PHILIPS

Philips Semiconductors

Interconnectivity

December 1998

PDIUSBH11A Application Notes

PDIUSBH11A Application Notes

Overview

These application notes describe the implementation of a USB hub using Philips Semiconductors PDIUSBH11A (H11A). The H11A is pin-compatible to the PDIUSBH11 (H11) and the firmware is backward compatible. It provides the full USB Hub functionality and includes many improvements over the H11, such as:

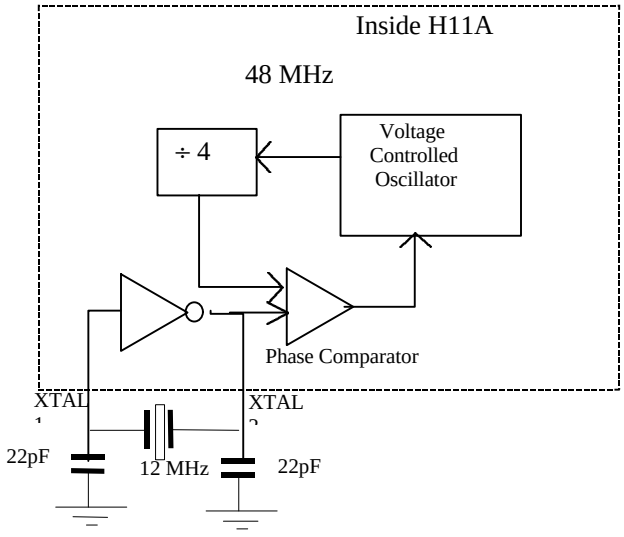
A 12 MHz crystal
Enhanced supply ripple and EMI suppression
Lazyclock™ technology
SOFTCONNECT™ technology
Enhanced functionalities such as GoodLink™
3 embedded ports

12 MHz crystal

Using a 12 MHz crystal instead of a 48 MHz third overtone crystal significantly reduces the cost of the oscillator circuitry because it uses fewer and less costly components. The internal Phase Lock Loop (PLL) locks onto the 12 MHz clock and generates a 48 MHz for internal use. The output clock frequency from the H11A, which is supplied to the micro-controller, is now programmable. This allows low cost micro-controllers to be used. Reducing the oscillator frequency also helps to reduce electromagnetic interference (EMI).

The USB traffic runs at 12 MHz. Therefore a sampling rate of at least twice this speed is required to determine the bits that are sent via the USB bus. In the H11, this is done at 48 MHz. An external clock source or a crystal oscillator circuit provides this clock frequency. Using an external oscillator is very expensive. On the other hand, to implement a third overtone circuit requires additional part count to the circuit board and increases the overall size of the circuit board. In the H11A, all these problems are solved by using an internal Phase Lock Loop (PLL) to generate the internally required 48 MHz clock rate while running at an external crystal frequency of 12 MHz. This is shown in Figure 1.

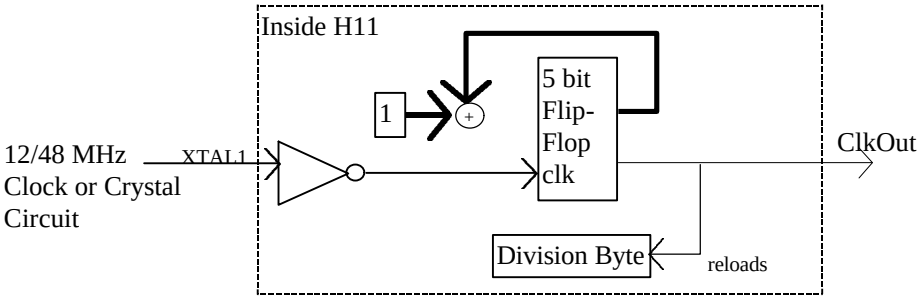
Figure 1: External 12 MHz Crystal Oscillator Circuit



External Clocking

The H11A oscillator clock also supplies the clocking for the external micro-controller. This clock frequency is programmable. It has two variables that can generate a clock frequency ranging from 4 MHz to 24 MHz. The first variable selects either a 12 MHz clock source or a 48 MHz clock source by hardwiring the TEST2 and TEST1 pins. The second variable is the clock division byte. The clock rate is further sub-divided into a lower frequency by a divider whose reload value is software controlled. It is shown in Figure 2. Hence, the user could use a low-end micro-controller that has a low operating frequency, yet drive it with the H11A.

