Slave Address	A9H

Table C-6. USB Function SFRs

Mnemonic	Name	Address
EPCON	Endpoint Control Register.	E1H
EPINDEX	Endpoint Index Register.	F1H
FADDR	Function Address Register.	8FH
RXCNTL	Receive FIFO Byte-Count Low Register.	E6H
RXCON	Receive FIFO Control Register.	E4H
RXDAT	Receive FIFO Data Register.	E3H
RXFLG	Receive FIFO Flag Register.	E5H
RXSTAT	Endpoint Receive Status Register.	E2H
TXCNTL	Transmit Count Low Register.	F6H
TXCON	Transmit FIFO Control Register.	F4H
TXDAT	Transmit FIFO Data Register.	F3H
TXFLG	Transmit Flag Register.	F5H
TXSTAT	Endpoint Transmit Status Register.	F2H

Table C-7. USB Hub SFRs

Mnemonic	Name	Address
HADDR	Hub Address Register.	97H
HPCON	Hub Port Control.	CFH
HPINDEX	Hub Port Index Register.	D4H
HPPWR	Hub Port Power Control.	9AH
HPSC	Hub Port Status Change.	D5H
HPSTAT	Hub Port Status.	D7H
HSTAT	Hub Status and Configuration.	AEH

Table C-8. Timer/Counter SFRs

Mnemonic	Name	Address
TL0	Timer/Counter 0 Low Byte	8AH
TH0	Timer/Counter 0 High Byte	8CH
TL1	Timer/Counter 1 Low Byte	8BH
TH1	Timer/Counter 1 High Byte	8DH
TL2	Timer/Counter 2 Low Byte	CCH
TH2	Timer/Counter 2 High Byte	CDH
TCON	Timer/Counter 0 and 1 Control	88H
TMOD	Timer/Counter 0 and 1 Mode Control	89H
T2CON	Timer/Counter 2 Control	C8H
T2MOD	Timer/Counter 2 Mode Control	C9H
RCAP2L	Timer 2 Reload/Capture Low Byte	CAH
RCAP2H	Timer 2 Reload/Capture High Byte	CBH

C.2 SFR DESCRIPTIONS

This section contains descriptions of all 8*x*931 SFRs. They are presented in alphabetical order.

NOTE

SFR bits are firmware read/write unless otherwise noted in the bit definition. SFRs may be accessed only as bytes; they may not be accessed as words.

ACC

Address: E0H
Reset State: 0000 0000B

Accumulator. ACC provides SFR access to the accumulator. Instructions in the MCS® 51 architecture use the accumulator as both source and destination for calculations and moves.

7							0
Accumulator Contents							

Bit Number	Bit Mnemonic	Function
7:0	ACC.7:0	Accumulator.

B

Address: F0H
Reset State: 0000 0000B

B Register. The B register is used during multiply and divide operations. For other instructions, it can be treated as another scratch pad register.

7							0
B Register Contents							

Bit Number	Bit Mnemonic	Function
7:0	B.7:0	B Register.

DPH

Address: 83H
Reset State: 0000 0000B

Data Pointer High. DPH is the upper byte of the 16-bit data pointer, DPTR. Instructions in the MCS® 51 architecture use DPTR for data moves, code moves, and for a jump instruction (JMP @A+DPTR). See also DPL.

7							0
DPH Contents							

Bit Number	Bit Mnemonic	Function
7:0	DPH.7:0	Data Pointer High: Bits 8–15 of the data pointer.

DPL

Address: 82H
Reset State: 0000 0000B

Data Pointer Low. DPL is the low byte of the 16-bit data pointer, DPTR. Instructions in the MCS® 51 architecture use the 16-bit data pointer for data moves, code moves, and for a jump instruction (JMP @A+DPTR). See also DPH.

7							0
DPL Contents							

Bit Number	Bit Mnemonic	Function
7:0	DPL.7:0	Data Pointer Low: Bits 0–7 of the data pointer.

EPCON (Endpoint-indexed)

Address: E1H

Reset State: Endpoint 0 0011 0101B; Function Endpoints 1, 2 0001 0000B

Endpoint Control Register. This SFR configures the operation of the endpoint specified by EPINDEX.

7							0
RXSTL	TXSTL	CTLEP	RXSPM	RXIE	RXEPEN	TXOE	TXEPEN

Bit Number	Bit Mnemonic	Function
7	RXSTL	Stall Receive Endpoint: Set this bit to stall the receive endpoint. Clear this bit only when the host has intervened through commands sent down endpoint 0. When this bit is set and RXSETUP is clear, the receive endpoint will respond with a STALL handshake to a valid OUT token. When this bit is set and RXSETUP is set, the receive endpoint will NAK. This bit does not affect the reception of SETUP tokens by a control endpoint.
6	TXSTL	Stall Transmit Endpoint: Set this bit to stall the transmit endpoint. This bit should be cleared only when the host has intervened through commands sent down endpoint 0. When this bit is set and RXSETUP is clear, the receive endpoint will respond with a STALL handshake to a valid IN token. When this bit is set and RXSETUP is set, the receive endpoint will NAK.
5	CTLEP	Control Endpoint:† Set this bit to configure the endpoint as a control endpoint. Only control endpoints are capable of receiving SETUP tokens.
4	RXSPM	Receive Single Packet Mode:† Set this bit to configure the receive endpoint for single data packet operation. When enabled, only a single data packet is allowed to reside in the receive FIFO. **NOTE:** For control endpoints (CTLEP=1), this bit should be set for single packet mode operation as the recommended firmware model. However, it is possible to have a control endpoint configured in dual packet mode as long as the firmware handles the endpoint correctly.
3	RXIE	Receive Input Enable: Set this bit to enable data from the USB to be written into the receive FIFO. If cleared, the endpoint will not write the received data into the receive FIFO and at the end of reception, but will return a NAK handshake on a valid OUT token if the RXSTL bit is not set.This bit does not affect a valid SETUP token. A valid SETUP token and packet override this bit if it is cleared, and place the receive data in the FIFO.

† For hub endpoint 0 (EPINDEX = 1000 0000), bits 5 and 4 are hard-wired to '1' since hub endpoint 0 is always a control endpoint.

EPCON (Continued) (Endpoint-indexed)

Address: E1H
Reset State: Endpoint 0 0011 0101B
Function Endpoints 1, 2 0001 0000B

Endpoint Control Register. This SFR configures the operation of the endpoint specified by EPINDEX.

7 **0**

RXSTL	TXSTL	CTLEP	RXSPM	RXIE	RXEPEN	TXOE	TXEPEN

Bit Number	Bit Mnemonic	Function
2	RXEPEN	Receive Endpoint Enable: Set this bit to enable the receive endpoint. When disabled, the endpoint does not respond to a valid OUT or SETUP token. This bit is hardware read-only and has the highest priority among RXIE and RXSTL. Note that endpoint 0 is enabled for reception upon reset.
1	TXOE	Transmit Output Enable: This bit is used to enable the data in TXDAT to be transmitted. If cleared, the endpoint returns a NAK handshake to a valid IN token if the TXSTL bit is not set.
0	TXEPEN	Transmit Endpoint Enable: This bit is used to enable the transmit endpoint. When disabled, the endpoint does not respond to a valid IN token. This bit is hardware read only. Note that endpoint 0 is enabled for transmission upon reset.

† For hub endpoint 0 (EPINDEX = 1000 0000), bits 5 and 4 are hard-wired to ‘1’ since hub endpoint 0 is always a control endpoint.

EPINDEX Address: F1H
Reset State: 1xxx xx00B

Endpoint Index Register. This register identifies the endpoint pair. Its contents select the transmit and receive FIFO pair and serve as an index to endpoint-specific SFRs.

7 **0**

HORF	—	—	—	—	—	EPINX1	EPINX0

Bit Number	Bit Mnemonic	Function
7	HORF	Hub/Function Bit: 1 = Hub. Selects USB hub FIFOs and SFRs. 0 = Function. Selects USB function FIFOs and SFRs.
6:2	—	Reserved: Write zeros to these bits.
1:0	EPINX1:0	Endpoint Index: **EPINDEX**† — Function Endpoint: 0xxx xx00 — Function Endpoint 0 0xxx xx01 — Function Endpoint 1 0xxx xx10 — Function Endpoint 2 **EPINDEX**† — Hub Endpoint: 1xxx xx00 — Hub Endpoint 0 1xxx xx01 — Hub Endpoint 1

† The EPINDEX register identifies the endpoint pair and selects the associated transmit and receive FIFO pair. The value in this register plus SFR addresses select the associated bank of endpoint-indexed SFRs (TXDAT, TXCON, TXFLG, TXCNTL, RXDAT, RXCON, RXFLG, RXCNTL, EPCON, TXSTAT, and RXSTAT).

FADDR

Address: 8FH
Reset State:0000 0000B

Function Address Register. This SFR holds the address for the USB function. During bus enumeration it is written with a unique value assigned by the host.

7	0
—	A6:0

Bit Number	Bit Mnemonic	Function
7	—	Reserved: Write a zero to this bit.
6:0	A6:0	7-bit Programmable Function Address: This register is programmed through the commands received via endpoint 0 on configuration, which should be the only time the firmware should change the value of this register. This register is read-only by hardware.

FIE

Address: A2H
Reset State: xx00 0000B

Function Interrupt Enable Register. Enables and disables the receive and transmit done interrupts for the three function endpoints.

7 **0**

—	—	FRXIE2	FTXIE2	FRXIE1	FTXIE1	FRXIE0	FTXIE0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	FRXIE2	Function Receive Interrupt Enable 2: Enables the receive done interrupt for endpoint 2 (FRXD2).
4	FTXIE2	Function Transmit Interrupt Enable 2: Enables the transmit done interrupt for endpoint 2 (FTXD2).
3	FRXIE1	Function Receive Interrupt Enable 1: Enables the receive done interrupt for endpoint 1 (FRXD1).
2	FTXIE1	Function Transmit Interrupt Enable 1: Enables the transmit done interrupt for endpoint 1 (FTXD1).
1	FRXIE0	Function Receive Interrupt Enable 0: Enables the receive done interrupt for endpoint 0 (FRXD0).
0	FTXIE0	Function Transmit Interrupt Enable 0: Enables the transmit done interrupt for endpoint 0 (FTXD0).

NOTE: For all bits, a '1' means the interrupt is enabled and will cause an interrupt to be signaled to the microcontroller. A '0' means the associated interrupt source is disabled and cannot cause an interrupt, even though the interrupt bit's value is still reflected in the FIFLG register.

FIFLG

Address: C0H
Reset State: xx00 0000B

Function Interrupt Flag Register. Contains the USB function's transmit and receive done interrupt flags for non-isochronous endpoints.

7 **0**

—	—	FRXD2	FTXD2	FRXD1	FTXD1	FRXD0	FTXD0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	FRXD2	Function Receive Done Flag, Endpoint 2
4	FTXD2	Function Transmit Done Flag, Endpoint 2
3	FRXD1	Function Receive Done Flag, Endpoint 1
2	FTXD1	Function Transmit Done Flag, Endpoint 1
1	FRXD0	Function Receive Done Flag, Endpoint 0
0	FTXD0	Function Transmit Done Flag, Endpoint 0

NOTES:
1. For all bits in the Interrupt Flag Register, a '1' indicates that an interrupt is actively pending; a '0' indicates that the interrupt is not active. The interrupt status is shown regardless of the state of the corresponding interrupt enable bit in the FIE.
2. Bits are set-only by hardware and clearable in firmware. Firmware can also set the bits for test purposes, allowing the interrupt to be generated in firmware. This SFR is bit-addressable.
3. A set bit indicates either:
   Valid data waiting to be serviced in the RX FIFO for the indicated endpoint and that the data was received without error and has been acknowledged; or
   Data was received with a Receive Data Error requiring firmware intervention to be cleared.

HADDR Address: 97H
Reset State: 0000 0000B

Hub Address Register. This SFR holds the address for the hub device. During bus enumeration it is written with a unique value assigned by the host.

7							0
—	Hub Address						

Bit Number	Function
7	Reserved. Write a zero to this bit.
6:0	Hub address register: Updated using a SET_ADDRESS USB host request. This address is used by the HIU to perform token decoding.

HIE Address: A1H
Reset State: xxxx xx00B

Hub Interrupt Enable Register. Enables and disables the receive and transmit done interrupts for hub endpoint 0.

7							**0**
—	—		—	—	—	HRXE0	HTXE0

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved: Write zeros to these bits.
1	HRXE0	HRXE0: Enable the hub endpoint 0 receive done interrupt (HRXD0).†
0	HTXE0	HTXE0: Enable the hub endpoint 0 transmit done interrupt (HTXD0).†

† For both bits, a '1' means the interrupt is enabled, and will cause an interrupt to be signaled to the microcontroller. A '0' means the associated interrupt source is disabled and cannot cause an interrupt, even though its value is still reflected in the HIFLG register.

HIFLG

Address: E8H
Reset State: xxxx xx00B

Hub Interrupt Flag Register. Contains the hub's transmit and receive done interrupt flags for hub endpoint 0.

7							0
—	—		—	—	—	HRXD0	HTXD0

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved: Write zeros to these bits.
1	HRXD0	Hub Receive Done, Endpoint 0.: Hardware sets this bit to indicate that there is either: (1) valid data waiting to be serviced in the receive data buffer for hub endpoint 0 and that the data was received without error and has been acknowledged; or (2) that data was received with a FIFO error requiring firmware intervention to be cleared.
0	HTXD0	Hub Transmit Done, Endpoint 0: Hardware sets this bit to indicate that one of two conditions exists in the transmit data buffer for hub endpoint 0: (1) the transmit data has been transmitted and the host has sent acknowledgment which was successfully received; or (2) a FIFO-related error occurred during transmission of the data packet, which requires servicing by firmware to be cleared.

NOTES:
1. Note that because the HIFLG appears in the first SFR column, it is a bit-addressable SFR. All bits are set in hardware and cleared by firmware. Firmware can also set these bits for test purposes, allowing the interrupt to be generated by firmware.
2. For both HRXD0 and HTXD0, a '1' indicates that an interrupt is actively pending; a '0' indicates that the interrupt is not active. The interrupt status is shown regardless of the state of the corresponding interrupt enable bit in the HIE.

HPCON
(Indexed by HPINDEX)

Address: CFH
Reset State:xxxx x000B

Hub Port Control Register. Firmware writes to this register to disable, enable, reset, suspend, and resume a port.

7							0
—	—	—	—	—	HPCON2	HPCON1	HPCON0

Bit Number	Bit Mnemonic	Function
7:3	—	Reserved: Write zeros to these bits.
2:0	HPCON.2:0	Encoded Hub Port Control Commands: All bits should be set and cleared by firmware after receiving the USB requests ClearPortFeature and SetPortFeature from the host. The bits are encoded as follows (all other bit combinations are ignored by the hardware): 000 — Disable port 001 — Enable port 010 — Reset and enable port 011 — Suspend port 100 — Resume port See Table 7-6 on page 7-16 for a complete description of the encoded hub port control commands.

HPINDEX

Address D4H
Reset State xxxx x000B

Hub Port Index Register. This register contains the binary value of the port whose HPSC, HPSTAT, and HPCON registers are to be accessed.

7							0
—	—	—	—	—	HPIDX2	HPIDX1	HPIDX0

Bit Number	Bit Mnemonic	Function
7:3	—	Reserved: Write zeros to these bits.
2:0	HPIDX.2:0	Port Index Select: Used to select the port to be indexed by the following registers: HPSC, HPSTAT, and HPCON. This register is hardware read-only. The ports are addressed using the following HPIDX2:0 bit combinations: Port 1 = “001” (internal port) Port 2 = “010” Port 3 = “011” Port 4 = “100” Port 5 = “101”

NOTE: Port 0 = “000” (the root port) and all other combinations not shown above are not valid port indexes and are ignored.

HPPWR Address: 9AH
Reset State: xx00 001x

Hub Port Power Control Register. This register is used to control power to the hub's downstream ports.

7							0
—	—	HPPWR5	HPPWR4	HPPWR3	HPPWR2	HPPWR1	—

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5:2	HPPWR5:2	Port Power Control for USB Ports 5-2: Bit 5 is power control for port 5, bit 4 for port 4, bit 3 for port 3, and bit 2 for port 2. These bits are set and cleared by firmware via a USB host request SetPortFeature with the PORT_POWER feature selector. These bits will also be cleared by hardware upon detection of an over-current condition. This is done to prevent oscillation of the UPWEN# pin during an over-current condition with bus-powered devices. A value of '1' enables power to the downstream port and puts the port in a disconnected state. A value of '0' turns the downstream port power off. **NOTE:** The UPWEN# pin is set to '1' only if all port power enable bits are '0,' due to the use of a ganged (shared) power enable scheme.
1	HPPWR1	Port Power Control for USB Port 1 (read-only): Port 1 is an internal port and is always powered on. This bit is hard-wired to '1.'
0	—	Reserved: Write a zero to this bit.

HPSC
(Indexed by HPINDEX)

Address: D5H
Reset State: xxx0 0000B

Hub Port Status Change Register. This register indicates a change in status for a port, including over-current, reset, suspend, low-speed device, enable and connect status.

