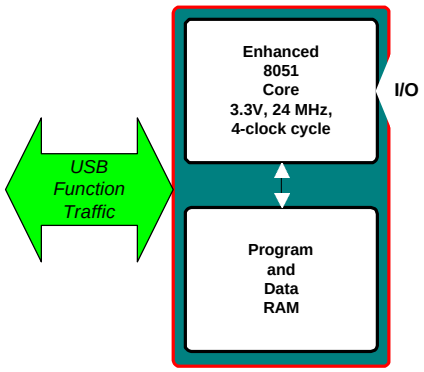
Soft USB Controller Design Challenges

Lane Hauck
Systems Engineering Manager
Anchor Chips Inc.

Implementing peripheral solutions for an emerging bus standard such as USB can be challenging. Peripheral developers must be responsive to external, co-developed events. For USB, these include functionality with various host platforms (using different chip sets and BIOS), various releases of the host's operating system, and a continual evolution of peripheral device classes. A good way to accommodate USB development changes is to created USB controller chips that operate "soft", that is from code downloaded from the host computer into on-chip RAM, rather than using the traditional ROM approach.

On the surface it may seem an easy matter to download code over the USB and then execute the code to function as a USB peripheral device. But the USB Specification allows a device to enumerate only once, so there is an inherent conflict between the device that downloads code and the resultant device that executes the custom application. This paper describes the problem in detail and outlines the novel solution embodied in the EZ-USB chip family.

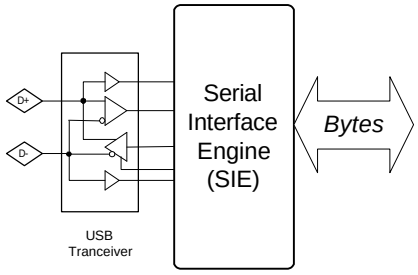
The Objective: *SOFT*



Here's the objective—a soft, single-chip USB peripheral solution. One side of the device receives and sends USB traffic, while the other side interfaces to the device's peripheral circuitry. Program code and data are stored in volatile RAM, which is downloaded from the host via the USB channel.

When the chip powers on, there is no code in RAM and the CPU is held in reset. To understand the requirements of a "soft" architecture, it is helpful to review the anatomy of a USB peripheral device from the inside out. Then the special measures required to implement the soft feature will be apparent.

The Basic USB Interface

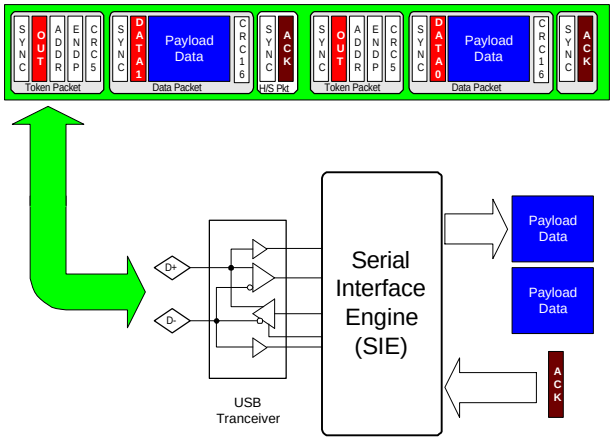


Inside every USB peripheral is a Serial Interface Engine (SIE). The SIE:

- Serializes and de-serializes USB data.
- Decodes the NRZI format used by USB.
- Transfers bytes to and from the device.
- Handles bit stuffing.
- Checks the USB data for validity using CRC fields.
- Handles bus signaling like reset, suspend and resume.
- Re-tries certain USB transfers if errors are encountered.

The SIE is roughly analogous to the UART chip connected to a serial port. Serial data enters and leaves the SIE, and parallel bytes are delivered to, and accepted from, the peripheral. However, USB is much more complex than a serial port. The two examples that follow illustrate some of the added complexity.