Figure 2: Clockout Division

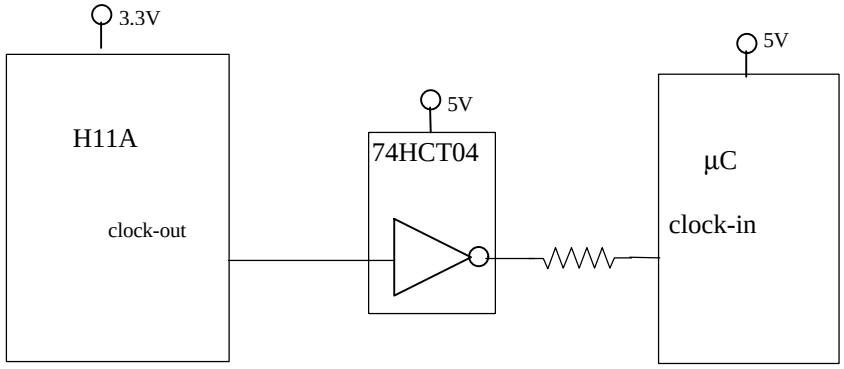


Pins			Default	
Test2	Test1	Input Freq	Division Byte	Clkout
0	1	12 MHz	11	4MHz
1	0	12 MHz	11	4MHz
0	0	48 MHz	3	12MHz
1	1	48 MHz	3	12MHz

Clock Output

The clock output from the H11A has a peak voltage of 3.3V. This is converted into a clock output with a peak voltage of 5V via a 74HCT04 inverter. In addition, the clock output is connected through a 33 Ω resistor to limit the transition time of the signal going into the clock input of the micro-controller. See Figure 3. This reduces electromagnetic radiation by lowering the strength of the harmonics originating from the sharp rising and falling edge of the clock. They are one of the sources that contribute to EMI.

Figure 3: Clock input for the micro-controller



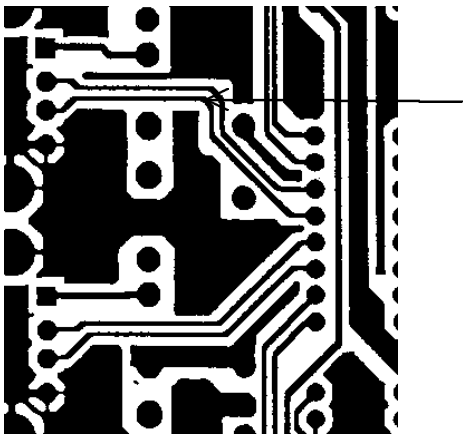
Enhanced supply ripple and EMI suppression

Suppressing ripples on the power supply prevents many problems that are related to internal logic glitches from occurring. ICs that are not well de-coupled might behave in an erratic way and much painstaking effort may be required to trace the problems that are difficult to pinpoint. Additionally, the supply ripples contribute to EMI.

These are some precautions that have been implemented on the H11A demonstration board to prevent ripples from being injected into the system See Figure 4.

The power supply lines are used to shield high bandwidth signal lines.
Complementary lines are put as close together as possible.
Break/rejoin tracks on the PCB are avoided.
Physical irregularities on a signal line, which add to RF emission due to signal leakage, are minimized.

Figure 4: PCB Layout of D+ & D- lines



LazyClock™

In the event of a suspend condition, the clock output of the H11A switches to a slow clock at 24kHz called the LazyClock™. This hardware simplifies the firmware design. It effectively transfers the power conservation responsibility to the H11A , removing the need to create a “sleep condition” in the software.

PDIUSBH11A Application Notes

SOFTCONNECT™

SOFTCONNECT™ allows software control of attachment and detachment of the H11A. This is done by floating the D+ and D- bus when detaching the H11A hub and pulling-up of the D+ line when attaching it. With this capability, the micro-controller is able to complete the initialization process before doing a software-controlled connection. Another benefit is that the device can initiate a detachment and attachment, thereby prompting the host into doing a bus reset to the device and reloading all necessary device drivers.