7							0
—	—	—	RSTSC	—	PSSC	PESC	PCSC

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved: Write zeros to these bits.
4	RSTSC	Reset Status Change (read-, clear-only): This bit is cleared in firmware via the USB host request ClearPortFeature with a C_PORT_RESET feature selector. ‘1’ indicates reset of port complete; ‘0’ indicates no change. Port ***x*** (***x***=2,3,4,5): This bit is set by hardware approximately 10 msec after receipt of a port reset and enable command (SetPortFeature with PORT_RESET feature selector). Port 1: This bit is set by hardware at the EOF2 point near the end of a frame after completion of the hardware-timed reset due to firmware execution of a port reset and enable command (SetPortFeature with PORT_RESET feature selector).
3	—	Reserved: Write a zero to this bit.
2	PSSC	Port Suspend Status Change (read-, clear-only): This bit is cleared by firmware upon a USB host request ClearPortFeature with C_PORT_SUSPEND feature selector. ‘1’ = resume process complete; ‘0’ = no change. The resume process is initiated by firmware upon reception of a SetPortFeature request with a PORT_SUSPEND feature selector. Port ***x*** (***x***=2,3,4,5): This bit is set by hardware upon completion of the firmware-initiated resume process. Port 1: This bit is set by hardware 20 msec after the next EOF2 point after completion of the resume process.

NOTE: Bits are returned as part of the second word (2 bytes) in response to a Get Port Status request from the USB host. The upper 11 MSbs are reserved and always ‘0’ per USB 1.0: 0000 0000 000.4.3.2.1.0 (MSB at left).

HPSC (Continued) Address: D5H
(Indexed by HPINDEX) Reset State: xxx0 0000B

Hub Port Status Change Register. This register indicates a change in status for a port, including over-current, reset, suspend, low-speed device, enable and connect status.

7							0
—	—	—	RSTSC	—	PSSC	PESC	PCSC

Bit Number	Bit Mnemonic	Function
1	PESC	Port Enable/Disable Status Change (read, clear-only): This bit's status does not change due to USB requests. This bit is cleared by firmware via the USB host request ClearPortFeature with a C_PORT_ENABLE feature selector. '1' indicates port enabled/ disabled status change; '0' indicates no change. Port ***x*** (***x***=2,3,4,5): This bit is set by hardware due to hardware events only (this bit indicates the port was disabled due to babble, physical disconnects, or overcurrent). Port 1: This bit is set by hardware at the EOF2 point near the end of frame due to hardware events only (e.g., the port was disabled due to babble).
0	PCSC	Port Connect Status Change (read-, clear-only): This bit is cleared by firmware via a USB host request ClearPortFeature with C_PORT_CONNECTION feature selector. '1' indicates connect status change; '0' indicates no change. Port ***x*** (***x***=2,3,4,5): This bit is set by hardware at the EOF2 point near the end of a frame due to hardware connects and disconnects. Port 1: This bit is set by hardware at the next EOF2 after completion of a hub reset (since the internal port is always connected).

NOTE: Bits are returned as part of the second word (2 bytes) in response to a Get Port Status request from the USB host. The upper 11 MSbs are reserved and always '0' per USB 1.0: 0000 0000 000.4.3.2.1.0 (MSB at left).

HPSTAT
(Indexed by HPINDEX)

Address: D7H
Reset State: 100d 0000B

Hub Port Status Register. This register indicates the current status for a port, including power, reset, suspend, low-speed device, enable, connect, D_P, and D_M status.

7							0
DPSTAT	DMSTAT	LSSTAT	PPSTAT	PRSTAT	PSSTAT	PESTAT	PCSTAT

Bit Number	Bit Mnemonic	Function
7	DPSTAT	D_P Status (read-only): Value of D_P for port ***x*** at end of last frame. Firmware must return this bit in response to a GetBusState request from the host. Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware at the EOF2 point near the end of a frame (used for diagnostics). Port 1: Hard-wired to '1', since there is no D_P signal for the embedded port
6	DMSTAT	D_M Status (read-only): Value of D_M for port ***x*** at end of last frame. Firmware must return this bit in response to a GetBusState request from the host. Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware at the EOF2 point near the end of a frame (used for diagnostics). Port 1: Hard-wired to '0', since there is no D_M signal for the embedded port.
5	PPSTAT	Port Power Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware based on the present power status of the port, as controlled either by firmware using the HPPWR register, or by an overcurrent condition in hardware. '1' = port ***x*** is powered on. '0' = port ***x*** is powered off. The port ***x*** power status is only sampled at the EOF2 point near end-of-frame. Port 1: Hard-wired to '1', since the internal function is always powered-on.
4	LSSTAT	Low-speed Device Attach Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware upon detection of the presence or absence of a low-speed device at the EOF2 point near end-of-frame. '1' = low-speed device is attached to port ***x***. '0' = full-speed device is attached to port ***x***. Port 1: Hard-wired to '0' (full-speed), since port 1 is permanently attached to the embedded USB function.

NOTES:

Firmware returns the bits of this register in the first word of the **8*x***931HA response to the host's GetPortStatus request. See "GetPortStatus Request Firmware" on page 8-25.

Overcurrent indication is not represented on a per-port basis because the **8*x***931HA supports ganged power control and overcurrent indication.

HPSTAT (Continued) Address: D7H
(Indexed by HPINDEX) Reset State: 100d 0000B

Hub Port Status Register. This register indicates the current status for a port, including power, reset, suspend, low-speed device, enable, connect, D_P, and D_M status.

7							0
DPSTAT	DMSTAT	LSSTAT	PPSTAT	PRSTAT	PSSTAT	PESTAT	PCSTAT

Bit Number	Bit Mnemonic	Function
3	PRSTAT	Port Reset Status (read-only): Port ***x*** ((***x***=2,3,4,5): Set and cleared by hardware as a result of initiating a port ***x*** reset by writing to HPCON. '1' = reset signaling is currently asserted for port ***x***. '0' = reset signaling is not asserted. Sampled only at the EOF2 point near end of frame. Port 1: Same as port ***x***.
2	PSSTAT	Port Suspend Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware as controlled by firmware via HPCON. '1' = port ***x*** is currently suspended. '0' = not suspended. Sampled only at the EOF2 point near end of frame. Port 1: Same as port ***x***.
1	PESTAT	Port Enable/Disable Status (read-only): Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware as controlled by firmware via HPCON. '1' = port ***x*** is currently enabled. '0' = port is disabled. Sampled only at the EOF2 point near end of frame. Port 1: Same as port ***x***.
0	PCSTAT	Port Connect Status (read-only): Port ***x*** connect status from previous frame time. Port ***x*** (***x***=2,3,4,5): Set and cleared by hardware after sampling the connect state at EOF2 near the end of the present frame. '1' = device is present on port ***x***. '0' = device is not present. This bit will be set if either a physical connection is detected, or during a hub reset when a downstream device is already connected. This bit will be cleared if a disconnect is detected. Port 1: Hard-wired to '1', since the internal function is permanently connected.

NOTES:

Firmware returns the bits of this register in the first word of the **8*x***931HA response to the host's GetPortStatus request. See "GetPortStatus Request Firmware" on page 8-25.

Overcurrent indication is not represented on a per-port basis because the **8*x***931HA supports ganged power control and overcurrent indication.

HSTAT

Address: AEH
Reset State: 0000 0000B

Hub Status and Configuration Register. This SFR contains bits for remote wake-up request, status and status change indicators for over-current and hub endpoint 1 stall and enable.

7							0
OVRIEN	HRWUPE	EP1STL	EP1EN	OVISC	—	OVI	—

Bit Number	Bit Mnemonic	Function
7	OVRIEN	Overcurrent Detect Enable Bit: This bit is used to gate off the overcurrent input detect which is multiplexed with P3.0. When set, a low on P3.0/OVRI# pin will trigger over current detection logic. When this bit is '0' the over current detection logic is disabled.
6	HRWUPE	Hub Remote Wake-up Enable Bit: Set if the device is currently enabled to request remote wake-up. This bit is modified through the SetFeature and ClearFeature requests using the DEVICE_REMOTE_WAKEUP feature selector. When '0,' the hub blocks resume signaling for connect/disconnect and resume events detected on downstream ports. **NOTE:** Do not set this bit until after the hub is enumerated and the host issues a SET_FEATURE command with a DEVICE_ REMOTE_WAKEUP feature selector.
5	EP1STL	Hub Endpoint 1 Stall Field: Set to '1' via the USB SetFeature request with endpoint stall feature selector. When '1,' will force a stall response when endpoint 1 is addressed. Reset with USB ClearFeature request with endpoint stall feature selector.
4	EP1EN	Hub Endpoint 1 Enable: Set to '1' upon receipt of a USB SetConfiguration request value of 0001H. Endpoint 1 cannot respond unless this bit is set. Bit is reset upon receipt of configuration value other than 0001H or a system or USB reset. **NOTE:** This bit must be set in order for the UPWEN# pin to enable power to the downstream ports. Downstream power cannot be applied until this is done.
3	OVISC	Hub Over-current Indicator Status Change (read/clear-only): † Set to '1' if change is detected in the over-current status, even if the condition goes away before it is detected by firmware. Cleared via a USB ClearFeature request with C_HUB_OVER_CURRENT feature selector. Cleared to '0' if no change.
2	—	Reserved: The value read from this bit is indeterminate. Write a zero to this bit.

† Bits 1 and 3 are returned in response to a Get Hub Status request from the USB host. This response is a four-byte field with zero padding (MSB at left): 0000 0000 0000 00[3] 0 0000 0000 0000 00[1] 0.

HSTAT (Continued)

Address: AEH
Reset State: 0000 0000B

Hub Status and Configuration Register. This SFR contains bits for remote wake-up request, status and status change indicators for over-current and hub endpoint 1 stall and enable.

7 **0**

OVRIEN	HRWUPE	EP1STL	EP1EN	OVISC	—	OVI	—

Bit Number	Bit Mnemonic	Function
1	OVI	Latest Over-current Indicator (read-only): † Hardware sets and clears this bit via the OVRI# input pin.'1' indicates an over-current condition. '0' indicates normal power operation.
0	—	Reserved: The value read from this bit is indeterminate. Write a zero to this bit.

† Bits 1 and 3 are returned in response to a Get Hub Status request from the USB host. This response is a four-byte field with zero padding (MSB at left): 0000 0000 0000 00[3] 0 0000 0000 0000 00[1] 0.

IEN0

Address: A8H
Reset State: 0000 0000B

Interrupt Enable Register 0. IEN0 contains two types of interrupt enable bits. The global enable bit (EA) enables/disables all of the interrupts (including those in IEN1). The remaining bits enable/disable the other individual interrupts.

7 **0**

EA	—	ET2	ES	ET1	EX1	ET0	EX0

Bit Number	Bit Mnemonic	Function
7	EA	Global Interrupt Enable: Setting this bit enables all interrupts that are individually enabled by the other bits of this register, as well as the interrupts enabled by the bits in the IEN1 SFR. Clearing this bit disables all interrupts, except the TRAP interrupt, which are always enabled.
6	—	Reserved: Write a zero to this bit.
5	ET2	Timer 2 Overflow Interrupt Enable: Setting this bit enables the timer 2 overflow interrupt.
4	ES	Serial I/O Port Interrupt Enable: Setting this bit enables the serial I/O port interrupt.
3	ET1	Timer 1 Overflow Interrupt Enable: Setting this bit enables the timer 1 overflow interrupt.
2	EX1	External Interrupt 1 Enable: Setting this bit enables external interrupt 1.
1	ET0	Timer 0 Overflow Interrupt Enable: Setting this bit enables the timer 0 overflow interrupt.
0	EX0	External Interrupt 0 Enable: Setting this bit enables external interrupt 0.

NOTE: Note that because the IEN0 appears in the first SFR column, it is a bit-addressable SFR.

IEN1

Address: B1H
Reset State: 0000 0000B

Interrupt Enable Register 1. Contains the enable bits for the USB interrupts.

7							0
EX2	—	—	—	—	ESR	EF	ESOF

Bit Number	Bit Mnemonic	Function
7	EX2	External Interrupt 2 Enable (Keyboard Scan): Setting this bit enables the external interrupt used for the keyboard scan. **NOTE:** Setting this bit causes the 8*x*931 to trigger a hardware interrupt when a keyboard scan interrupt occurs, but only if the KSEN bit in the KBCON register is also set.
6:3	—	Reserved: Write zeros to these bits.
2	ESR	Enable Suspend/Resume: USB global suspend/resume interrupt enable bit.
1	EF	Enable Function: Transmit/Receive Done interrupt enable bit for non-isochronous USB function endpoints.
0	ESOF	Enable USB Hub/Start of Frame: Any start-of-frame interrupt enable for isochronous endpoints, or USB hub interrupt enable.

IPH0

Address: B7H
Reset State: x000 0000B

Interrupt Priority High Control Register 0. IPH0, together with IPL0, assigns each interrupt in IEN0 a priority level from 0 (lowest) to 3 (highest):

IPH0*x*	IPL0*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7 **0**

—	—	IPH0.5	IPH0.4	IPH0.3	IPH0.2	IPH0.1	IPH0.0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	IPH0.5	Timer 2 Overflow Interrupt Priority Bit High.
4	IPH0.4	Serial I/O Port Interrupt Priority Bit High.
3	IPH0.3	Timer 1 Overflow Interrupt Priority Bit High.
2	IPH0.2	External Interrupt 1 Priority Bit High.
1	IPH0.1	Timer 0 Overflow Interrupt Priority Bit High.
0	IPH0.0	External Interrupt 0 Priority Bit High.

IPL0

Address: B8H
Reset State: x000 0000B

Interrupt Priority Low Control Register 0. IPL0, together with IPH0, assigns each interrupt in IEN0 a priority level from 0 (lowest) to 3 (highest):

IPH0*x*	IPL0*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
—	—	IPL0.5	IPL0.4	IPL0.3	IPL0.2	IPL0.1	IPL0.0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Write zeros to these bits.
5	IPL0.5	Timer 2 Overflow Interrupt Priority Bit Low.
4	IPL0.4	Serial I/O Port Interrupt Priority Bit Low.
3	IPL0.3	Timer 1 Overflow Interrupt Priority Bit Low.
2	IPL0.2	External Interrupt 1 Priority Bit Low.
1	IPL0.1	Timer 0 Overflow Interrupt Priority Bit Low.
0	IPL0.0	External Interrupt 0 Priority Bit Low.

IPH1

Address: B3H
Reset State: 0000 0000B

Interrupt Priority High Control Register 1. IPH1, together with IPL1, assigns each interrupt in IEN1 a priority level from 0 (lowest) to 3 (highest):

IPH1.*x*	IPL1.*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
IPH1.7	—	—	—	—	IPH1.2	IPH1.1	IPH1.0

Bit Number	Bit Mnemonic	Function
7	IPH1.7	Keyboard Scan Interrupt Priority Bit High.
6:3	—	Reserved: Write zeros to these bits.
2	IPH1.2	Global Suspend/Resume Interrupt Priority Bit High.
1	IPH1.1	USB Function Interrupt Priority Bit High.
0	IPH1.0	USB Hub/SOF Interrupt Priority Bit High.

IPL1

Address: B2H
Reset State: 0000 0000B

Interrupt Priority Low Control Register 1. IPL1, together with IPH1, assigns each interrupt in IEN1 a priority level from 0 (lowest) to 3 (highest):

IPH1.*x*	IPL1.*x*	Priority Level
0	0	0 (lowest priority)
0	1	1
1	0	2
1	1	3 (highest priority)

7							0
IPL1.7	—	—	—	—	IPL1.2	IPL1.1	IPL1.0

Bit Number	Bit Mnemonic	Function
7	IPL1.7	Keyboard Scan Interrupt Priority Bit Low.
6:3	—	Reserved: Write zeros to these bits.
2	IPL1.2	Global Suspend/Resume Interrupt Priority Bit Low.
1	IPL1.1	USB Function Interrupt Priority Bit Low.
0	IPL1.0	USB Hub/SOF Interrupt Priority Bit Low.

KBCON

Address: F8H
Reset State: 0xx0 0000B

Keyboard Control Register. This register controls the keyboard scan input and output activity, enables and configures the keyboard scan interrupt, and drives the keyboard LEDs.

7							0
IE2	—	KSEN	IT2	LED3	LED2	LED1	LED0

Bit Number	Bit Mnemonic	Function
7	IE2	Interrupt 2 Flag: Set when external interrupt 2 is detected if the KSEN bit is set. Firmware must clear this bit when the interrupt is serviced.
6	—	Reserved: Write a zero to this bit.
5	KSEN	Keyboard Scan Enable: Setting this bit enables the pullup resistors on the KSI input lines, enables the keyboard scan interrupt (INT2#), and enables the LED drivers. **NOTE:** The EX2 bit in the IEN0 SFR must also be set to enable the KSI external interrupt.
4	IT2	Interrupt 2 Type Control Bit: If set, a negative edge detect on any of the KSI pins causes IE2 to be set. When clear, IE2 acts as a level 0 triggered interrupt.
3:0	LED3:0	LED Driver Control: Clearing one of these bits turns on the associated LED. Setting a bit turns off the associated LED. **NOTE:** The KSEN (Keyboard Scan Enable) bit must be set in order to activate the LED drivers. After reset, the LED driver control bits are cleared. This means that when KSEN is set, the LEDs will turn on. Firmware must set the LED driver control bits to turn off the LEDs.

P0

Address: 80H
Reset State: 1111 1111B

Port 0. P0 is the SFR that contains data to be driven out from the port 0 pins. Read-modify-write instructions that read port 0 read this register. The other instructions that read port 0 read the port 0 pins. When port 0 is used for an external bus cycle, the CPU always writes FFH to P0, and the former contents of P0 are lost.

7							0
P0 Contents							

Bit Number	Bit Mnemonic	Function
7:0	P0.7:0	Port 0 Register: Write data to be driven onto the port 0 pins to these bits.

P1

Address: 90H
Reset State: 1111 1111B

Port 1. P1 is the SFR that contains data to be driven out from the port 1 pins. Read-modify-write instructions that read port 1 read this register. Other instructions that read port 1 read the port 1 pins.