What the SIE Does



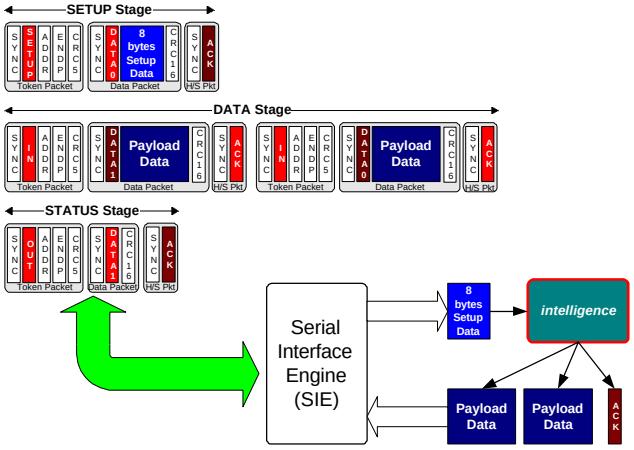
Here's a simple example of what the SIE does. USB traffic is shown at the top, with time traveling from left to right. This transaction represents a USB Bulk data transfer.

A USB transaction consists of data packets that are identified by special codes called Packet ID's (PIDS). The bulk transfer shown above uses four PID types: OUT, DATA0, DATA1, and ACK.

The first packet is an OUT token, announcing that the host is about to send data to the peripheral (USB direction is host-centric, OUT means host to device). The second packet contains the DATA1 PID followed by a block of bytes labeled "Payload Data". The device indicates successful receipt of the data by sending the ACK PID in the third, handshake packet. The host then sends another OUT token, this time using the DATA0 PID, followed by more data. Finally, the devices sends another ACK to conclude the transfer.

The two data PIDS, DATA0 and DATA1, provide added data security beyond CRC checking to guard against corrupted handshake packets, and to maintain synchronism throughout long bulk transfers. Bulk data is transferred using alternating DATA0/1 PIDS. The host and peripheral maintain "data toggle" bits that are complemented when data is successfully sent and acknowledged. If either side fails to read a correct handshake, it does *not* flip its data toggle, causing a mis-match with the next data PID. This initiates a re-try. All of this is handled automatically by the SIE.

A USB Control Transfer



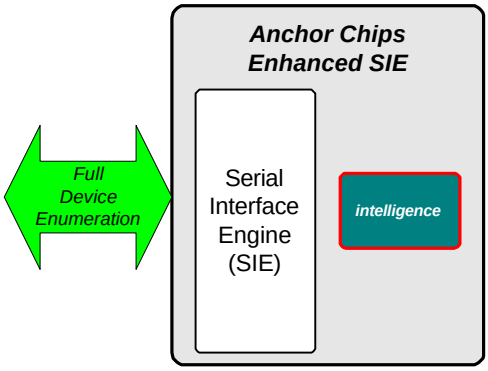
Here's a more complex SIE illustration, which demonstrates processing performed by the USB protocol layer. The protocol layer uses the SIE to respond to standard USB requests. The protocol layer can be implemented in logic or with the aid of a CPU. This figure shows a protocol layer transaction called a CONTROL transfer.

CONTROL transfers consist of two stages, SETUP and STATUS, and an optional third data stage. This example uses a data stage. The "Intelligence" block first decodes the host request using the eight Setup Data bytes delivered by the SIE. In this example, the host has requested data from the peripheral. The "Intelligence" block decodes the request from the eight SETUP bytes, retrieves the requested data from internal memory, constructs packets of the proper size, and sends them back through the SIE for USB transmission. After the data has been transferred, the "Intelligence" block commands the SIE to ACK the STATUS phase to conclude the control transfer.

When a device is first attached to USB, it answers a series of host requests during a process called "enumeration". During enumeration the device tells the host about its capabilities and requirements. The CONTROL transfer shown above is typical of the USB traffic during enumeration.

In a soft controller, the program RAM powers on in an unknown state, so the on-chip CPU is not available to perform the "Intelligence" function described above. Therefore, the SIE must be enhanced to handle enumeration without using the CPU.