A bus-powered device may not be stable due to a slow V_{cc} rise time. This causes wrong speed detection by the host or an upstream hub. The device may not reset in time and hence could not "ACK" the upstream requests.

Figure 5 shows the pull-ups at the D+ line of the USB bus for a full-speed device.

To counter these effects, the H11A incorporates SOFTCONNECT™ technology on the upstream port. It provides a software pull-up to the 3.3V. As shown in Figure 6, the pull-up resistor is absorbed into the H11A. Hence the firmware can now choose to make its presence known to the host.

The steps to do a SOFTCONNECT™ involve setting the mode register as shown.

I^2C traffic
Write to slave (H11A): Address(0x36) Data(0xF3)
Write to slave (H11A): Address(0x34) Data(0x??) Data(0x??)

In order to prevent supplying power to upstream in a self-powered system when there is a loss of power on the upstream, the H11A will automatically break the internal resistor connection when Vbus disappears. The sensing of Vbus is provided through OCURRENT_N pins (see the H11A datasheet for more details).

PDIUSBH11A Application Notes

Figure 5: Pull-ups of Full Speed device at D+ line

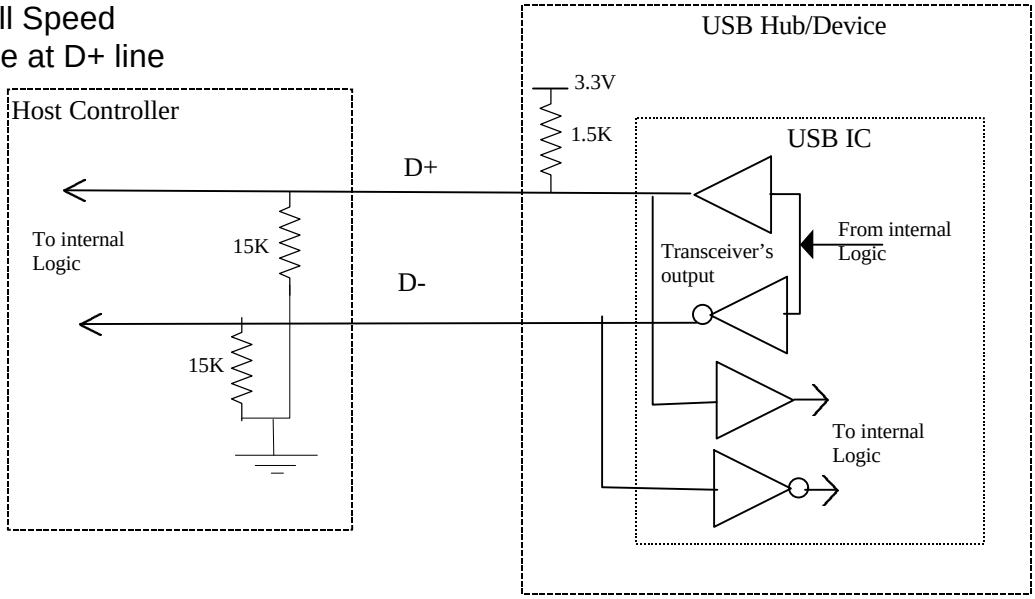
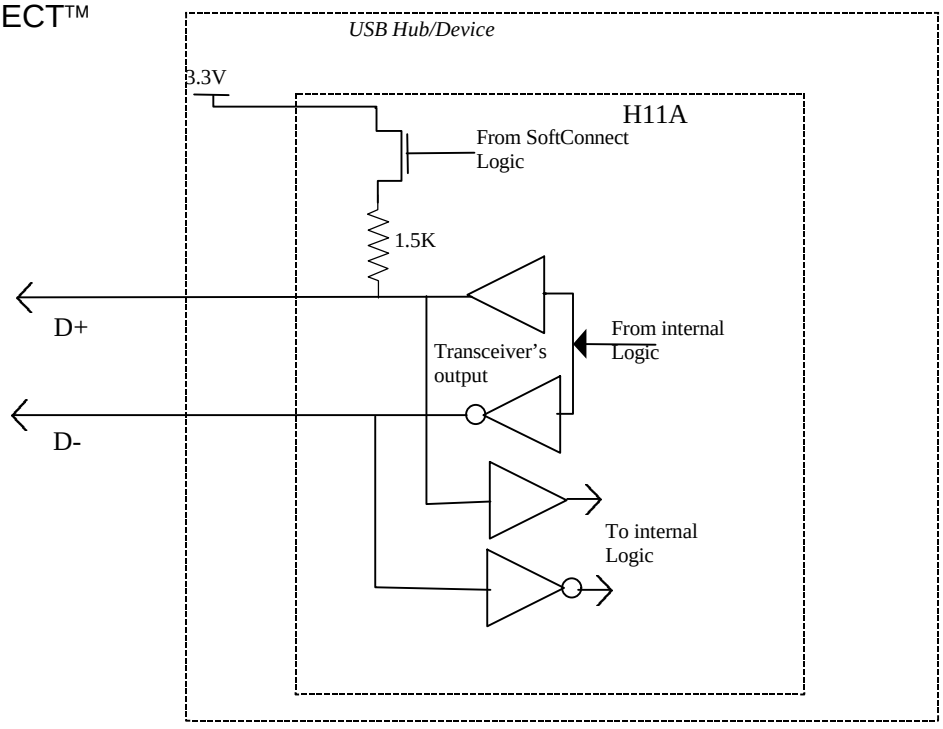