7							0
P1 Contents							

Bit Number	Bit Mnemonic	Function
7:0	P1.7:0	Port 1 Register: Write data to be driven onto the port 1 pins to these bits.

P2

Address: A0H
Reset State: 1111 1111B

Port 2. P2 is the SFR that contains data to be driven out from the port 2 pins. Read-modify-write instructions that read port 2 read this register. Other instructions that read port 2 read the port 2 pins.

7							0
P2 Contents							

Bit Number	Bit Mnemonic	Function
7:0	P2.7:0	Port 2 Register: Write data to be driven onto the port 2 pins to these bits.

P3

Address: B0H
Reset State: 1111 1111B

Port 3. P3 is the SFR that contains data to be driven out from the port 3 pins. Read-modify-write instructions that read port 3 read this register. Other instructions that read port 3 read the port 3 pins.

7							0
P3 Contents							

Bit Number	Bit Mnemonic	Function
7:0	P3.7:0	Port 3 Register: Write data to be driven onto the port 3 pins to these bits.

PCON Address: 87H
Reset State: 001d 0000B

Power Control Register. Contains the power off flag (POF) and bits for enabling the idle and powerdown modes and two general-purpose flags.

7 **0**

SMOD1	SMOD0	LC	POF	GF1	GF0	PD	IDL

Bit Number	Bit Mnemonic	Function
7	SMOD1	Double Baud Rate Bit: When set, doubles the baud rate when timer 1 is used and mode 1, 2, or 3 is selected in the SCON register. See "Baud Rates" on page 11-10.
6	SMOD0	SCON.7 Select: When set, read/write accesses to SCON.7 are to the FE bit. When clear, read/write accesses to SCON.7 are to the SM0 bit.See the SCON register (Figure 11-2 on page 11-4).
5	LC	Low-clock Mode Enable: Setting this bit forces the internal clock (F_{CLK}) distributed to the CPU and peripherals (but not the USB module) to 3 MHz. This bit is automatically set after a reset. Clearing this bit through firmware returns F_{CLK} to the normal clock frequency.
4	POF	Power Off Flag: Set by hardware on the rising edge of Vcc. set or cleared by software. This flag allows detection of a reset caused by a power failure. Vcc must remain above 3 volts to retain this bit.
3	GF1	General Purpose Flag: Set or cleared by firmware. One use is to indicate whether an interrupt occurred during normal operation or during idle mode.
2	GF0	General Purpose Flag: Set or cleared by firmware. One use is to indicate whether an interrupt occurred during normal operation or during idle mode.
1	PD	Powerdown Mode Bit: When set, activates powerdown mode. This bit should only be set if the GSUS bit is also set. Cleared by hardware when an interrupt or reset occurs.
0	IDL	Idle Mode Bit: When set, activates idle mode. Cleared by hardware when an interrupt or reset occurs. If IDL and PD are both set, PD takes precedence.

PCON1 Address: DFH
Reset State: xxxx x000B

USB Power Control Register. Facilitates the control and status relating to global suspend and resume, USB reset separation, and remote wake-up of the 8*x*931.

7							0
—	—	—	URDIS	URST	RWU	GRSM	GSUS

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved: Write zeros to these bits.
4	URDIS	USB Reset Disable: When cleared by firmware, a chip reset occurs upon receiving of a USB reset signal. This resets the MCS® 51 microcontroller core, USB blocks and all peripherals. When set by firmware, the core and peripherals will not reset when a USB reset signal is detected. Upon detecting a USB reset signal, the 8*x*931 resets all the USB blocks (FIFOs, FIU, SIE, and transceiver), sets the URST bit and generates a USB reset interrupt (refer to the description of URST).
3	URST	USB Reset Flag: This flag will be set by hardware when a USB reset occurs, regardless of whether the ESR bit in the IEN1 register is enabled or disabled. The URST also serves as the interrupt bit, ORed with GRSM and GSUS bits to generate an interrupt. Should be cleared by firmware when serving the USB reset interrupt.
2	RWU	Remote Wake-up Bit: 1 = wake-up. This bit is used by the USB function to initiate a remote wake-up. Set by firmware to drive resume signaling on the USB lines to the host or upstream hub. Cleared by hardware when resume signaling is done. **NOTE:** Do not set this bit unless the USB function is suspended (GSUS = 1 and GRSM = 0). See Figure 14-2 on page 14-4.

† Firmware should prioritize GRSM over GSUS if both bits are set simultaneously.

PCON1 (Continued) Address: DFH
Reset State: xxxx x000B

USB Power Control Register. Facilitates the control and status relating to global suspend and resume, USB reset separation, and remote wake-up of the 8*x*931.

7							0
—	—	—	URDIS	URST	RWU	GRSM	GSUS

1	GRSM	Global Resume Bit: 1 = resume. Set by hardware when a global resume is detected on the USB lines. This bit is ORed with GSUS to generate the interrupt† Cleared by firmware when servicing the global suspend/resume interrupt. (This bit can also be set/cleared by firmware for testability.) This bit is not set if remote wakeup is used (see RWU). See Figure 14-2 on page 14-4.
0	GSUS	Global Suspend Bit: 1 = suspend. This bit is set by hardware when global suspend is detected on the USB lines. This bit is ORed with the GRSM bit to generate the interrupt.† During the global suspend ISR, firmware should set the PD bit to enter the suspend mode. Cleared by hardware when a resume occurs. See Figure 14-2 on page 14-4.

† Firmware should prioritize GRSM over GSUS if both bits are set simultaneously.

PSW

Address: D0H
Reset State: 0000 0000B

7							0
CY	AC	F0	RS1	RS0	OV	UD	P

Bit Number	Bit Mnemonic	Function
7	CY	Carry Flag: The carry flag is set by an addition instruction (ADD, ADDC) if there is a carry out of the MSB. It is set by a subtraction (SUB, SUBB) or compare (CMP) if a borrow is needed for the MSB. The carry flag is also affected by logical bit, bit move, multiply, decimal adjust, and some rotate and shift instructions.
6	AC	Auxiliary Carry Flag: The auxiliary carry flag is affected only by instructions that address 8-bit operands. The AC flag is set if an arithmetic instruction with an 8-bit operand produces a carry out of bit 3 (from addition) or a borrow into bit 3 (from subtraction). Otherwise, it is cleared. This flag is useful for BCD arithmetic.
5	F0	Flag 0: This general-purpose flag is available to the user.
4:3	RS1:0	Register Bank Select Bits 1 and 0: These bits select the memory locations that comprise the active bank of the register file (registers R0–R7). RS1 RS0 Bank Address 0 0 0 00H–07H 0 1 1 08H–0FH 1 0 2 10H–17H 1 1 3 18H–1FH
2	OV	Overflow Flag: This bit is set if an addition or subtraction of signed variables results in an overflow error (i.e., if the magnitude of the sum or difference is too great for the seven LSBs in 2's-complement representation). The overflow flag is also set if a multiplication product overflows one byte or if a division by zero is attempted.
1	UD	User-definable Flag: This general-purpose flag is available to the user.
0	P	Parity Bit: This bit indicates the parity of the accumulator. It is set if an odd number of bits in the accumulator are set. Otherwise, it is cleared. Not all instructions update the parity bit. The parity bit is set or cleared by instructions that change the contents of the accumulator (ACC).

RCAP2H, RCAP2L

Address: RCAP2H CBH
RCAP2L CAH
Reset State: 0000 0000B

Timer 2 Reload/Capture Registers. This register pair stores 16-bit values to be loaded into or captured from the timer register (TH2/TL2) in timer 2.

7							0
High/Low Byte of Timer 2 Reload/Capture Value							

Bit Number	Bit Mnemonic	Function
7:0	RCAP2H.7:0	High byte of the timer 2 reload/recapture register
	RCAP2L.7:0	Low byte of the timer 2 reload/recapture register

RXCNTL[†]
(Endpoint-indexed)

Address: E6H
Reset State: xxxx xxxxB

Receive FIFO Byte-count Low Register. Ring buffer used to store the byte count for the data packets received in the receive FIFO specified by EPINDEX.

7							0
—	—	—	BC4	BC3	BC2	BC1	BC0

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved. Write zeros to these bits.
4:0	BC4:0	Receive Byte Count. Five-bit, ring buffer. Stores receive byte count.

[†] Not implemented for hub endpoint 1.

RXCON
(Endpoint-indexed)

Address: E4H
Reset State: 0xx0 0100B

Receive FIFO Control Register. Controls the receive FIFO specified by EPINDEX.

7							0
RXCLR	—	—	RXFFRC	RXISO	ARM	ADVWM	REVWP

Bit Number	Bit Mnemonic	Function
7	RXCLR	Clear the Receive FIFO: Set this bit to flush the entire receive FIFO. All flags in RXFLG revert to their reset states (RXEMP is set; all other flags clear). The ARM, RXISO, and RXWS bits in this register and the RXSEQ bit in the RXSTAT register are not affected by this operation. Hardware clears this bit when the flush operation is completed.
6:5	—	Reserved: Values read from this bit are indeterminate. Write zero to this bit.
4	RXFFRC	FIFO Read Complete: Set this bit to release the receive FIFO when a data set read is complete. Setting this bit “clears” the RXFIF “bit” (in the RXFLG register) corresponding to the data set that was just read. Hardware clears this bit after the RXFIF bit is cleared. All data from this data set must have been read. **NOTE:** FIFO Read Complete only works if STOVW and EDOVW are cleared.
3	RXISO	Isochronous Data Type: Set this bit to indicate that the receive FIFO is programmed to receive isochronous data and to set up the USB Interface to handle an isochronous data transfer. This bit is not reset when the RXCLR bit is set; it must be cleared by firmware.
2	ARM	Auto Receive Management: When set, the write pointer and write marker are adjusted automatically based on the following conditions: **RXISO / RX Status / Write Pointer / Write Marker** X / ACK / Unchanged / Advanced 0 / NAK / Reversed / Unchanged 1 / NAK / Unchanged / Advanced When this bit is set, setting REVWP or ADVWM has no effect. Hardware neither clears nor sets this bit. This is a sticky bit that is not reset when RXCLR is set. **NOTE:** This bit should always be set, except for testing.

† ARM mode is recommended. ADVWM and REVWP, which control the write marker and write pointer when ARM = 0, are used for test purposes.

RXCON (Continued) Address: E4H
(Endpoint-indexed) Reset State: 0xx0 0100B

Receive FIFO Control Register. Controls the receive FIFO specified by EPINDEX.

7							0
RXCLR	—	—	RXFFRC	RXISO	ARM	ADVWM	REVWP

Bit Number	Bit Mnemonic	Function
1	ADVWM	Advance Write Marker: † (For non-ARM mode only) Set this bit to advance the write marker to the origin of the next data set. Advancing the write marker is used for back-to-back receptions. Hardware clears this bit after the write marker is advanced. Setting this bit is effective only when the REVWP, ARM, and RXCLR bits are clear.
0	REVWP	Reverse Write Pointer: † (For non-ARM mode only) Set this bit to return the write pointer to the origin of the last data set received, as identified by the write marker. The FIU can then receive the last data packet again and write to the receive FIFO starting from the same origin when the host re-sends the same data packet. Hardware clears this bit after the write pointer is reversed. Setting this bit is effective only when the ADVWM, ARM, and RXCLR bits are all clear. REVWP is used when a data packet is bad. When the function interface receives the data packet again, the write starts at the origin of the previous (bad) data set.

† ARM mode is recommended. ADVWM and REVWP, which control the write marker and write pointer when ARM = 0, are used for test purposes.

RXDAT Address: E3H
(Endpoint-indexed) Reset: xxxx xxxxB

Receive FIFO Data Register. Receive FIFO data specified by EPINDEX is stored and read from this register.

7	0
RXDAT.7:0	

Bit Number	Bit Mnemonic	Function
7:0	RXDAT.7:0	To write data to the receive FIFO, the FIU writes to this register. To read data from the receive FIFO, the 8*x*931 reads from this register. The write pointer and read pointer are incremented automatically after a write and read, respectively.

RXFLG
(Endpoint-indexed)

Address: E5H
Reset State: 00xx 1000B

Receive FIFO Flag Register. These flags indicate the status of data packets in the receive FIFO specified by EPINDEX.

7							0
RXFIF1	RXFIF0	—	—	RXEMP	RXFULL	RXURF	RXOVF

Bit Number	Bit Mnemonic	Function
7:6	RXFIF1:0	Receive FIFO Index Flags: (read-only) These read-only flags indicate which data packets are present in the receive FIFO (see Table 6-6 on page 6-27). The RXFIF bits are updated after each write to RXCNT to reflect the addition of a data packet. Likewise, the RXFIF bits are cleared in sequence after each setting of the RXFFRC bit. The next-state table for RXFIF bits is shown below for operation in dual packet mode.

RXFIF1:0	Operation	Flag	Next RXFIF1:0	Next Flag
00	Adv WM	X	01	Unchanged
01	Adv WM	X	01	Unchanged
10	Adv WM	X	11	Unchanged
00	Set RXFFRC	X	00	Unchanged
01	Set RXFFRC	X	00	Unchanged
11	Set RXFFRC	X	10/01	Unchanged
10	Set RXFFRC	X	00	Unchanged
XX	Rev WP	X	Unchanged	Unchanged

When the receive FIFO is programmed to operate in single packet mode (RXSPM set in EPCON), valid RXFIF states are 00 and 01 only.

In ISO mode, RXOVF, RXURF, and RXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. RXFIF is “incremented” by the USB and “decremented” by firmware.Therefore, setting RXFFRC “decrements” RXFIF immediately. However, a successful USB transaction within a frame “increments” RXFIF only at SOF. For traceability, you must check the RXFIF flags before and after reads from the receive FIFO and the setting of RXFFRC in RXCON.

NOTE: To simplify firmware development, it is recommended that you utilize control endpoints in single-packet mode only.

5:4	—	Reserved: Values read from these bits are indeterminate. Write zeros to these bits.
3	RXEMP	Receive FIFO Empty Flag (read-only): Hardware sets this flag when the write pointer is at the same location as the read pointer **and** the write pointer equals the write marker and neither pointer has rolled over. Hardware clears the bit when the empty condition no longer exists. This is not a sticky bit and always tracks the current status of the receive FIFO, regardless of ISO or non-ISO mode.

† When set, all transmissions are NAKed.

RXFLG (Continued) Address: E5H
(Endpoint-indexed) Reset State: 00xx 1000B

Receive FIFO Flag Register. These flags indicate the status of data packets in the receive FIFO specified by EPINDEX.

7							0
RXFIF1	RXFIF0	—	—	RXEMP	RXFULL	RXURF	RXOVF

Bit Number	Bit Mnemonic	Function
2	RXFULL	Receive FIFO Full Flag (read-only): Hardware sets this flag when the write pointer has rolled over and equals the read pointer. Hardware clears the bit when the full condition no longer exists. This is not a sticky bit and always tracks the current status of the receive FIFO, regardless of ISO or non-ISO mode.
1	RXURF	Receive FIFO Underrun Flag†: Hardware sets this bit when an additional byte is read from an empty receive FIFO or RXCNT. Hardware does **not** clear this bit, so you must clear it in firmware. When the receive FIFO underruns, the read pointer will not advance — it remains locked in the empty position. In ISO mode, RXOVF, RXURF, and RXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since underrun can only be caused by firmware, RXURF is updated immediately. You must check the RXURF flag after reads from the receive FIFO before setting the RXFFRC bit in RXCON. **NOTE:** When this bit is set, the FIFO is in an unknown state. It is recommended that you reset the FIFO in the error management routine using the RXCLR bit in the RXCON register.
0	RXOVF	Receive FIFO Overrun Flag†: This bit is set when the FIU writes an additional byte to a full receive FIFO or writes a byte count to RXCNT with FIF1:0 = 11. This is a sticky bit that ***must*** be cleared through firmware, although it can be cleared by hardware if a SETUP packet is received after an RXOVF error had already occurred.† When this bit is set, the FIFO is in an unknown state, thus it is recommended that you reset the FIFO in the error management routine using the RXCLR bit in the RXCON register. When the receive FIFO overruns, the write pointer will not advance — it remains locked in the full position. In ISO mode, RXOVF, RXURF, and RXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since overrun can only be caused by the USB, RXOVF is updated only at the next SOF regardless of where the overrun occurred during the current frame.†

† When set, all transmissions are NAKed.

RXSTAT
(Endpoint-indexed)

Address: E2H
Reset State: 0000 0000B

Endpoint Receive Status Register. Contains the current endpoint status of the receive FIFO specified by EPINDEX. (Endpoint-indexed SFR)

7							0
RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK

Bit Number	Bit Mnemonic	Function
7	RXSEQ	Receiver Endpoint Sequence Bit (read, conditional write): † This bit will be toggled on completion of an ACK handshake in response to an OUT token. This bit will be set (or cleared) by hardware after reception of a SETUP token. This bit can be written by firmware if the RXSOVW bit is set when written along with the new RXSEQ value. **NOTE:** Always verify this bit after writing to ensure that there is no conflict with hardware, which could occur if a new SETUP token is received.
6	RXSETUP	Received Setup Token: This bit is set by hardware when a valid SETUP token has been received. When set, this bit causes received IN or OUT tokens to be NAKed until the bit is cleared to allow proper data management for the transmit and receive FIFOs from the previous transaction. IN or OUT tokens are NAKed even if the endpoint is stalled (RXSTL or TXSTL) to allow a control transaction to clear a stalled endpoint. Clear this bit upon detection of a SETUP token after the firmware is ready to complete the status stage of a control transaction.
5	STOVW	Start Overwrite Flag (read-only): Set by hardware upon receipt of a SETUP token for any control endpoint to indicate that the receive FIFO is being overwritten with new SETUP data. When set, the FIFO state (FIF and read pointer) resets and is locked for this endpoint until EDOVW is set. This prevents a prior, ongoing firmware read from corrupting the read pointer as the receive FIFO is being cleared and new data is being written into it. This bit is cleared by hardware at the end of handshake phase transmission of the setup stage. This bit is used only for control endpoints.