Figure 6: SOFTCONNECT™ of H11A



GoodLink™

In the H11, when you attach a downstream device, an LED indicates that the downstream port is enabled. This provides the only visual cue to downstream functionality. However, in many instances, such feedback is insufficient because the enabled LED may remain lit even when the downstream device fails. GoodLink™ increases the visual feedback by introducing a blinking 5 Hz signal on the port-enable LED whenever USB traffic is detected at the downstream port. To a lay user, a blinking LED is an indication that the device connected to the downstream port is working normally. This enhances the aesthetic value of the USB hub.

The H11A has a GoodLink™ feature that blinks the enabled Light Emitting Diodes (LED's) of the downstream ports whenever there is an ACK of the data on the IN token of the downstream ports. Upon successful enumeration of the downstream device, the enabled LED on the downstream port will be lit (enabled). This LED will blink at 5 Hz when there is traffic on the USB port. In the event of a detachment or device failure, the LED will remain unlit (disabled).

3 embedded ports

The H11A can also be configured to have 3 embedded ports with a maximum of 12 endpoints. The backward compatibility of a single embedded function remains available.

Firmware for the additional ports

The firmware for the H11A is similar to that of the H11, with additional functionalities to control the addition embedded ports.

These are shown on the table below.

Endpoint	Endpoint Number	Index
Hub Control OUT	0	0
Hub Control IN	0	1
Embedded Function 1 Control OUT	0	2
Embedded Function 1 Control IN	0	3
Embedded Function 1 Generic IN	1	4
Embedded Function 1 Generic OUT	1	5
Embedded Function 6 Control OUT	0	10
Embedded Function 6 Control IN	0	11
Embedded Function 6 Generic OUT	1	6
Embedded Function 6 Generic IN	1	7
Embedded Function 7 Control OUT	0	12

Embedded Function 7 Control IN	0	13
Embedded Function 7 Generic OUT	1	8
Embedded Function 7 Generic IN	1	9

Controlling the H11A (Appendix to PDIUSBH11A Application Notes)

The firmware and software used on the H11A is the same as that used on the H11 except for the firmware required to control the additional ports as noted above.

The H11A/H11 utilizes the I^2C interface to communicate to the external micro-controller. This I^2C port can run at a maximum speed of 100Kbps for the H11 and up to1Mbps for the H11A. The use of the I^2C allows a slow device to communicate with the H11A/H11 as well. Hence, any micro-controller can emulate I^2C by software through any port. There is no necessity to use a micro-controller with hardware I^2C.

The H11A/H11 acts as a slave device on the application board. The micro-controller will always be the master to initiate all data transfer between them. There are two 7-bit I^2C slave addresses residing on the H11A/H11 that can be used to program the H11A/H11.

They are listed in the following table.

Purpose	**I^2C Address**	**Read/Write**
Command	0011011B (0x1B)	Write only
Data	0011010B (0x1A)	Read and Write

Commands are used to instruct the H11A/H11 to perform the functions documented in the H11A/H11 specifications. The following table lists some of the commands and the corresponding command code.

Command Name	**Recipient**	**Command Code**	**Data Phase**
Set Address/Enable	Hub	0xD0	Write 1 byte
Set Endpoint Enable	Hub + Embedded Functions	0xD8	Write 1 byte
Select Endpoint	Hub Control OUT	0x00	Read 1 byte
Read Endpoint Status	Hub Control IN	0x81	Read 1 byte

This is not an exhaustive command table, it is meant to show some examples of the command set only

Programming the H11A/H11 consist of two steps.
Command Phase - Write the command code to the Command I^2C Slave.
Data Phase - Write or read data from the Data I^2C Slave Address.

PDIUSBH11A Application Notes

Writing to the H11A/H11
For example, to set the address of the Hub to 0x05 and to enable the Hub, we use command code 0xD0 (Figure 11). The data to be written has the format below:

Bit #	7	6	5	4	3	2	1	0
Value	1	0	0	0	0	1	0	1

7-bit Address

Enable

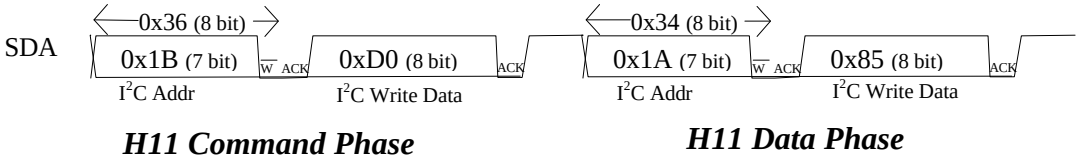


Figure 11: Programming the H11, in a command+write operation

Reading from the H11A/H11

As an example, to read endpoint status, we use command code 0x81 (Figure 12).

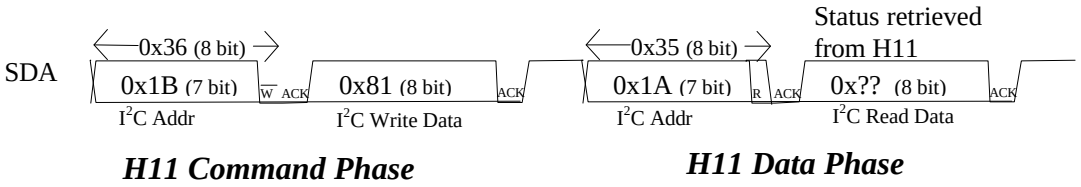


Figure 12: Programming the H11, in a command+read operation

Indexing of Endpoints

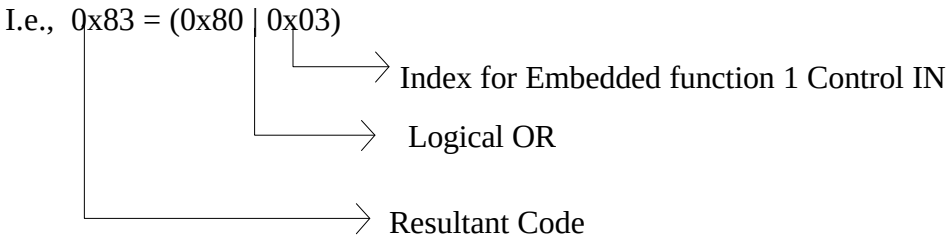
The endpoints of the H11A are indexed as follows:

Single Embedded Function Mode

Endpoint	Endpoint Number	Index

Hub Control OUT	0	0
Hub Control IN	0	1
Embedded Function 1 Control OUT	0	2
Embedded Function 1 Control IN	0	3
Embedded Function 1 Generic IN	1	4
Embedded Function 1 Generic OUT	1	5
Embedded Function 1 Generic OUT	2	6
Embedded Function 1 Generic IN	2	7
Embedded Function 1 Generic OUT	3	8
Embedded Function 1 Generic IN	3	9

Indexing is used to ease the programming effort on the micro-controller. For instance, to read the endpoint status, the command code is 0x80 for the Hub Control OUT. To read the endpoint status of Embedded Function 1 Control IN, the command code is 0x83 or (base Command | index).