† Under normal operation, this bit should not be modified by the user.

†† For additional information on the operation of these bits see Appendix D, “Data Flow Model”.

††† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

RXSTAT (Continued) Address: E2H
(Endpoint-indexed) Reset State: 0000 0000B

Endpoint Receive Status Register. Contains the current endpoint status of the receive FIFO specified by EPINDEX. (Endpoint-indexed SFR)

7							0
RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK

Bit Number	Bit Mnemonic	Function
4	EDOVW	End Overwrite Flag: This flag is set by hardware during the handshake phase of a SETUP stage. It is set after every SETUP packet is received and **must** be cleared prior to reading the contents of the FIFO. When set, the FIFO state (FIF and read pointer) remains locked for this endpoint until this bit is cleared. This prevents a prior, ongoing firmware read from corrupting the read pointer after the new data has been written into the receive FIFO. This bit is only used for control endpoints. **NOTE:** Make sure the EDOVW bit is cleared prior to reading the contents of the FIFO.
3	RXSOVW	Receive Data Sequence Overwrite Bit: † Write a '1' to this bit to allow the value of the RXSEQ bit to be overwritten. Writing a '0' to this bit has no effect on RXSEQ. This bit always returns '0' when read. †††
2	RXVOID	Receive Void Condition (read-only):†† This bit is set when no valid data is received in response to a SETUP or OUT token due to one of the following conditions: 1. The receive FIFO is still locked 2. The EPCON register's RXSTL bit is set This bit is set and cleared by hardware. For non-isochronous transactions, this bit is updated by hardware at the end of the transaction in response to a valid OUT token. For isochronous transactions, it is not updated until the next SOF.

† Under normal operation, this bit should not be modified by the user.

†† For additional information on the operation of these bits see Appendix D, "Data Flow Model".

††† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

RXSTAT (Continued) Address: E2H
(Endpoint-indexed) Reset State: 0000 0000B

Endpoint Receive Status Register. Contains the current endpoint status of the receive FIFO specified by EPINDEX. (Endpoint-indexed SFR)

7							0
RXSEQ	RXSETUP	STOVW	EDOVW	RXSOVW	RXVOID	RXERR	RXACK

Bit Number	Bit Mnemonic	Function
1	RXERR	Receive Error (read-only):†† Set when an error condition has occurred with the reception. Complete or partial data has been written into the receive FIFO. No handshake is returned. The error can be one of the following conditions: 1. Data failed CRC check. 2. Bit stuffing error. 3. A receive FIFO goes into overrun or underrun condition while receiving. This bit is updated by hardware at the end of a valid SETUP or OUT token transaction (non-isochronous) or at the next SOF on each valid OUT token transaction (isochronous). The corresponding FRXD***x*** bit of FIFLG or FIFLG1 (8***x***930A***x*** with 6EPP) is set when active. This bit is updated with the RXACK bit at the end of data reception and is mutually exclusive with RXACK.
0	RXACK	Receive Acknowledged (read-only):†† This bit is set when data is received completely into a receive FIFO and an ACK handshake is sent. This read-only bit is updated by hardware at the end of a valid SETUP or OUT token transaction (non-isochronous) or at the next SOF on each valid OUT token transaction (isochronous). The corresponding FRXD***x*** bit of FIFLG or FIFLG1 (8***x***930A***x*** with 6EPP) is set when active. This bit is updated with the RXERR bit at the end of data reception and is mutually exclusive with RXERR.

† Under normal operation, this bit should not be modified by the user.

†† For additional information on the operation of these bits see Appendix D, “Data Flow Model”.

††† The SIE will handle all sequence bit tracking. This bit should be used only when initializing a new configuration or interface.

SADDR

Address: A9H
Reset State: 0000 0000B

Slave Individual Address Register. SADDR contains the device's individual address for multiprocessor communication.

7							0
Slave Individual Address							

Bit Number	Bit Mnemonic	Function
7:0	SADDR.7:0	

SADEN

Address: B9H
Reset State: 0000 0000B

Mask Byte Register. This register masks bits in the SADDR register to form the device's given address for multiprocessor communication.

7							0
Mask for SADDR							

Bit Number	Bit Mnemonic	Function
7:0	SADEN.7:0	

SBUF

Address: 99H
Reset State: xxxx xxxxB

Serial Data Buffer. Writing to SBUF loads the transmit buffer of the serial I/O port. Reading SBUF reads the receive buffer of the serial I/O port.

7							0
Data Sent/Received by Serial I/O Port							

Bit Number	Bit Mnemonic	Function
7:0	SBUF.7:0	

SCON Address: 98H
Reset State: 0000 0000B

Serial Port Control Register. SCON contains serial I/O control and status bits, including the mode select bits and the interrupt flag bits.

7							0
FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit Number	Bit Mnemonic	Function
7	FE	Framing Error Bit: To select this function, set the SMOD0 bit in the PCON register. Set by hardware to indicate an invalid stop bit. Cleared by firmware, not by valid frames.
	SM0	Serial Port Mode Bit 0: To select this function, clear the SMOD0 bit in the PCON register. Firmware writes to bits SM0 and SM1 to select the serial port operating mode. Refer to the SM1 bit for the mode selections.
6	SM1	Serial Port Mode Bit 1: Firmware writes to bits SM1 and SM0 (above) to select the serial port operating mode. **SM0 SM1 Mode Description Baud Rate**[†] 0 0 0 Shift register $F_{OSC}/12$ 0 1 1 8-bit UART Variable 1 0 2 9-bit UART $F_{OSC}/64$[††] or $F_{OSC}/32$[††] 1 1 3 9-bit UART Variable [†]F_{OSC} = Oscillator frequency. [††]Select by programming the SMOD1 bit in the PCON register (see section “Baud Rates” on page 11-10).
5	SM2	Serial Port Mode Bit 2: Firmware writes to bit SM2 to enable and disable the multiprocessor communication and automatic address recognition features. This allows the serial port to differentiate between data and command frames and to recognize slave and broadcast addresses.
4	REN	Receiver Enable Bit: To enable reception, set this bit. To disable reception, clear this bit.
3	TB8	Transmit Bit 8: In modes 2 and 3, firmware writes the ninth data bit to be transmitted to TB8. Not used in modes 0 and 1.

SCON (Continued)

Address: 98H
Reset State: 0000 0000B

Serial Port Control Register. SCON contains serial I/O control and status bits, including the mode select bits and the interrupt flag bits.

7							0
FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit Number	Bit Mnemonic	Function
2	RB8	Receiver Bit 8: Mode 0: Not used. Mode 1 (SM2 clear): Set or cleared by hardware to reflect the stop bit received. Modes 2 and 3 (SM2 set): Set or cleared by hardware to reflect the ninth data bit received.
1	TI	Transmit Interrupt Flag Bit: Set by the transmitter after the last data bit is transmitted. Cleared by firmware.
0	RI	Receive Interrupt Flag Bit: Set by the receiver after the last data bit of a frame has been received. Cleared by firmware.

SOFH

Address: D3H
Reset State: 0000 1000B

Start-of-frame High Register. Contains isochronous data transfer enable and interrupt bits and the upper three bits of the 11-bit time stamp received from the host.

7							0
SOFACK	ASOF	SOFIE	FTLOCK	SOFODIS	TS10	TS9	TS8

Bit Number	Bit Mnemonic	Function
7	SOFACK	SOF Token Received without Error (read-only): When set, this bit indicates that the 11-bit time stamp stored in SOFL and SOFH is valid. This bit is updated every time an SOF token is received from the USB bus, and it is cleared when an artificial SOF is generated by the frame timer. This bit is set and cleared by hardware.
6	ASOF	Any Start of Frame: This bit is set by hardware to indicate that a new frame has started. The interrupt can result either from reception of an actual SOF packet or from an artificially-generated SOF from the frame timer. This interrupt is asserted in hardware even if the frame timer is not locked to the USB bus frame timing. When set, this bit is an indication that either an actual SOF packet was received or an artificial SOF was generated by the frame timer. This bit must be cleared by firmware or inverted and driven to the SOF# pin. The effect of setting this bit by firmware is the same as hardware: the external pin will be driven with an inverted ASOF value for eight T_{CLK}s. This bit also serves as the SOF interrupt flag. This interrupt is only asserted in hardware if the SOF interrupt is enabled (SOFIE set) and the interrupt channel is enabled.
5	SOFIE	SOF Interrupt Enable: When this bit is set, setting the ASOF bit causes an interrupt request to be generated if the interrupt channel is enabled. Hardware reads but does not write this bit.
4	FTLOCK	Frame Timer Locked (read-only): When set, this bit indicates that the frame timer is presently locked to the USB bus' frame time. When cleared, this bit indicates that the frame timer is attempting to synchronize to the frame time.
3	SOFODIS	SOF# Pin Output Disable: When set, the SOF# pin will be disabled and will respond like a port pin. The SOF# pin will be driven to '1' when SOFODIS is set. When this bit is clear, setting the ASOF bit causes the SOF# pin to be toggled with a low pulse for eight T_{CLK}s.
2:0	TS10:8	Time stamp received from host: TS10:8 are the upper three bits of the 11-bit frame number issued with an SOF token. This time stamp is valid only if the SOFACK bit is set. If an artificial SOF is generated, the time stamp remains at its previous value and it is up to firmware to update it. These bits are set and cleared by hardware.

SOFL Address: D2H
Reset State:0000 0000B

Start-of-Frame Low Register. Contains the lower eight bits of the 11-bit time stamp received from the host.

7							0
TS7:0							

Bit Number	Bit Mnemonic	Function
7:0	TS7:0	Time stamp received from host: This time stamp is valid only if the SOFACK bit in the SOFH register is set. TS7:0 are the lower eight bits of the 11-bit frame number issued with a SOF token. If an artificial SOF is generated, the time stamp remains at its previous value and it is up to firmware to update it. These bits are set and cleared by hardware.

SP Address: 81H
Reset State: 0000 0111B

Stack Pointer. The Stack Pointer register is 8-bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the stack pointer is initialized to 07h after reset. This causes the stack to begin at location 08h.

7							**0**
SP Contents							

Bit Number	Bit Mnemonic	Function
7:0	SP.7:0	Stack Pointer: Bits 0–7 of the stack pointer.

T2CON Address: C8H
Reset State: 0000 0000B

Timer 2 Control Register. Contains the receive clock, transmit clock, and capture/reload bits used to configure timer 2. Also contains the run control bit, counter/timer select bit, overflow flag, external flag, and external enable for timer 2.

7 **0**

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2#	CP/RL2#

Bit Number	Bit Mnemonic	Function
7	TF2	Timer 2 Overflow Flag: Set by timer 2 overflow. Must be cleared by firmware. TF2 is not set if RCLK = 1 or TCLK = 1.
6	EXF2	Timer 2 External Flag: If EXEN2 = 1, capture or reload caused by a negative transition on T2EX sets EFX2. EXF2 does not cause an interrupt in up/down counter mode (DCEN = 1).
5	RCLK	Receive Clock Bit: Selects timer 2 overflow pulses (RCLK = 1) or timer 1 overflow pulses (RCLK = 0) as the baud rate generator for serial port modes 1 and 3.
4	TCLK	Transmit Clock Bit: Selects timer 2 overflow pulses (TCLK = 1) or timer 1 overflow pulses (TCLK = 0) as the baud rate generator for serial port modes 1 and 3.
3	EXEN2	Timer 2 External Enable Bit: Setting EXEN2 causes a capture or reload to occur as a result of a negative transition on T2EX unless timer 2 is being used as the baud rate generator for the serial port. Clearing EXEN2 causes timer 2 to ignore events at T2EX.
2	TR2	Timer 2 Run Control Bit: Setting this bit starts the timer.
1	C/T2#	Timer 2 Counter/Timer Select: C/T2# = 0 selects timer operation: timer 2 counts the divided-down system clock. C/T2# = 1 selects counter operation: timer 2 counts negative transitions on external pin T2.
0	CP/RL2#	Capture/Reload Bit: When set, captures occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto-reloads occur on timer 2 overflows or negative transitions at T2EX if EXEN2 = 1. The CP/RL2# bit is ignored and timer 2 forced to auto-reload on timer 2 overflow, if RCLK = 1 or TCLK = 1.

T2MOD Address: C9H
Reset State: xxxx xx00B

Timer 2 Mode Control Register. Contains the timer 2 down count enable and clock-out enable bits for timer 2 .

7 **0**

—	—	—	—	—	—	T2OE	DCEN

Bit Number	Bit Mnemonic	Function
7:2	—	Reserved: Values read from these bits are indeterminate. Write zeros to these bits.
1	T2OE	Timer 2 Output Enable Bit: In the timer 2 clock-out mode, connects the programmable clock output to external pin T2.
0	DCEN	Down Count Enable Bit: Configures timer 2 as an up/down counter.

TCON

Address: 88H
Reset State: 0000 0000B

Timer/Counter Control Register. Contains the overflow and external interrupt flags and the run control and interrupt transition select bits for timer 0 and timer 1.

7 **0**

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit Number	Bit Mnemonic	Function
7	TF1	Timer 1 Overflow Flag: Set by hardware when the timer 1 register overflows. Cleared by hardware when the processor vectors to the interrupt routine.
6	TR1	Timer 1 Run Control Bit: Set/cleared by firmware to turn timer 1 on/off.
5	TF0	Timer 0 Overflow Flag: Set by hardware when the timer 0 register overflows. Cleared by hardware when the processor vectors to the interrupt routine.
4	TR0	Timer 0 Run Control Bit: Set/cleared by firmware to turn timer 0 on/off.
3	IE1	Interrupt 1 Flag: Set by hardware when an external interrupt is detected on the INT1# pin. Edge- or level- triggered (see IT1). Cleared when interrupt is processed if edge-triggered.
2	IT1	Interrupt 1 Type Control Bit: Set this bit to select edge-triggered (high-to-low) for external interrupt 1. Clear this bit to select level-triggered (active low).
1	IE0	Interrupt 0 Flag: Set by hardware when an external interrupt is detected on the INT0# pin. Edge- or level- triggered (see IT0). Cleared when interrupt is processed if edge-triggered.
0	IT0	Interrupt 0 Type Control Bit: Set this bit to select edge-triggered (high-to-low) for external interrupt 0. Clear this bit to select level-triggered (active low).

TMOD

Address: 89H
Reset State: 0000 0000B

Timer/Counter Mode Control Register. Contains mode select, run control select, and counter/timer select bits for controlling timer 0 and timer 1.

7 **0**

GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

Bit Number	Bit Mnemonic	Function
7	GATE1	Timer 1 Gate: When GATE1 = 0, run control bit TR1 gates the input signal to the timer register. When GATE1 = 1 and TR1 = 1, external signal INT1 gates the timer input.
6	C/T1#	Timer 1 Counter/Timer Select: C/T1# = 0 selects timer operation: timer 1 counts the divided-down system clock. C/T1# = 1 selects counter operation: timer 1 counts negative transitions on external pin T1.
5, 4	M11, M01	Timer 1 Mode Select: **M11 M01** 0 0 Mode 0: 8-bit timer/counter (TH1) with 5-bit prescalar (TL1) 0 1 Mode 1: 16-bit timer/counter 1 0 Mode 2: 8-bit auto-reload timer/counter (TL1). Reloaded from TH1 at overflow. 1 1 Mode 3: Timer 1 halted. Retains count.
3	GATE0	Timer 0 Gate: When GATE0 = 0, run control bit TR0 gates the input signal to the timer register. When GATE0 = 1 and TR0 = 1, external signal INT0 gates the timer input.
2	C/T0#	Timer 0 Counter/Timer Select: C/T0# = 0 selects timer operation: timer 0 counts the divided-down system clock. C/T0# = 1 selects counter operation: timer 0 counts negative transitions on external pin T0.
1, 0	M10, M00	Timer 0 Mode Select: **M10 M00** 0 0 Mode 0: 8-bit timer/counter (T0) with 5-bit prescalar (TL0) 0 1 Mode 1: 16-bit timer/counter 1 0 Mode 2: 8-bit auto-reload timer/counter (TL0). Reloaded from TH0 at overflow. 1 1 Mode 3: TL0 is an 8-bit timer/counter. TH0 is an 8-bit timer using timer 1's TR1 and TF1 bits.

TH0, TL0

Address: TH08CH
TL08AH
Reset State: 0000 0000B

TH0, TL0 Timer Registers. These registers operate in cascade to form the 16-bit timer register in timer 0 or separately as 8-bit timer/counters.

7							0
High/Low Byte of Timer 0 Register							

Bit Number	Bit Mnemonic	Function
7:0	TH0.7:0	High byte of the timer 0 timer register.
	TL0.7:0	Low byte of the timer 0 timer register.

TH1, TL1

Address: TH18DH
TL1 8BH
Reset State: 0000 0000B

TH1, TL1 Timer Registers. These registers operate in cascade to form the 16-bit timer register in timer 1 or separately as 8-bit timer/counters.

7							0
High/Low Byte of Timer 1 Register							

Bit Number	Bit Mnemonic	Function
7:0	TH1.7:0	High byte of the timer 1 timer register.
	TL1.7:0	Low byte of the timer 1 timer register.