Reading the OUT buffer of the H11A

Data transferred from the host via the USB bus is stored in a buffer on the H11A/H11. Each endpoint OUT/IN has its own buffer either as a temporary storage for data sent from the host (in the case of an OUT token), or as a holding place for data to be sent to the host while waiting for the IN token.

As these buffers share the same entry point on the I^2C, a buffer must be selected before data can be transferred. This is done using the Select Endpoint command with the base command code 0x00.

The steps in reading from the buffer area of the H11A/H11 are as follows:

(The following is a read buffer command from the Hub control endpoint OUT)

Select Endpoint to read from.

I^2C traffic
Write to slave (H11A): Address(0x36) Data(0x00)

Read from slave (H11A): Address(0x35) Data(0x??)

The data read stores the status of the endpoint as defined in the H11A specifications.

Read (N+1) bytes from Buffer.

I²C traffic
Write to slave (H11A): Address(0x36) Data(0xF0)
Read from slave (H11A): Address(0x35) Data0(0x??) …. DataN(0x??)

Data0 : 0x00 - reserved.
Data1 : 0xYY - Length of the actual data received from Host
Data2 : 0x?? - First byte of USB data
Data3 : 0x?? - Second byte of USB data
…and so on.

Clear Buffer

I²C traffic
Write to slave (H11A): Address(0x36) Data(0xF2)

Writing to the IN buffer of the H11A

To send data to the host via the USB bus, the host must initiate the transfer. The responsibility of the micro-controller is to write the data to the IN buffer of the target endpoint on the H11A. The data will remain in the buffer until the H11A receives the IN token from the host. When processing of the IN token completes, the H11A will send the data in the buffer to the host. The H11A will be interrupted when the data has been successfully transferred.

The steps are similar to the read operation.

(The following is a write buffer command to the Hub control endpoint IN)

Select Endpoint to write to.

I²C traffic
Write to slave (H11A): Address(0x36) Data(0x01)
Read from slave (H11A): Address(0x35) Data(0x??)

Write (N+1) bytes to Buffer.

I²C traffic
Write to slave (H11A): Address(0x36) Data(0xF0)
Write to slave (H11A) : Address(0x34) Data0(0x??) …. DataN(0x??)

Data0 : 0x00 - reserved.
Data1 : 0xYY - Length of the actual data for Host
Data2 : 0x?? - First byte of USB data
Data3 : 0x?? - second byte of USB data
…and so on.

A total of N+1 bytes are transferred to the H11A, but the actual data to be sent starts only from Data2. Hence, the total number sent is N-1.

Data0 and Data1 allow the H11A to track the actual number of bytes to be sent.

Validate Buffer.

I²C traffic
Write to slave: Address(0x36) Data(0xFA)

This command tells the H11A that the buffer has been filled with data ready to be transferred out.
Setting and Clearing Hub Features

The control of the downstream ports is done by software. There are 8 settings that can be cleared or set. They are listed in the table below.

FEATURE	FEATURE CODE	SET	CLEAR
F_PORT_ENABLE	0	Enables a port	Disables a port
F_PORT_SUSPEND	1	Suspends a port	Resumes a port
FC_PORT_RESET	2	Resets a port	Clears a port reset change bit
F_PORT_POWER	3	Powers all ports	Unpowers all ports
C_PORT_CONNECTION	4	-	Clears a port connection change bit
C_PORT_ENABLE	5	-	Clears a port enable

			change bit
C_PORT_SUSPEND	6	-	Clears a port suspend change bit
C_PORT_OVERCURRENT	7	-	Clears a port (Mode 1) or hub (Mode 0) over-current change bit

The command code to Set or Clear is indexed; similar to how the endpoints are indexed. This is shown below.

To Clear Feature:

Command code	**Destination Port**
0xE0	1
0xE1	2
0xE2	3
0xE3	4

To Set Feature:

Command code	**Destination Port**
0xE8	1
0xE9	2
0xEA	3
0xEB	4

The following shows the commands for powering downstream port number 3.