TH2, TL2

Address: TH2 CDH
TL2 CCH
Reset State: 0000 0000B

TH2, TL2 Timer Registers. These registers operate in cascade to form the 16-bit timer register in timer 2.

7							0
High/Low Byte of Timer 2 Register							

Bit Number	Bit Mnemonic	Function
7:0	TH2.7:0 TL2.7:0	High byte of the timer 2 timer register. Low byte of the timer 2 timer register.

TXCNTL† (Endpoint-indexed)

Address: F6H
Reset States: xxxx xxxxB

Transmit FIFO Byte-count Register. Ring buffer used to store the byte count for the data packets in the transmit FIFO specified by EPINDEX.

7							0
—	—	—	BC4	BC3	BC2	BC1	BC0

Bit Number	Bit Mnemonic	Function
7:5	—	Reserved. Write zeros to these bits.
4:0	BC4:0	Transmit Byte Count. (write, conditional read††) Five-bit, ring buffer. Stores transmit byte count for endpoints 0 and 2.

† Byte count registers are not implemented for hub endpoint 1.

†† Read these bits only if TXFIF1:0 = 0; otherwise underrun errors may occur.

TXCON
(Endpoint-indexed)

Address: F4H
Reset State: 0xxx 0100B

USB Transmit FIFO Control Register. Controls the transmit FIFO specified by EPINDEX.

7							0
TXCLR	—	—	—	TXISO	ATM	ADVRM	REVRP

Bit Number	Bit Mnemonic	Function
7	TXCLR	Transmit Clear: Setting this bit flushes the transmit FIFO, resets all the read/write pointers and markers, sets the EMPTY bit in TXFLG, and clears all other bits in TXFLG. After the flush, hardware clears this bit. Setting this bit does not affect the ATM, TXISO, and FFSZ bits, or the TXSEQ bit in the TXSTAT register.
6:4	—	Reserved: Values read from this bit are indeterminate. Write zeros to these bits.
3	TXISO	Transmit Isochronous Data: Firmware sets this bit to indicate that the transmit FIFO contains isochronous data. The FIU uses this bit to set up the handshake protocol at the end of a transmission. This bit is not reset when TXCLR is set and must be cleared by firmware.
2	ATM	Automatic Transmit Management: Setting this bit (the default value) causes the read pointer and read marker to be adjusted automatically as indicated: **TXISO TX Status Read Pointer Read Marker** X ACK Unchanged Advanced (1) 0 NAK Reversed (2) Unchanged 1 NAK Unchanged Advanced (1) (1) to origin of next data set (2) to origin of the data set last read This bit should always be set, except for test purposes. Setting this bit disables ADVRM and REVRP. This is a sticky bit that is not reset when TXCLR is set, but can be set and cleared by firmware. Hardware neither clears nor sets this bit.

† ATM mode is recommended. ADVRM and REVRP, which control the read marker and read pointer when ATM = 0, are used for test purposes.

TXCON (Continued) Address: F4H
(Endpoint-indexed) Reset State: 0xxx 0100B

USB Transmit FIFO Control Register. Controls the transmit FIFO specified by EPINDEX.

7							0
TXCLR	—	—	—	TXISO	ATM	ADVRM	REVRP

Bit Number	Bit Mnemonic	Function
1	ADVRM	Advance Read Marker Control (non-ATM mode only)†: Setting this bit prepares for the next packet transmission by advancing the read marker to the origin of the next data packet (the position of the read pointer). Hardware clears this bit after the read marker is advanced. This bit is effective only when the REVRP, ATM, and TXCLR bits are all clear.
0	REVRP	Reverse Read Pointer Control (non-ATM mode only)†: In the case of a bad transmission, the same data stack may need to be available for retransmit. Setting this bit reverses the read pointer to point to the origin of the last data set (the position of the read marker) so that the FIU can reread the last set for retransmission. Hardware clears this bit after the read pointer is reversed. This bit is effective only when the ADVRM, ATM, and TXCLR bits are all clear.

† ATM mode is recommended. ADVRM and REVRP, which control the read marker and read pointer when ATM = 0, are used for test purposes.

TXDAT Address: F3H
(Endpoint-indexed)† Reset State: xxxx xxxxB

USB Transmit FIFO Data Register. Data to be transmitted by the FIFO specified by EPINDEX is first written to this register.

7	0
Transmit Data Byte	

Bit Number	Bit Mnemonic	Function
7:0	TXDAT.7:0	Transmit Data Byte (write-only): To write data to the transmit FIFO, write to this register. The write pointer is incremented automatically after a write.

† For hub endpoint 1, TXDAT is used in a different manner. See Figure 7-7 on page 7-12.

TXDAT (For hub endpoint 1 only) EPINDEX=81H[†] Address: F3H
Reset State: xxxx xxxxB

7							0
—	—	TXDAT5	TXDAT4	TXDAT3	TXDAT2	TXDAT1	TXDAT0

Bit Number	Bit Mnemonic	Function
7:6	—	Reserved: Values read from this bit(s) are indeterminate.
5:0	TXDAT5:0	Hub Endpoint 1 Status Change (read-only[††]): Hardware communicates status changes to the host by setting the appropriate bit: TXDAT0 hub status change TXDAT1 port 1 status change TXDAT2 port 2 status change TXDAT3 port 3 status change TXDAT4 port 4 status change TXDAT5 port 5 status change A '1' indicates a status change and '0' indicates no status change. When endpoint 1 is addressed via an IN token, the entire byte is sent if at least one bit is a '1'. If all bits are zero, a NAK handshake is returned.

† TXDAT SFRs are also used for function (and hub endpoint 0) data transmission (EPINDEX=0xH or 80H). In that case, the bits are defined differently as shown in Figure 6-8 on page 6-16.

†† Bits 5:1 can be set indirectly by firmware by writing to a port's HPSC SFR. Setting any bit in port *x*'s HPSC results in the hardware setting bit *x* in TXDAT. TXDAT bits can be cleared indirectly in firmware by clearing all bits in that port's HPSC.

TXFLG Address: F5H
(Endpoint-indexed) Reset State: 00xx 1000B

Transmit FIFO Flag Register. These flags indicate the status of data packets in the transmit FIFO specified by EPINDEX.

7							0
TXFIF1	TXFIF0	—	—	TXEMP	TXFULL	TXURF	TXOVF

Bit Number	Bit Mnemonic	Function
7:6	TXFIF1:0	FIFO Index Flags (read-only): These flags indicate which data sets are present in the transmit FIFO. The FIF bits are set in sequence after each write to TXCNT to reflect the addition of a data set. Likewise, TXFIF1 and TXFIF0 are cleared in sequence after each advance of the read marker to indicate that the set is effectively discarded. The bit is cleared whether the read marker is advanced by firmware (setting ADVRM) or automatically by hardware (ATM = 1). The next-state table for the TXFIF bits is shown below:

TXFIF1:0	Operation	Flag	Next TXFIF1:0	Next Flag
00	Wr TXCNT	X	01	Unchanged
01	Wr TXCNT	X	11	Unchanged
10	Wr TXCNT	X	11	Unchanged
11	Wr TXCNT	X	11	TXOVF = 1
00	Adv RM	X	00	Unchanged
01	Adv RM	X	00	Unchanged
11	Adv RM	X	10/01	Unchanged
10	Adv RM	X	00	Unchanged
XX	Rev RP	X	Unchanged	Unchanged

In ISO mode, TXOVF, TXURF, and TXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. TXFIF is "incremented" by firmware and "decremented" by the USB.Therefore, writes to TXCNT "increment" TXFIF immediately. However, a successful USB transaction any time within a frame "decrements" TXFIF only at SOF. You must check the TXFIF flags before and after writes to the transmit FIFO and TXCNT for traceability. See the TXFLUSH bit in TXSTST.

NOTE: To simplify firmware development, configure control endpoints in single-packet mode.

5:4	—	Reserved: Values read from these bits are indeterminate. Write zeros to these bits.
3	TXEMP	Transmit FIFO Empty Flag (read-only): Hardware sets this bit when the write pointer has not rolled over and is at the same location as the read pointer. Hardware clears this bit when the pointers are at different locations. Regardless of ISO or non-ISO mode, this bit always tracks the current transmit FIFO status.

† When set, all transmissions are NAKed.

TXFLG (Continued)
(Endpoint-indexed)

Address: F5H
Reset State: 00xx 1000B

Transmit FIFO Flag Register. These flags indicate the status of data packets in the transmit FIFO specified by EPINDEX.

7							0
TXFIF1	TXFIF0	—	—	TXEMP	TXFULL	TXURF	TXOVF

Bit Number	Bit Mnemonic	Function
2	TXFULL	Transmit FIFO Full Flag (read-only): Hardware sets this bit when the write pointer has rolled over and equals the read marker. Hardware clears this bit when the full condition no longer exists. Regardless of ISO or non-ISO mode, this bit always tracks the current transmit FIFO status. Check this bit to avoid causing a TXOVF condition.
1	TXURF	Transmit FIFO Underrun Flag†: Hardware sets this flag when an additional byte is read from an empty transmit FIFO or TXCNT [This is caused when the value written to TXCNT is greater than the number of bytes written to TXDAT.]. This is a sticky bit that must be cleared through firmware. When this flag is set, the FIFO is in an unknown state, thus it is recommended that you reset the FIFO in your error management routine using the TXCLR bit in TXCON. When the transmit FIFO underruns, the read pointer will not advance — it remains locked in the empty position. If the TXCNT doesn't agree with the data, hardware sets TXURF. This indicates that the transmitted data was corrupted by a bit-stuffing or CRC error. In ISO mode, TXOVF, TXURF, and TXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since underrun can only be caused by USB, TXURF is updated at the next SOF regardless of where the underrun occurs in the frame.
0	TXOVF	Transmit FIFO Overrun Flag†: This bit is set when an additional byte is written to a full FIFO or full TXCNT with TXFIF1:0 = 11. This is a sticky bit that must be cleared through firmware. When this bit is set, the FIFO is in an unknown state, thus it is recommended that you reset the FIFO in your error management routine using the TXCLR bit in TXCON. When the receive FIFO overruns, the write pointer will not advance — it remains locked in the full position. Check this bit after loading the FIFO prior to writing the byte count register. In ISO mode, TXOVF, TXURF, and TXFIF are handled using the following rule: Firmware events cause status change immediately, while USB events cause status change only at SOF. Since overrun can only be caused by firmware, TXOVF is updated immediately. Check the TXOVF flag after writing to the transmit FIFO before writing to TXCNT.

† When set, all transmissions are NAKed.

TXSTAT Address: F2H
(Endpoint-indexed) Reset State: 0xx0 0000B

Endpoint Transmit Status Register. Contains the current endpoint status of the transmit FIFO specified by EPINDEX.

7							0
TXSEQ	—	—	TXFLUSH	TXSOVW	TXVOID	TXERR	TXACK

Bit Number	Bit Mnemonic	Function
7	TXSEQ	Transmitter's Current Sequence Bit (read, conditional write): † This bit will be transmitted in the next PID and toggled on a valid ACK handshake. This bit is toggled by hardware on a valid SETUP token. This bit can be written by firmware if the TXSOVW bit is set when written together with the new TXSEQ value.
6:5	—	Reserved: Write zeros to these bits.
4	TXFLUSH	Transmit FIFO Packet Flushed (read-only): When set, this bit indicates that hardware flushed a stale ISO data packet from the transmit FIFO due to a TXFIF1:0 = 11 at SOF. To guard against a missed IN token in ISO mode, if, with TXFIF1:0 = 11, no IN token is received for the current endpoint, hardware automatically flushes the oldest packet and decrements the TXFIF1:0 value.
3	TXSOVW	Transmit Data Sequence Overwrite Bit: † Write a '1' to this bit to allow the value of the TXSEQ bit to be overwritten. Writing a '0' to this bit has no effect on TXSEQ. This bit always returns '0' when read. ††
2	TXVOID	Transmit Void (read-only): ††† A void condition has occurred in response to a valid IN token. Transmit void is closely associated with the NAK/STALL handshake returned by the function after a valid IN token, due to the conditions that cause the transmit FIFO to be unenabled or not ready to transmit. Use this bit to check any NAK/STALL handshake returned by the function. This bit does not affect the FTXD***x***, TXERR or TXACK bits. This bit is updated by hardware at the end of a non-isochronous transaction in response to a valid IN token. For isochronous transactions, this bit is not updated until the next SOF.

† Under normal operation, this bit should not be modified by the user.

†† The SIE will handle all sequence bit tracking. This bit should only be used when initializing a new configuration or interface.

††† For additional information on the operation of these bits see Appendix D, "Data Flow Model".

TXSTAT (Continued) Address: F2H
(Endpoint-indexed) Reset State: 0xx0 0000B

Endpoint Transmit Status Register. Contains the current endpoint status of the transmit FIFO specified by EPINDEX.

7 **0**

TXSEQ	—	—	TXFLUSH	TXSOVW	TXVOID	TXERR	TXACK

Bit Number	Bit Mnemonic	Function
1	TXERR	Transmit Error (read-only):†† An error condition has occurred with the transmission. Complete or partial data has been transmitted. The error can be one of the following: 1. Data transmitted successfully but no handshake received 2. Transmit FIFO goes into underrun condition while transmitting The corresponding transmit done bit, FTXD***x*** in FIFLG or FIFLG1 (8***x***930A***x*** with 6EPP), is set when active. For non-isochronous transactions, this bit is updated by hardware along with the TXACK bit at the end of the data transmission (this bit is mutually exclusive with TXACK). For isochronous transactions, this bit is not updated until the next SOF.
0	TXACK	Transmit Acknowledge (read-only):†† Data transmission completed and acknowledged successfully. The corresponding transmit done bit, FTXD***x*** in FIFLG or FIFLG1 (8***x***930A***x*** with 6EPP), is set when active. For non-isochronous transactions, this bit is updated by hardware along with the TXERR bit at the end of data transmission (this bit is mutually exclusive with TXERR). For isochronous transactions, this bit is not updated until the next SOF.

† Under normal operation, this bit should not be modified by the user.

†† The SIE will handle all sequence bit tracking. This bit should only be used when initializing a new configuration or interface.

††† For additional information on the operation of these bits see Appendix D, “Data Flow Model”.

D

Data Flow Model

APPENDIX D
DATA FLOW MODEL

This appendix describes the data flow model for the 8*x*931 USB transactions. This data flow model, presented in truth table form, is intended to bridge the hardware and firmware layers of the 8*x*931. It describes the behavior of the 8*x*931 in response to a particular USB event, given a known state/configuration.

The types of data transfer supported by the 8*x*931 are:

- Non-isochronous transfer (interrupt, bulk)
- Isochronous transfer
- Control transfer

Table D-1. Non-isochronous Transmit Data Flow

TXFIF (1:0)	Event	New TXFIF (1:0)	TX ERR	TX ACK	TX Void	TX OVF (1)	TX URF (1)	TX Inter-rupt	USB Response	Comments
00	Received IN token, but no data or TXOE = 0.	00	no chg	no chg	1	no chg	no chg	None	NAK	No data was loaded, so NAK.
	Received IN token, RXSETUP =1.	00	no chg	no chg	1	no chg	no chg	None	NAK	Control endpoint only. Endpoint will NAK when RXSETUP = 1 even if TXSTL = 1.
	Data loaded into FIFO from CPU, CNT written.	01	no chg	no chg	no chg	no chg	no chg	None	N/A	Firmware should always check TXFIF bits before loading and TXOVF after loading.
	Data loaded into FIFO, FIFO error occurs.	00	no chg	no chg	no chg	1	no chg	None	NAKs future transactions	Only overrun FIFO error can occur here. Firmware should always check TXOVF before write CNT.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes TXEPEN and ATM are enabled.
2. Future transactions are NAKed even if the transmit endpoint is stalled when RXSETUP = 1.

Table D-1. Non-isochronous Transmit Data Flow (Continued)

TXFIF (1:0)	Event	New TXFIF (1:0)	TX ERR	TX ACK	TX Void	TX OVF (1)	TX URF (1)	TX Inter-rupt	USB Response	Comments
01/10	Received IN token, data transmitted, host ACKs.	00	0	1	0	no chg	no chg	Set transmit interrupt	Send data	ACK received, so no errors. Read marker advanced.
	Received IN token, data transmitted, no ACK (time-out).	01/10	1	0	0	no chg	no chg	Set transmit interrupt	Send data	SIE times-out. Read pointer reversed.
	Received IN token, but RXSETUP = 1 (or TXOE = 0).	01/10	no chg	no chg	1	no chg	no chg	None	NAK, NAKs future transactions except SETUP.	Received Setup token (or transmit disabled), so IN tokens are NAKed. (2)
	Received IN token, data transmitted, FIFO error occurs.	01/10	1	0	0	no chg	1	Set transmit interrupt	Send data with bit-stuff error. NAKs future transactions.	Only underrun FIFO error can occur here. Read pointer reversed.
	Received IN token with existing FIFO error and TXERR set.	01/10	1 (no chg)	0 (no chg)	1	no chg	1 (no chg)	None	NAK	Treated like a "void" condition.
	Received IN token without existing FIFO error but TXERR set, data retransmitted, host ACKs.	00	0	1	0	no chg	no chg	Set transmit interrupt	Send data	Data is retransmitted. TXACK is set and TXERR is cleared. The TXERR was set by previous transaction when host time-out.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes TXEPEN and ATM are enabled.
2. Future transactions are NAKed even if the transmit endpoint is stalled when RXSETUP = 1.

Table D-1. Non-isochronous Transmit Data Flow (Continued)

TXFIF (1:0)	Event	New TXFIF (1:0)	TX ERR	TX ACK	TX Void	TX OVF (1)	TX URF (1)	TX Inter-rupt	USB Response	Comments
	Data loaded into FIFO from CPU, CNT written.	11	no chg	no chg	no chg	no chg	no chg	None	N/A	Firmware should always check TXFIF bits before loading and TXOVF after loading.
	Data loaded into FIFO, FIFO error occurs. CNT not written yet.	01/10	no chg	no chg	no chg	1	no chg	None	NAKs future transactions	Only overrun FIFO error can occur here. Firmware should always check TXOVF before write CNT NOTE: no TXERR, but TXOVF set.
11	Received IN token, data transmitted, host ACKs.	10/01	0	1	0	no chg	no chg	Set transmit interrupt	Send data	ACK received, so no errors. Read marker advanced.
	Received IN token, data transmitted, no ACK (time-out).	11	1	0	0	no chg	no chg	Set transmit interrupt	Send data	SIE times-out. Read pointer reversed.
	Received IN token, but RXSETUP = 1 (or TXOE = 0).	11	0	0	1	no chg	no chg	None	NAK, NAKs future transactions	Received Setup token (or transmit disabled), so IN tokens are NAKed. (2)
	Received IN token, data transmitted, FIFO error occurs.	11	1	0	0	no chg	1	Set transmit interrupt	Send data with bit-stuff error, NAK future transactions	Only FIFO underrun error can occur here. Read pointer reversed.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes TXEPEN and ATM are enabled.
2. Future transactions are NAKed even if the transmit endpoint is stalled when RXSETUP = 1.

Table D-1. Non-isochronous Transmit Data Flow (Continued)

TXFIF (1:0)	Event	New TXFIF (1:0)	TX ERR	TX ACK	TX Void	TX OVF (1)	TX URF (1)	TX Inter-rupt	USB Response	Comments
	Received IN token with existing FIFO error and TXERR set.	11	1 (no chg)	0 (no chg)	1	no chg	1 (no chg)	None	NAK	Treated like a "void" condition.
	Received IN token without existing FIFO error but TXERR set, data retransmitted, host ACKs.	10/ 01	0	1	0	no chg	no chg	Set transmit interrupt	Send data	Data is retransmitted. TXACK is set and TXERR is cleared. The TXERR was set by previous transaction when host time-out.
	Data loaded into FIFO from CPU, CNT written.	11	no chg	no chg	no chg	1	no chg	None	N/A	Writing into CNT when TXFIF = 11 sets TXOVF bit. Firmware should always check TXFIF bits before loading.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes TXEPEN and ATM are enabled.
2. Future transactions are NAKed even if the transmit endpoint is stalled when RXSETUP = 1.

Table D-2. Isochronous Transmit Data Flow in Dual-packet Mode

TXFIF (1:0)	Event	New TX FIF (1:0) (2)	(at next SOF)			TX OVF (1,2)	TX URF (1,2)	TX Inter-rupt	USB Response	Comments
			TX ERR	TX ACK	TX Void					
00	Received IN token, but no data or TXOE=0.	00	no chg	no chg	1	no chg	no chg	None	Timeout	No data was loaded, so timeout (i.e., no response). This event should never happen.
	Data loaded into FIFO from CPU, CNT written.	01	no chg	no chg	no chg	no chg	no chg	None	N/A	Firmware should always check TXFIF bits before loading and TXOVF after loading.
	Data loaded into FIFO, FIFO error.	00	no chg	no chg	no chg	1	no chg	None	N/A	Only overrun FIFO error can occur here. Firmware should always check TXOVF before write CNT.
01/10	Received IN token, data transmitted with or without transmission error.	00	0	1	0	no chg	no chg	None	Send data	No ACK (time-out) for ISO. Read marker advanced.
	Received IN token, data transmitted, FIFO error occurs.	00	1	0	0	no chg	1	None	Send CRC with bit-stuff error	Only underrun FIFO error can occur here. Read marker advanced.

NOTES:
1. These are sticky bits, which must be cleared by firmware.
2. TXFIF, TXOVF and TXURF are handled with the following golden rule: Firmware events cause status change immediately while USB events only cause status change at SOF.
   TXOVF: Since overrun can only be caused by firmware, TXOVF is updated immediately.
   TXURF: Since underrun can only be caused by USB, TXURF is updated at SOF.
   TXFIF: TXFIF is “incremented” by firmware and “decremented” by USB. Therefore, writes to TXCNT will “increment” TXFIF immediately. However, a successful USB transaction anytime in a frame will only “decrement” TXFIF at SOF.
   TXERR, TXACK, and TXVOID can only be caused by USB; thus they are updated at the end of every valid transaction.
3. NOTE: This table assumes TXEPEN and ATM are enabled.

Table D-2. Isochronous Transmit Data Flow in Dual-packet Mode (Continued)

TXFIF (1:0)	Event	New TX FIF (1:0) (2)	(at next SOF)			TX OVF (1,2)	TX URF (1,2)	TX Inter-rupt	USB Response	Comments
			TX ERR	TX ACK	TX Void					
	Received IN token with existing FIFO error.	01/10	1 (no chg)	0 (no chg)	1	no chg	1 (no chg)	None	Timeout	Treated like a "void" condition.
	Received IN token, but TXOE = 0.	01/10	0	0	1	no chg	no chg	None	Timeout	Endpoint not enabled for transmit, but no NAK for ISO.
	Data loaded into FIFO from CPU, CNT written.	11	no chg	no chg	no chg	no chg	no chg	None	N/A	Firmware should always check TXFIF bits before loading and TXOVF after loading.
	Data loaded into FIFO, FIFO error occurs.	01/10	no chg	no chg	no chg	1	no chg	None	N/A	Only overrun FIFO error can occur here. Firmware should always check TXOVF before write CNT. Note: no TXERR, but TXOVF set.
11	Received IN token, data transmitted with or without transmission error.	10/01	0	1	0	no chg	no chg	None	Send data	No ACK (time-out) for ISO. Read marker advanced.

NOTES:
1. These are sticky bits, which must be cleared by firmware.
2. TXFIF, TXOVF and TXURF are handled with the following golden rule: Firmware events cause status change immediately while USB events only cause status change at SOF.
   TXOVF: Since overrun can only be caused by firmware, TXOVF is updated immediately.
   TXURF: Since underrun can only be caused by USB, TXURF is updated at SOF.
   TXFIF: TXFIF is "incremented" by firmware and "decremented" by USB. Therefore, writes to TXCNT will "increment" TXFIF immediately. However, a successful USB transaction anytime in a frame will only "decrement" TXFIF at SOF.
   TXERR, TXACK, and TXVOID can only be caused by USB; thus they are updated at the end of every valid transaction.
3. NOTE: This table assumes TXEPEN and ATM are enabled.

Table D-2. Isochronous Transmit Data Flow in Dual-packet Mode (Continued)

TXFIF (1:0)	Event	New TX FIF (1:0) (2)	(at next SOF)			TX OVF (1,2)	TX URF (1,2)	TX Inter-rupt	USB Response	Comments
			TX ERR	TX ACK	TX Void					
	Received IN token, data transmitted, FIFO error occurs.	10/01	1	0	0	no chg	1	None	Send data with bitstuffing error	Only a FIFO underrun error can occur here. Read marker advanced.
	Received IN token with existing FIFO error.	11	1 (no chg)	0 (no chg)	1	no chg	1 (no chg)	None	Timeout	Treated like a “void” condition.
	Received IN token, but TXOE = 0.	11	0	0	1	no chg	no chg	None	Timeout	Endpoint not enabled for transmit, but no NAK for ISO.
	Receive SOF indication.	10/01	no chg	no chg	no chg	no chg	no chg	None (SOF interrupt set) ASOF set.	None	Host never read last frame’s ISO. packet. Read marker and pointer advanced, oldest packet is flushed from FIFO.
	Data loaded into FIFO from CPU, CNT written.	11	no chg	no chg	no chg	1	no chg	None	N/A	CNT written when TXFIF=11 will set TXOVF bit. Firmware should always check TXFIF bits before loading.

NOTES:

1. These are sticky bits, which must be cleared by firmware.
2. TXFIF, TXOVF and TXURF are handled with the following golden rule: Firmware events cause status change immediately while USB events only cause status change at SOF.
   TXOVF: Since overrun can only be caused by firmware, TXOVF is updated immediately.
   TXURF: Since underrun can only be caused by USB, TXURF is updated at SOF.
   TXFIF: TXFIF is “incremented” by firmware and “decremented” by USB. Therefore, writes to TXCNT will “increment” TXFIF immediately. However, a successful USB transaction anytime in a frame will only “decrement” TXFIF at SOF.
   TXERR, TXACK, and TXVOID can only be caused by USB; thus they are updated at the end of every valid transaction.
3. NOTE: This table assumes TXEPEN and ATM are enabled.

Table D-3. Non-isochronous Receive Data Flow in Single-packet Mode (RXSPM = 1)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
00	Received OUT token, but RXIE = 0	00	no chg	no chg	1	no chg	no chg	no chg	None	NAK	FIFO not ready.
	Received OUT token, but timed-out waiting for data.	00	no chg	no chg	no chg	no chg	no chg	no chg	None	None	FIFO not loaded. Write pointer reversed.
	Received OUT token, no errors.	01	0	1	0	0	no chg	no chg	Set receive interrupt	ACK	Received, no errors, advance write marker.
	Received OUT token, data CRC or bit-stuff error.	00	1	0	0	0	no chg	no chg	Set receive interrupt	Time-out	Write pointer reversed. (Possible to have RXERR cleared by hardware before seen by firmware.)
	Received OUT token, FIFO error occurs.	00	1	0	0	0	1	no chg	Set receive interrupt	Time-out, NAK future transactio ns	Only RXOVF FIFO error can occur, requires firmware intervention.
	Received OUT token with FIFO error already existing.	00	1 (no chg)	0 (no chg)	1	0	1 (no chg)	no chg	None	NAK	Considered to be a "void" condition. Will NAK until firmware clears condition.
	Received OUT token, but data sequence mismatch.	00	no chg	no chg	1	no chg	no chg	no chg	None	ACK	Last ACK corrupted, so send again but ignore the data.
	Received SETUP token, no errors.	01	0	1	0	1	0	0	Set receive interrupt	ACK	RXIE or RXSTL has no effect.(2) RXSETUP will be set (control endpoints only).

NOTE:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.

Table D-3. Non-isochronous Receive Data Flow in Single-packet Mode (RXSPM = 1) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received SETUP token, but timed-out waiting for data.	00	1	0	0	0	0	0	Set receive interrupt	Time-out	FIFO is reset automatically and FIFO data is invalid. (2)
	Received SETUP token, data CRC or bit-stuff error.	00	1	0	0	1	0	0	Set receive interrupt	Time-out	Write pointer reversed. (2)
	Received SETUP token, FIFO error occurs.	00	1	0	0	1	1	0	Set receive interrupt	Time-out, NAK future transactio ns	(2)
	Received SETUP token with FIFO error already existing.	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received.RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	CPU reads FIFO, causes FIFO error.	00	no chg	no chg	no chg	no chg	no chg	1	None	NAK future transactio ns, except SETUP	FIFO was empty when read. Should always check RXFIF bits before reading.
01	Received OUT token.	01	no chg	no chg	1	no chg	no chg	no chg	None	NAK	FIFO not ready, so data is ignored (CRC or FIFO error not possible).

NOTE:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.

Table D-3. Non-isochronous Receive Data Flow in Single-packet Mode (RXSPM = 1) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received SETUP token, no errors.	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token, but timed-out waiting for data.	01	1	0	0	0	0	0	Set receive interrupt	Time-out	FIFO is reset automatically and FIFO data is invalid. (2)
	Received SETUP token, data CRC or bit-stuff error.	00	1	0	0	1	0	0	Set receive interrupt	Time-out	Write pointer reversed. RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token, FIFO error occurs.	00	1	0	0	1	1	0	Set receive interrupt	Time-out, NAK future transactio ns	(2) (control endpoints only).
	Received SETUP token with FIFO error already existing.	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	CPU reads FIFO, sets RXFFRC.	00	no chg	no chg	no chg	no chg	no chg	no chg	None	None	

NOTE:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.

Table D-3. Non-isochronous Receive Data Flow in Single-packet Mode (RXSPM = 1) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	CPU reads FIFO, causes FIFO error. RXFFRC not set yet.	01	no chg	no chg	no chg	no chg	no chg	1	None	Time-out, NAK future transactio ns	Firmware should check RXURF bit before writing RXFFRC.
	CPU reads FIFO, causes FIFO error. Set RXFFRC.	00	no chg	no chg	no chg	no chg	no chg	1	None	Time-out, NAK future transactio ns	Firmware should check RXURF bit before writing RXFFRC.

NOTE:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
00	Received OUT token, but RXIE = 0.	00	no chg	no chg	1	no chg	no chg	no chg	None	NAK	FIFO not ready.
	Received OUT token, but timed-out waiting for data.	00	no chg	no chg	1	no chg	no chg	no chg	None	None	FIFO not loaded. Write pointer reversed.
	Received OUT token, no errors.	01	0	1	0	0	no chg	no chg	Set receive interrupt	ACK	Received, no errors, advance write marker.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received OUT token, data CRC or bit-stuff error.	00	1	0	0	0	no chg	no chg	Set receive interrupt	Time-out	Write pointer reversed. (Possible to have RXERR cleared by hardware before seen by firmware.)
	Received OUT token, FIFO error occurs.	00	1	0	0	0	1	no chg	Set receive interrupt	Time-out, NAK future transaction s	Only RXOVF FIFO error can occur, requires firmware intervention.
	Received OUT token with FIFO error already existing.	00	1 (no chg)	0 (no chg)	1	0	1 (no chg)	no chg	None	NAK	Considered to be a "void" condition. Will NAK until firmware clears condition.
	Received OUT token, but data sequence mismatch.	00	no chg	no chg	no chg	no chg	no chg	no chg	None	ACK	Last ACK corrupted, so send again but ignore the data.
	Received SETUP token, no errors (dual packet mode not recommende d!).	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token, but timed-out waiting for data.	00	1	0	0	0	0	0	Set receive interrupt	Time-out	FIFO is reset automatically and FIFO data is invalid. (2)

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received SETUP token, data CRC or bit-stuff error (dual packet mode not recommended).	00	1	0	0	1	0	0	Set receive interrupt	Time-out	Write pointer reversed, RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token, FIFO error occurs.	00	1	0	0	1	1	0	Set receive interrupt	Time-out, NAK future transactions	RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token with FIFO error already existing.	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	CPU reads FIFO, causes FIFO error.	00	no chg	no chg	no chg	no chg	no chg	1	None	NAK future transactions	FIFO was empty when read. Should always check RXFIF bits before reading.
01/10	Received OUT token, but RXIE = 0.	01/10	no chg	no chg	1	no chg	no chg	no chg	None	NAK	FIFO not ready.
	Received OUT token, but timed-out waiting for data.	01/10	no chg	no chg	1	no chg	no chg	no chg	None	None	FIFO not loaded. Write pointer reversed.
	Received OUT token, no errors.	11	0	1	0	0	no chg	no chg	Set receive interrupt	ACK	Received, no errors, advance write marker.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received OUT token, data CRC or bit-stuff error.	01/10	1	0	0	0	no chg	no chg	Set receive interrupt	Time-out	Write pointer reversed. (Possible to have RXERR cleared by hardware before seen by firmware.)
	Received OUT token, FIFO error occurs.	01/10	1	0	0	0	1	no chg	Set receive interrupt	Time-out, NAK future transaction s	Only RXOVF FIFO error can occur, requires firmware intervention.
	Received OUT token with FIFO error already existing.	01/10	1 (no chg)	0 (no chg)	1	0	1 (no chg)	no chg	None	NAK	Considered to be a "void" condition. Will NAK until firmware clears condition.
	Received OUT token, but data sequence mismatch.	01/10	no chg	no chg	no chg	no chg	no chg	no chg	None	ACK	Last ACK corrupted, so send again but ignore the data.
	Received SETUP token, no errors (dual-packet mode not recommende d).	01/10	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token, but timed-out waiting for data.	01/10	1	0	0	0	0	0	Set receive interrupt	Time-out	FIFO is reset automatically, forcing new SETUP to be received. (2)

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received SETUP token, data CRC or bit-stuff error (dual-packet mode not recommended).	00	1	0	0	1	0	0	Set receive interrupt	Time-out	Write pointer reversed. RXIE or RXSTL has no effect. (2)
	Received SETUP token, FIFO error occurs.	00	1	0	0	1	1	0	Set receive interrupt	Time-out, NAK future transactions	RXIE or RXSTL has no effect, (2) RXSETUP will be set (control endpoints only).
	Received SETUP token with FIFO error already existing.	01/10	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. (2) RXSETUP will be set (control endpoints only).
	CPU reads FIFO, sets RXFFRC.	00	no chg	no chg	no chg	no chg	no chg	no chg	None	None	
	CPU reads FIFO, causes FIFO error. RXFFRC not set yet.	01/10	no chg	no chg	no chg	no chg	no chg	1	None	Time-out, NAK future transactions	Firmware should check RXURF bit before writing RXFFRC.
	CPU reads FIFO, causes FIFO error. Set RXFFRC.	00	no chg	no chg	no chg	no chg	no chg	1	None	Time-out, NAK future transactions	Firmware should check RXURF bit before writing RXFFRC.
11	Received OUT token.	11	no chg	no chg	1	no chg	no chg	no chg	None	NAK	FIFO not ready, so data is ignored (CRC or FIFO error not possible).