Set Feature with Feature code 3

I^2C traffic
Write to slave (H11A): Address(0x36) Data(0xEA)
Write to slave (H11A): Address(0x34) Data(0x03)

To suspend downstream port 2, the commands are:

Clear Feature port 2 with Feature code 1

I^2C traffic
Write to slave (H11A): Address(0x36) Data(0xE1)
Write to slave (H11A): Address(0x34) Data(0x01)

Enumeration of the Hub

The enumeration process of the Hub consists of many USB standard requests. All standard requests or class requests come with a pre-packet consisting of a Setup Token. The next packet contains the relevant data packet that translates to either a standard request or a class request. The format of such a request is shown in Figure 13.

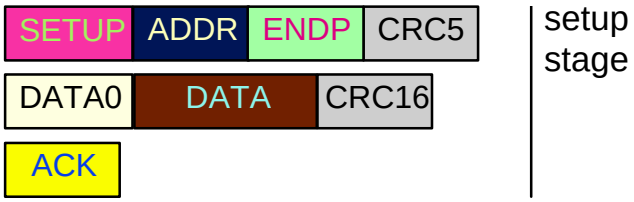


Figure 13: A USB request

The data transported from the host is captured by the H11A and stored in the OUT buffer. The destination buffer depends on matching the packet's address with the current address settings of either the Hub or the embedded function.

To process this request, the buffer must first be read (as explained in the earlier section). Depending on the request, the firmware can then either send a zero packet or packets of data to the IN endpoint.

Enumeration of the hub requires stepping through the following requests (as in Memphis 1434):

Get Device Descriptor
Set Address
Get Device Descriptor
Get Configuration Descriptor
Set Configuration
Get Hub Descriptor
Get Device Status
Set Feature, Port 1 .. Last Port (in sequence)
Get Port Status, Port 1 .. Last Port (in sequence)

Interpretation of the USB requests will not be shown here. Please refer to the USB specifications, Chapter 9 and Chapter 11.

USB requests receiving and processing on the H11A/H11

When the H11A/H11 intercepts a setup token, the data on the next packet is taken and stored in the respective OUT buffer.

An interrupt signal is generated on the INT_N pin of the H11A/H11. This interrupt pin remains low until a read last transaction command is given to the H11A/H11.

The endpoint status should be read to ensure that the current packet is a setup packet. If so, an Acknowledge Setup command needs to be given to unlock the buffers, both the IN buffer as well as the OUT buffer. Data can only be written into the buffer after it has been unlocked.

Finally, the OUT buffer will be read and interpreted.

This process is highlighted below assuming that a setup packet on the Hub control OUT endpoint has been received by the H11A/H11.

Condition	I^2C traffic	Purpose
INT_N low	Write to slave (H11A) Address(0x36) Data(0xF4)	To check interrupt source.
	Read from slave (H11A) Address(0x35) Data(0x??)	
Ctrl OUT is source of interrupt	Write to slave (H11A) Address(0x36) Data(0x40)	To reset interrupt flag by reading last transaction status.
	Read from slave (H11A) Address(0x35) Data(0x??)	
	Write to slave (H11A) Address(0x36) Data(0x80)	To check current endpoint status.
	Read from slave (H11A) Address(0x35) Data(0x??)	
it is a setup packet	Write to slave (H11A) Address(0x36) Data(0x01)	To Select Hub IN endpoint. The purpose is to acknowledge setup at Hub IN.
	Read from slave (H11A) Address(0x35) Data(0x??)	
	Write to slave (H11A) Address(0x36) Data(0xF1)	Acknowledge Setup of selected endpoint
	Write to slave (H11A) Address(0x36) Data(0x00)	To Select Hub OUT endpoint. The purpose is to acknowledge setup at Hub OUT.
	Read from slave (H11A) Address(0x35) Data(0x??)	
	Write to slave (H11A) Address(0x36) Data(0xF1)	Acknowledge Setup of selected endpoint
	Write to slave (H11A) Address(0x36) Data(0xF0)	To read 10 bytes from the buffer.
	Read from slave (H11A) Address(0x35) Data0(0x??) Data1(0x??) … Data9(0x??)	
	Write to slave (H11A) Address(0x36) Data(0xF2)	Clear Buffer of selected endpoint
With the request data, the firmware can now process the requests.		