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	Received SETUP token, no errors (dual-packet mode not recommended!).	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. (2) RXSETUP will be set. (control endpoints only).
	Received SETUP token, but timed-out waiting for data.	11	1	0	0	0	0	0	Set receive interrupt	Time-out	FIFO is reset automatically and FIFO data is invalid. (2)
	Received SETUP token, data CRC or bit-stuff error (dual-packet mode not recommended).	00	1	0	0	1	0	0	Set receive interrupt	Time-out	Write pointer reversed. RXIE or RXSTL has no effect. (2)
	Received SETUP token, FIFO error (dual-packet mode not recommended).	00	1	0	0	1	1	0	Set receive interrupt	Time-out, NAK future transactions	RXIE or RXSTL has no effect. (2) RXSETUP will be set (control endpoints only).
	Received SETUP token with FIFO error already existing.	01	0	1	0	1	0	0	Set receive interrupt	ACK	Causes FIFO to reset automatically, forcing new SETUP to be received. (2) RXSETUP will be set (control endpoints only).
	CPU reads FIFO, sets RXFFRC.	10/01	no chg	no chg	no chg	no chg	no chg	no chg	None	None	

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-4. Non-isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New FIF (1:0)	RX ERR	RX ACK	RX Void	RX Setup	RX OVF (1)	RX URF (1)	RX Inter-rupt	USB Response	Comments
	CPU reads FIFO, causes FIFO error. RXFFRC not written yet.	11	no chg	no chg	no chg	no chg	no chg	1	None	NAKs future transaction s	Firmware should check RXURF bit before writing FFRC.
	CPU reads FIFO, causes FIFO error. Set RXFFRC.	10/01	no chg	no chg	no chg	no chg	no chg	1	None	NAKs future transaction s	Firmware should check RXURF bit before writing FFRC.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. STOVW is set after a valid SETUP token is received and cleared during handshake phase. EDOVW is set during handshake phase.
3. NOTE: Dual-packet mode is NOT recommended for control endpoints.

Table D-5. Isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0)

FIF (1:0)	Event	New RXFIF (1:0) (2)	(at next SOF)			RX OVF (1,2)	RX URF (1,2)	RX Inter-rupt	USB Response	Comments
			RX ERR	RX ACK	RX Void					
00	Received OUT token, but RXIE = 0.	00	no chg	no chg	1	no chg	no chg	None	None/ Time-out	FIFO not ready, or timed-out waiting for data packet, but no NAK sent.
	Received OUT token, but timed-out waiting for data.	00	no chg	no chg	no chg	no chg	no chg	None	None/ Time-out	FIFO not loaded.
	Received OUT token, no errors.	01	0	1	0	no chg	no chg	None	None/ Time-out	Received, no errors, advance write marker.
	Received OUT token, data CRC or bit-stuff error.	01	1	0	0	no chg	no chg	None	None/ Time-out	Bad data still loaded into FIFO.
	Received OUT token, FIFO error occurs.	01	1	0	0	1	no chg	None	None/ Time-out	Only RXOVF FIFO error can occur, requires firmware intervention.
	Received OUT token with FIFO error already existing.	00	1 (no chg)	0 (no chg)	1	1 (no chg)	no chg	None	None/ Time-out	Treated like a "void" condition.
	CPU reads FIFO, causes FIFO error.	00	no chg	no chg	no chg	no chg	1	None	None/ Time-out	FIFO was empty when read. Should always check RXFIF bits before reading.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. RXFIF, RXOVF and RXURF are handled with the following golden rule: Firmware events cause status change immediately while USB events only cause status change at SOF.
   RXURF: Since underrun can only be caused by firmware, RXURF is updated immediately.
   RXOVF: Since overrun can only be caused by USB, RXOVF is updated at SOF.
   RXFIF: RXFIF is "incremented" by USB and "decremented" by firmware. Therefore, setting RXFFRC will "decrement" RXFIF immediately. However, a successful USB transaction anytime in a frame will only "increment" RXFIF at SOF.
   RXERR, RXACK, and RXVOID can only be caused by USB, thus they are updated at the end of transaction.

Table D-5. Isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New RXFIF (1:0) (2)	(at next SOF)			RX OVF (1,2)	RX URF (1,2)	RX Inter-rupt	USB Response	Comments
			RX ERR	RX ACK	RX Void					
	Receive SOF indication.	no chg/up dated	up-dated	up-dated	up-dated	up-dated	no chg	None (SOF interrupt)	None/ Time-out	Flags are updated at SOF. Firmware must check for RXFIF = 00 condition to detect no ISO packet received this frame.
01/10	Received OUT token, but RXIE = 0.	01/10	no chg	no chg	1	no chg	no chg	None	None/ Time-out	FIFO not ready.
	Received OUT token, but timed-out waiting for data.	01/10	no chg	no chg	no chg	no chg	no chg	None	None/ Time-out	FIFO not loaded.
	Received OUT token, no errors.	11	0	1	0	no chg	no chg	None	None/ Time-out	Received, no errors, advance write marker.
	Received OUT token, data CRC or bit-stuff error.	11	1	0	0	no chg	no chg	None	None/ Time-out	Possible to have RXERR cleared by hardware before seen by firmware. Reverse write pointer.
	Received OUT token, FIFO error occurs.	11	1	0	0	1	no chg	None	None/ Time-out	Only OVF FIFO error can occur, requires firmware intervention.
	Received OUT token with FIFO error already existing.	01/10	no chg	no chg	1	no chg	no chg	None	None/ Time-out	Treated like a “void” condition.

NOTES:

1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. RXFIF, RXOVF and RXURF are handled with the following golden rule: Firmware events cause status change immediately while USB events only cause status change at SOF.
   RXURF: Since underrun can only be caused by firmware, RXURF is updated immediately.
   RXOVF: Since overrun can only be caused by USB, RXOVF is updated at SOF.
   RXFIF: RXFIF is “incremented” by USB and “decremented” by firmware. Therefore, setting RXFFRC will “decrement” RXFIF immediately. However, a successful USB transaction anytime in a frame will only “increment” RXFIF at SOF.
   RXERR, RXACK, and RXVOID can only be caused by USB, thus they are updated at the end of transaction.

Table D-5. Isochronous Receive Data Flow in Dual-packet Mode (RXSPM = 0) (Continued)

FIF (1:0)	Event	New RXFIF (1:0) (2)	(at next SOF)			RX OVF (1,2)	RX URF (1,2)	RX Inter-rupt	USB Response	Comments
			RX ERR	RX ACK	RX Void					
	CPU reads FIFO, sets RXFFRC.	00	no chg	no chg	no chg	no chg	no chg	None	None/ Time-out	
	CPU reads FIFO, causes FIFO error.	00	no chg	no chg	no chg	no chg	1	None	None/ Time-out	Firmware should check RXURF bit before writing RXFFRC.
11	Received OUT token.	11	no chg	no chg	1	no chg	no chg	None	None/ Time-out	FIFO not ready, but data must be taken. This situation should never happen.
	Received SOF indication.	no chg/ up-dated	up-dated	up-dated	up-dated	up-dated	no chg	None (SOF interrupt)	None/ Time-out	Error condition (not handled by hardware). Firmware should not allow this condition.
	CPU reads FIFO, sets RXFFRC.	10 or 01	no chg	no chg	no chg	no chg	no chg	None	None/ Time-out	
	CPU reads FIFO, causes FIFO error. RXFFRC not set yet.	11	no chg	no chg	no chg	no chg	1	None	None/ Time-out	Firmware should check RXURF bit before writing RXFFRC.
	CPU reads FIFO, causes FIFO error. Set RXFFRC.	10 or 01	no chg	no chg	no chg	no chg	1	None	None/ Time-out	Firmware should check RXURF bit before writing RXFFRC.

NOTES:
1. These are sticky bits, which must be cleared by firmware. Also, this table assumes RXEPEN and ARM are enabled.
2. RXFIF, RXOVF and RXURF are handled with the following golden rule: Firmware events cause status change immediately while USB events only cause status change at SOF.
   RXURF: Since underrun can only be caused by firmware, RXURF is updated immediately.
   RXOVF: Since overrun can only be caused by USB, RXOVF is updated at SOF.
   RXFIF: RXFIF is "incremented" by USB and "decremented" by firmware. Therefore, setting RXFFRC will "decrement" RXFIF immediately. However, a successful USB transaction anytime in a frame will only "increment" RXFIF at SOF.
   RXERR, RXACK, and RXVOID can only be caused by USB, thus they are updated at the end of transaction.

E

8x931AA Design Considerations

APPENDIX E
8*x*931AA DESIGN CONSIDERATIONS

This appendix describes the differences between the hubless 8*x*931AA and the 8*x*931HA. The 8*x*931HA is described in the rest of this document.

E.1 DIFFERENCES BETWEEN THE 8*x*931AA AND THE 8*x*931HA

The 8*x*931AA differs from the 8*x*931HA in many ways, including:

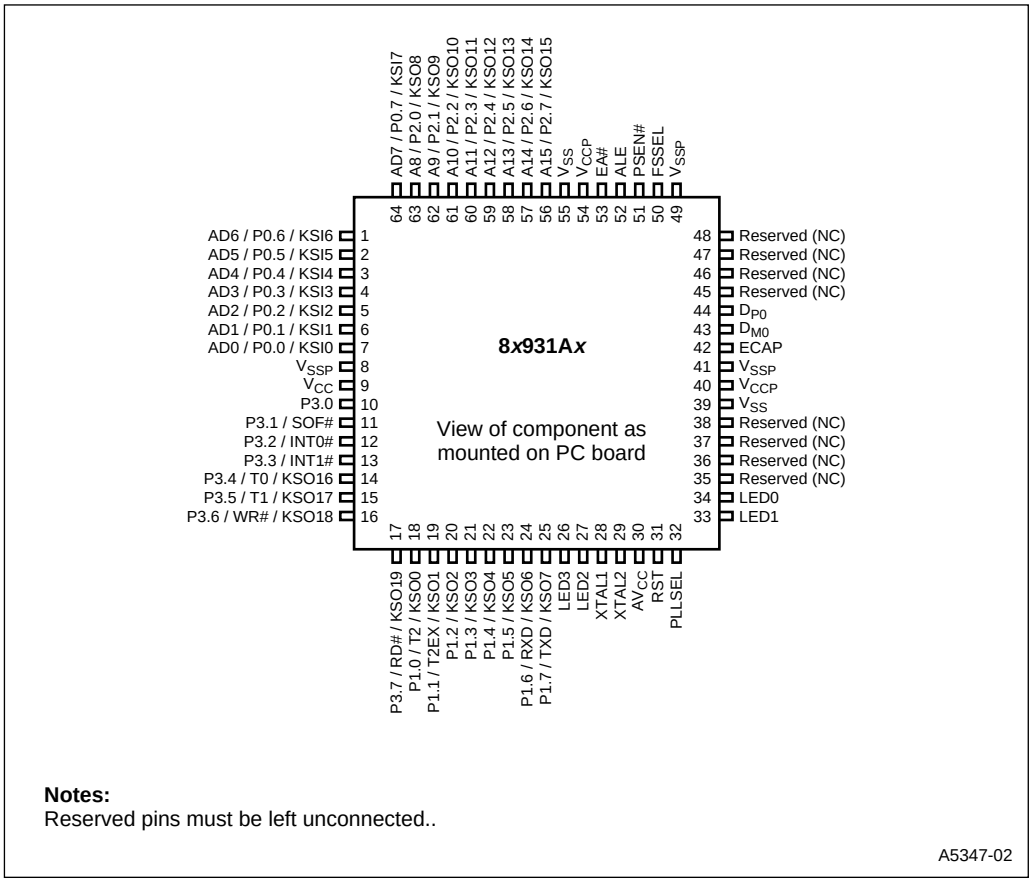
- The 8*x*931AA does not support hub operations and has no hub interface, hub repeater, or hub FIFOs. These features, included in the 8*x*931HA only, are discussed in “Universal Serial Bus Module” on page 2-11 and “Hub Operation” on page 8-17.
- The 8*x*931AA has no hub interrupt. The 8*x*931HA hub interrupt is discussed in “USB Hub Interrupt” on page 5-15.
- Because there is no on-chip hub, Chapter 7, “USB Hub” does not apply to the 8*x*931AA.
- The 8*x*931AA has no Hub Address Register (HADDR), so its enumeration process is simpler than the 8*x*931HA enumeration process given in “Enumeration” on page 8-2. The 8*x*931AA enumeration process is given in “8x931AA Enumeration Process” on page E-2
- The hub programming models described in “Hub Operation” on page 8-17 do not apply to the 8*x*931AA.
- Since the 8*x*931AA peripheral controller does not support a hub interface, there are no downstream ports to signal a resume condition. A resume condition can still be signalled by any of the other conditions described in “Global Resume Mode” on page 14-9.
- The 8*x*931HA SFR map given in Appendix C, “Registers” does not apply to the 8*x*931AA. The 8*x*931AA SFRs are given in “8x931AA SFR Map” on page E-10.
- The 8*x*931AA pin and signal descriptions differ from those described in Appendix B, “Pin Descriptions”. See “8x931AA Pin Descriptions” on page E-3 and “8x931AA Signal Descriptions” on page E-6.
- The 8*x*931AA allows operating frequency selection using the FSSEL pin. See “Operating Frequencies” on page E-9.

E.2 8*x*931AA ENUMERATION PROCESS

The 8*x*931AA enumeration process is simpler than the 8*x*931HA bus enumeration process given in "Enumeration" on page 8-2. The 8*x*931AA enumeration process has four steps:

1. Get device descriptor. The host requests and reads the device descriptor to determine maximum packet size.
2. Set address. The host sends the 8*x*931's function address in a data packet using function endpoint 0. Device firmware interprets the data and instructs the CPU to write the function address to FADDR.
3. Get device descriptor. The host requests and reads the device descriptor to determine such information as device class, USB specification compliance level, maximum packet size for endpoint 0, vendor id, product id, etc.
4. Get configuration descriptor. The host requests and reads the device configuration descriptor to determine such information as the number of interfaces and endpoints; endpoint transfer type, packet size, and direction; power source; maximum power; etc. For detailed information on configuration descriptors, see the "Device Framework" chapter in *Universal Serial Bus Specification*. When the host requests the configuration descriptor, all related interface and endpoint descriptors are returned.
5. Set configuration. The host assigns a configuration value to the device to establish the current configuration. Devices can have multiple configurations.

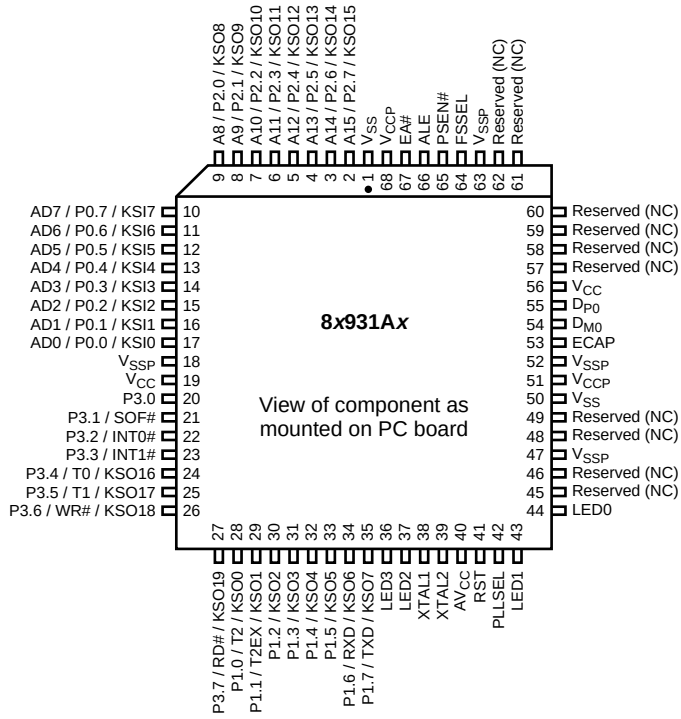
E.3 8*x*931AA PIN DESCRIPTIONS



Notes:
Reserved pins must be left unconnected..

A5347-02

Figure E-1. 8*x*931AA 64-pin QFP Package



Note:
Reserved pins must be left unconnected.

A5348-02

Figure E-2. 8*x*931AA 68-pin PLCC

Table E-1. 8*x*931AA Signals Arranged by Functional Category

Address and Data	
AD7:0	I/O
A15:8	I/O

Keyboard Scan I/O	
KSO19:0	O
KSI7:0	I

LED Drivers	
LED3:0	O

Bus Control	
ALE	O
PSEN#	O
RD#	O
WR#	O

Input/Output	
P0.7:0	I/O
P1.7:0	I/O
P2.7:0	I/O
P3.7:0	I/O
T0	I
T1	I

Processor Control	
EA#	I
INT0#	I
INT1#	I
RST	I
XTAL1	I
XTAL2	O

USB	
D_{M0}	I/O
D_{P0}	I/O
ECAP	I
PLLSEL	I
SOF#	O
FSSEL	O

Power & Ground	
AV_{CC}	PWR
V_{CC}	PWR
V_{CCP}	PWR
V_{SS}	GND
V_{SSP}	GND

E.4 8*x*931AA SIGNAL DESCRIPTIONS

Table E-2. 8*x*931AA Signal Descriptions

Signal Name	Type	Description	Alternate Function
A15:8	O	**Address Lines**. Upper byte of external memory address.	P2.7:0/KSO15:7
AD7:0	I/O	**Address/Data Lines**. Lower byte of external memory address multiplexed with data	P0.7:0/KSI7:0
ALE	O	**Address Latch Enable**. ALE signals the start of an external bus cycle and indicates that valid address information is available on lines A15:8 and AD7:0. An external latch can use ALE to demultiplex the address from the address/data bus.	—
AV_{CC}	PWR	**Analog V_{CC}**. A separate V_{CC} input for the phase-locked loop circuitry.	—
D_{M0}, D_{P0}	I/O	**USB Port 0**. D_{P0} and D_{M0} are the data plus and data minus lines of USB port 0, the upstream differential port. These lines do not have internal pullup resistors. For low-speed devices, provide an external 1.5 KΩ pullup resistor at D_{M0}. For full-speed devices, provide an external 1.5 KΩ pullup resistor at D_{P0}. NOTE: For the 8*x*931AA, either D_{P0} or D_{M0} must be pulled high. Otherwise a continuous SEO (USB reset) will be applied to these inputs causing the 8*x*931AA to stay in reset.	—
EA#	I	**External Access**. Directs program memory accesses to on-chip or off-chip code memory. For EA# strapped to ground, all program memory accesses are off-chip. For EA# strapped to V_{CC}, program accesses on-chip ROM if the address is within the range of the on-chip ROM; otherwise the access is off-chip. The value of EA# is latched at reset. For devices without on-chip ROM, EA# must be strapped to ground.	—
ECAP	I	**External Capacitor**. Connect a 1 µF or larger capacitor between this pin and V_{SS} to ensure proper operation of the differential line drivers.	—
FSSEL		**Full-speed Select.** See Table E-3 on page E-9.	—
INT1:0#	I	**External Interrupts 0 and 1**. These inputs set the IE1:0 interrupt flags in the TCON register. Bits IT1:0 in TCON select the triggering method: edge-triggered (high-to-low) or level triggered (active low). INT1:0 also serves as external run control for timer1:0 when selected by GATE1:0# in TCON.	P3.3:2
KSI7:0	I	**Keyboard Scan Input.** Schmitt-trigger inputs with firmware-enabled internal pullup resistors used for the input side of the keyboard scan matrix.	AD7:0/P0.7:0
KSO19 KSO18 KSO17:16 KSO15:8 KSO7:0	O	**Keyboard Scan Output.** Quasi-bidirectional ports with weak internal pullup resistors used for the output side of the keyboard scan matrix.	P3.7/RD# P3.6/WR# P3.5:4/T1:0 A15:8/P2.7:0 P1.7:0

Table E-2. 8*x*931AA Signal Descriptions (Continued)

Signal Name	Type	Description	Alternate Function
LED3:0	O	**LED Drivers.** Designed to drive LEDs connected directly to V_{CC}. The current each driver is capable of sinking is given as V_{OL2}.	—
P0.7:0	I/O	**Port 0**. Eight-bit, open-drain, bidirectional I/O port. Port 0 pins have Schmitt trigger inputs.	AD7:0/KSI7:0
P1.7:0	I/O	**Port 1**. Eight-bit quasi-bidirectional I/O port with internal pullups.	KSO7:0
P2.7:0	I/O	**Port 2**. Eight-bit quasi-bidirectional I/O port with internal pullups.	A15:8/KSO15:8
P3.0 P3.1 P3.2 P3.3 P3.4 P3.5 P3.6 P3.7	I/O	**Port 3**. Eight-bit quasi-bidirectional I/O port with internal pullups.	 SOF# INT0# INT1# T0/KSO16 T1/KSO17 WR#/KSO18 RD#/KSO19
PLLSEL	I	**Phase-locked Loop Select**. See Table E-3 on page E-9.	—
PSEN#	O	**Program Store Enable**. Read signal output. Asserted for read accesses to external program memory.	—
RD#	O	**Read.** Read signal output. Asserted for read accesses to external data memory.	P3.7/KSO19
RST	I	**Reset**. Reset input to the chip. Holding this pin high for 64 oscillator periods while the oscillator is running resets the device. The port pins are driven to their reset conditions when a voltage greater than V_{IH1} is applied, whether or not the oscillator is running. This pin has an internal pulldown resistor which allows the device to be reset by connecting a capacitor between this pin and V_{CC}. Asserting RST when the chip is in idle mode or powerdown mode returns the chip to normal operation.	—
RXD	I/O	**Receive Serial Data**. RXD sends and receives data in serial I/O mode 0 and receives data in serial I/O modes 1,2, and 3.	P1.6
SOF#	O	**Start of Frame**. Start of frame pulse. Active low. Asserted for 8 states (see Table 2-3 on page 2-9) when frame timer is locked to USB frame timing and SOF token or artificial SOF is detected.	P3.1
T1:0	I	**Timer 1:0 External Clock Input**. When timer 1:0 operates as a counter, a falling edge on the T1:0 pin increments the count.	P3.5:4/KSO17:16
T2	I/O	**Timer 2 Clock Input/Output**. For the timer 2 capture mode, this signal is the external clock input. For the clock-out mode, it is the timer 2 clock output.	P1.0

Table E-2. 8*x*931AA Signal Descriptions (Continued)

Signal Name	Type	Description	Alternate Function
T2EX	I	**Timer 2 External Input**. In timer 2 capture mode, a falling edge initiates a capture of the timer 2 registers. In auto-reload mode, a falling edge causes the timer 2 registers to be reloaded. In the up-down counter mode, this signal determines the count direction: 1=up, 0=down.	P1.1
TXD	O	**Transmit Serial Data**. TXD outputs the shift clock in serial I/O mode 0 and transmits serial data in serial I/O modes 1, 2, and 3.	P1.7
V_{CC}	PWR	**Supply Voltage**. Connect this pin to the +5V supply voltage.	—
V_{CCP}	PWR	**Supply Voltage for I/O buffers**. Connect this pin to the +5V supply voltage.	—
V_{SS}	GND	**Circuit Ground**. Connect this pin to ground.	—
V_{SSP}	GND	**Circuit Ground for I/O buffers**. Connect this pin to ground.	—
WR#	O	**Write.** Write signal output to external memory.	P3.6/KSO19
XTAL1	I	**Oscillator Amplifier Input**. When implementing the on-chip oscillator, connect the external crystal or ceramic resonator across XTAL1 and XTAL2. If an external clock source is used, connect it to this pin.	—
XTAL2	O	**Oscillator Amplifier Output**. When implementing the on-chip oscillator, connect the external crystal or ceramic resonator across XTAL1 and XTAL2. If an external oscillator is used, leave XTAL2 unconnected.	—

E.5 OPERATING FREQUENCIES

8*x*931AA operating frequencies and USB rates are shown in Table E-3.

Table E-3. 8*x*931AA Operating Frequencies

PLLSEL Pin	FSSEL Pin	LC Bit (1)	XTAL1 Frequency (MHz)	USB Rate (FS/LS) (2)	Core Frequency F_{CLK} (Mhz)	Comment
0	0	0	6	LS	3	PLL Off
0	0	1	6	LS	3	PLL Off
1	0	0	12	LS	6	PLL Off
1	0	1	12	LS	3	PLL Off
1	1	0	12	FS	6	PLL On
1	1	1	12	FS	3	PLL On

NOTES:
1. Reset and power up routines set the LC bit in PCON to put the 8*x*931AA in low-clock mode (core frequency = 3 MHz) for lower I_{CC} prior to device enumeration. Following completion of device enumeration, firmware should clear the LC bit to exit the low-clock mode. The user may switch the core frequency back and forth at any time, as needed.
2. USB rates: Low speed = 1.5 Mbps; Full speed = 12 Mbps. The USB sample rate is 4X the USB rate.

E.6 8*x*931AA SFR MAP

The 8*x*931AA SFR map (Table E-4 on page E-10) is identical to the 8*x*931HA SFR map, except the 8*x*931AA has no hub-related SFRs.

Table E-4. 8*x*931AA SFR Map

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
F8	KBCON 0xx00000								FF
F0	B 00000000	EPINDEX 1xxxxx00	TXSTAT 0xx00000	TXDAT xxxxxxxx†	TXCON 0xxx0100†	TXFLG 00xx1000	TXCNTL xxxxxxxx†		F7
E8									EF
E0	ACC 00000000	EPCON 00d10d0d†	RXSTAT 00000000	RXDAT xxxxxxxx	RXCON 0xx00100	RXFLG 00xx1000	RXCNTL xxxxxxxx†		E7
D8								PCON1 xxxxx000	DF
D0	PSW 00000000		SOFL 00000000	SOFH 00001000					D7
C8	T2CON 00000000	T2MOD xxxx xx00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0	FIFLG xx000000								C7
B8	IPL0 x0000000	SADEN 00000000							BF
B0	P3 11111111	IEN1 00000000	IPL1 00000000	IPH1 00000000				IPH0 00000000	B7
A8	IEN0 00000000	SADDR 00000000							AF
A0	P2 11111111		FIE xx000000						A7
98	SCON 00000000	SBUF xxxxxxxx							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		FADDR 00000000	8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 001d0000	87
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

MCS® 51 microcontroller SFRs

Endpoint-indexed SFRs

† For EPCON, TXCON, TXDAT, TXCNTL, and RXCNTL the reset value depends on the endpoint pair selected. Refer to the register definition tables in Appendix C or Chapter 6, "USB Function".

NOTE: "d" in the SFR reset value denotes configuration/operation dependence. Refer to the specific SFR description for more details.

Glossary

GLOSSARY

This glossary defines acronyms, abbreviations, and terms that have special meaning in this manual. (“Guide to this Manual” on page 1-1 discusses notational conventions and general terminology.)

#data An 8-bit constant that is immediately addressed in an instruction.

#data16 A 16-bit constant that is immediately addressed in an instruction.

ACK Acknowledgment. Handshake packet indicating a positive acknowledgment.

accumulator Instructions in the MCS® 51 architecture use the accumulator as both source and destination for calculations and moves..

addr11 An 11-bit destination address. The destination can be anywhere in the same 2 Kbyte block of memory as the first byte of the next instruction.

addr16 A 16-bit destination address. The destination can be anywhere within the same 64 Kbyte region as the first byte of the next instruction.

ALU Arithmetic-logic unit. The part of the CPU that processes arithmetic and logical operations.

assert The term *assert* refers to the act of making a signal active (enabled). The polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To *assert* RD# is to drive it low; to *assert* ALE is to drive it high.

big endien form Method of storing data that places the most significant byte at lower storage addresses.

bit A binary digit.

bit (operand) An addressable bit in the 8*x*931 architecture.

bit stuffing Insertion of a ‘0’ bit into a data stream to cause an electrical transition on the data wires allowing a PLL to remain locked.

bulk transfer Non-periodic, large, “bursty” communication typically used for a transfer that can use any available bandwidth and can also be delayed until bandwidth is available.

bus enumeration Detecting and identifying USB devices.

byte Any 8-bit unit of data.

clear The term *clear* refers to the value of a bit or the act of giving it a value. If a bit is *clear*, its value is "0"; *clearing* a bit gives it a "0" value.

code memory See *program memory*.

control transfer One of four USB transfer types. Control transfers support configuration/command /status type communications between client and function.

dir8 An 8-bit direct address. This can be a memory address or an SFR address.

DPTR The 16-bit data pointer.

deassert The term *deassert* refers to the act of making a signal inactive (disabled). The polarity (high/low) is defined by the signal name. Active-low signals are designated by a pound symbol (#) suffix; active-high signals have no suffix. To *deassert* RD# is to drive it high; to *deassert* ALE is to drive it low.

device address The address of a device on the Universal Serial Bus. The device address is the default address when the USB device is first powered or reset. Hubs and functions are assigned a unique device address by USB firmware.

doping The process of introducing a periodic table Group III or Group V element into a Group IV element (e.g., silicon). A Group III impurity (e.g., indium or gallium) results in a *p-type* material. A Group V impurity (e.g., arsenic or antimony) results in an *n-type* material.

edge-triggered The mode in which a device or component recognizes a falling edge (high-to-low transition), a rising edge (low-to-high transition), or a rising or falling edge of an input signal as the assertion of that signal. See also *level-triggered*.

encryption array An array of key bytes used to encrypt user code as it is read from code memory; protects against unauthorized access to user's code.

endpoint A uniquely identifiable portion of a USB device that is the source or sink of information in a commun-ication flow between the host and the device.

EPP Endpoint pair. See *endpoint*.

external address A 16-bit address presented on the device pins. The address decoded by an external device depends on how many of these address bits the external system uses. See also *internal address*.

F_{CLK}	Microcontroller internal clock frequency distributed to the CPU and on-chip peripherals.
FET	Field-effect transistor.
FIFO	First-in, first-out data buffer. Each USB endpoint pair has a transmit FIFO and a receive FIFO.
FIU	Function Interface Unit. Its function is to manage the data transaction that goes between the 8*x*931 and the USB host based on the transfer type and the FIFOs condition.
F_{OSC}	Frequency at pin XTAL1. The frequency of the on-chip oscillator or external source.
frame	The time from the start of one SOF token to the start of the subsequent SOF token (1 msec); consists of a series of transactions.
function	A USB device that provides a capability to the host. For example, an ISDN connection, a digital microphone, or speakers.
handshake packet	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NACK.
HIU	Hub Interface Unit.
host	The host computer system where the USB host controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
hub	A Universal Serial bus device that provides additional connections to the Universal Serial Bus.
idle mode	The power conservation mode that freezes the core clocks but leaves the peripheral clocks running.
input leakage	Current leakage from an input pin to power or ground.
integer	Any member of the set consisting of the positive and negative whole numbers and zero.
internal address	The 16-bit address that the device generates. See also *external address*.
interrupt handler	The module responsible for handling interrupts that are to be serviced by user-written interrupt service routines.
interrupt latency	The delay between an interrupt request and the time when the first instruction in the interrupt service routine begins execution.
interrupt response time	The time delay between an interrupt request and the resulting break in the current instruction stream.

interrupt service routine	**(ISR)** The firmware routine that services an interrupt.
interrupt transfer	One of four USB transfer types. Interrupt transfer characteristics are small data, non periodic, low frequency, bounded latency, device initiated communication typically used to notify the host of device service needs.
ISO	Isochronous
isochronous data	A stream of data whose timing is implied by its delivery rate.
isochronous transfer	One of four USB transfer types, isochronous transfers provide periodic, continuous communication between host and device.
level-triggered	The mode in which a device or component recognizes a high level (logic one) or a low level (logic zero) of an input signal as the assertion of that signal. See also *edge-triggered.*
low-clock mode	The default mode upon reset, low-clock mode ensures that the I_{CC} drawn by the 8*x*931 is less than one unit load (F_{CLK} = 3MHz).
machine cycle	One machine cycle equals six state times.
maskable interrupt	An interrupt that can be disabled (masked) by its individual mask bit in an interrupt enable register.
MSB	Most-significant bit of a byte or most-significant byte of a word.
multiplexed bus	A bus on which the data is time-multiplexed with (some of) the address bits.
***n*-channel FET**	A field-effect transistor with an *n*-type conducting path (channel).
***n*-type material**	Semiconductor material with introduced impurities (*doping*) causing it to have an excess of negatively charged carriers.
nonmaskable interrupt	An interrupt that cannot be disabled (masked).
***npn* transistor**	A transistor consisting of one part *p*-type material and two parts *n*-type material.
NRZI	Non Return to Zero Invert. A method of encoding serial data in which ones and zeroes are represented by opposite and alternating high and low voltages where there is no return to zero (reference) voltage between encoded bits. Eliminates the need for clock pulses.
***p*-channel FET**	A field-effect transistor with a *p*-type conducting path.
***p*-type material**	Semiconductor material with introduced impurities (*doping*) causing it to have an excess of positively charged carriers.
PC	Program counter.

phase-locked loop A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.

PID Packet ID. A field in a USB packet that identifies the type of packet and hence its format.

PLL See *phase-locked loop*.

program memory A part of memory where instructions can be stored for fetching and execution.

powerdown mode The power conservation mode that freezes both the core clocks and the peripheral clocks.

rel A signed (two's complement) 8-bit, relative destination address. The destination is -128 to +127 bytes relative to the first byte of the next instruction.

reserved bits Register bits that are not used in this device but may be used in future implementations. Avoid any firmware dependence on these bits. In the 8*x*931, the value read from a reserved bit is indeterminate; do not write a "1" to a reserved bit.

resume Once a device is in the suspend state, its operation can be resumed by receiving non-idle signaling on the bus. See also *suspend*.

RT Real-time

root hub A USB hub directly attached to the host controller. This hub is attached to the host; tier 0.

root port The upstream port on a hub.

SE0 Single-ended zero. This is a reference to the USB reset signal which is defined as both D_{P0} and D_{M0} below their threshold voltage.

SIE Serial Bus Interface Engine. Handles the communications protocol of the USB.

set The term *set* refers to the value of a bit or the act of giving it a value. If a bit is *set*, its value is "1"; *setting* a bit gives it a "1" value.

SFR A special function register that resides in its associated on-chip peripheral or in the 8*x*931 core.

sink current Current flowing **into** a device to ground. Always a positive value.

SOF Start of Frame. The SOF is the first transaction in each frame. SOF allows endpoints to identify the start of frame and synchronize internal endpoint clocks to the host.

source current	Current flowing **out of** a device from V_{CC}. Always a negative value.
SP	Stack pointer.
state time (or state)	The basic time unit of the device; With XTAL1 = 12 MHz and two T_{OSC} periods per state, one state = 166.7 ns.
suspend	A low current mode used when the USB bus is idle. The 8*x*931 enters suspend when there is a constant idle state on the bus lines for more than 3.0 msec. When a device is in suspend state, it draws less than 500 µA from the bus. See also *resume*.
token packet	A type of packet that identifies what transaction is to be performed on the bus.
USB	Universal Serial Bus. An industry-standard extension to the PC architecture with a focus on Computer Telephony Integration (CTI), consumer, and productivity applications.

Index

J

K

L

M

N

O

P

R

S

T

U

V

W

X