EZ-USB Enhanced SIE

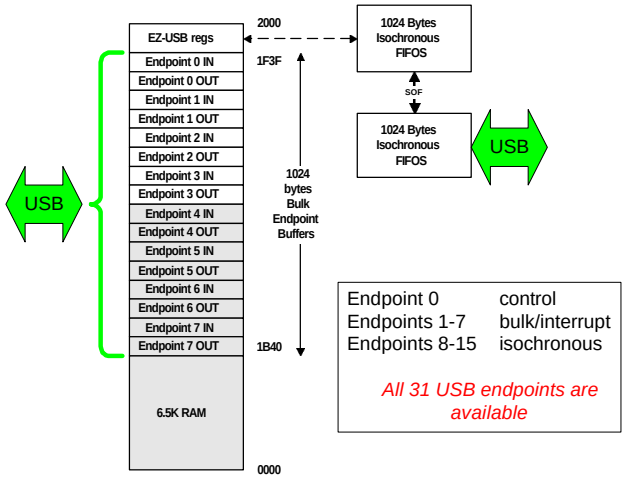


The intelligence to fully enumerate as a USB device is incorporated into the SIE logic. This "Enhanced SIE" contains hard-coded descriptor tables to identify it as an "Anchor Generic" device. These descriptors instruct the operating system to load the correct driver to operate the device. The Anchor Generic device contains the following sets of USB endpoints:

Endpoint	**Type**	**Alternate Setting**		
		0	**1**	**2**
		Max Packet Size (bytes)		
0	CTL	64	64	64
1 IN	INT	0	16	64
2 IN	BULK	0	64	64
2 OUT	BULK	0	64	64
4 IN	BULK	0	64	64
4 OUT	BULK	0	64	64
6 IN	BULK	0	64	64
6 OUT	BULK	0	64	64
8 IN	ISO	0	16	256
8 OUT	ISO	0	16	256
9 IN	ISO	0	16	16
9 OUT	ISO	0	16	16
10 IN	ISO	0	16	16
10 OUT	ISO	0	16	16

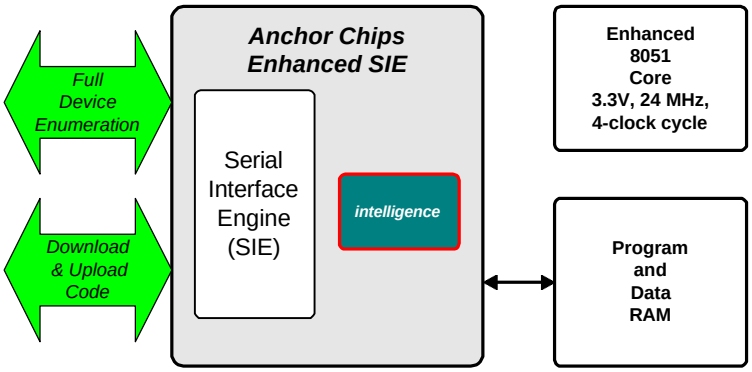
Having a default set of endpoints simplifies the USB learning curve, since USB transfers can be programmed and studied starting with a fully functional USB device.

AN2131Q Memory Map



The preceding table of default endpoints is actually a subset of the 31 endpoints available in the AN2131Q. The CPU firmware reports the desired endpoints to the host via descriptor tables, and enables the desired endpoints.

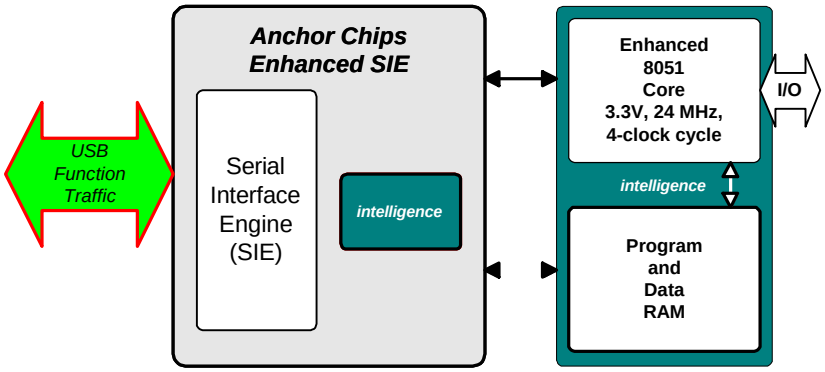
Advanced SIE Enumerates & Loads Code



It's not enough just to enumerate. The Enhanced SIE must also download code into on-chip RAM for operation as the final USB device. The Enhanced SIE accomplishes this by decoding a vendor-specific request that downloads code into internal RAM. This request is handled over endpoint zero, the default CONTROL endpoint. The eight setup bytes that define the "Anchor Download" are shown below:

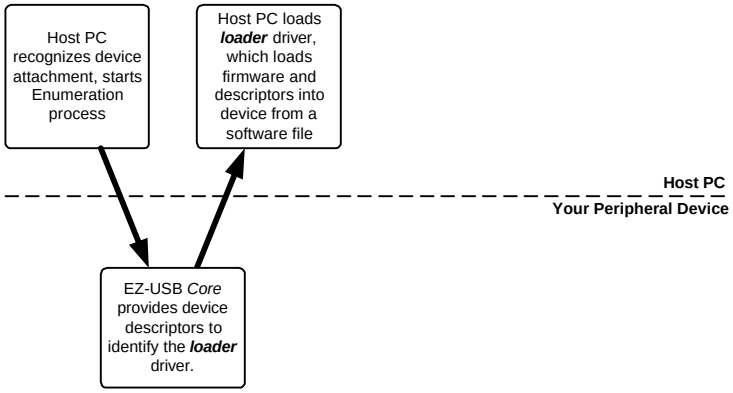
Byte	Field	Value	Meaning
0	bmRequest	**0x40**	Vendor Request, OUT
1	bRequest	**0xA0**	"Anchor Load"
2	wValueL	**AddrL**	Starting address
3	wValueH	**AddrH**	
4	wIndexL	0x00	
5	wIndexH	0x00	
6	wLengthL	**LenL**	Number of Bytes
7	wLengthH	**LenH**	

Final USB Device



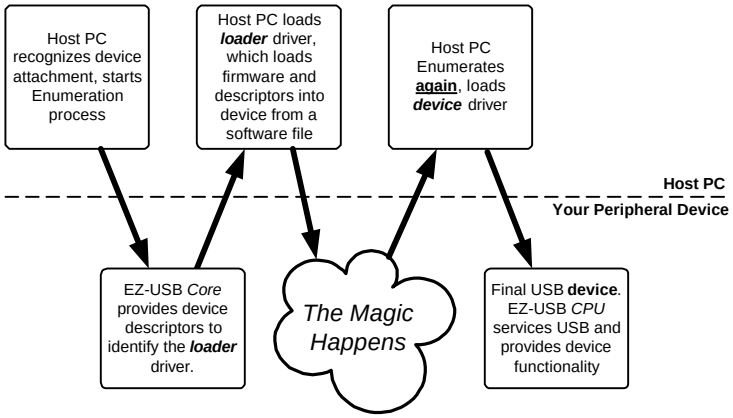
Once the code is loaded and the CPU is brought out of reset, the final USB device is operational. Now the CPU is in charge. The CPU handles the USB device requests that were initially fielded by the enhanced SIE. Because the CPU has access to the added SIE intelligence, the firmware is simplified. In effect, the enhanced SIE becomes a high level engine for USB requests.

The ReNumeration™ Process

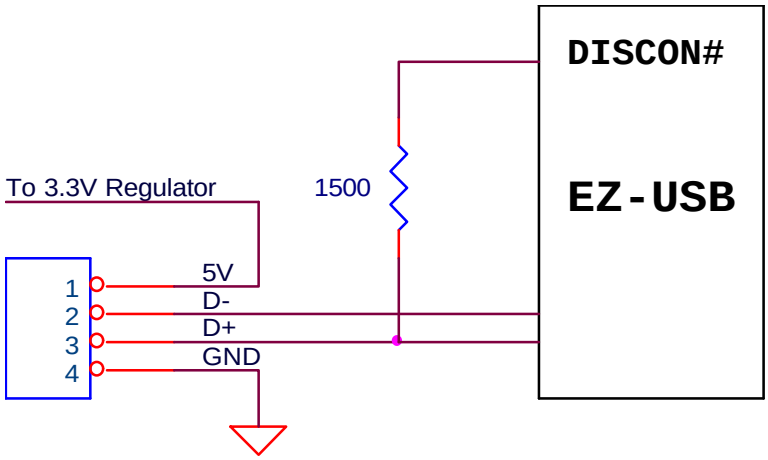


There's a hitch. USB allows a device to enumerate only *once*. The three steps shown above accomplish the enumeration that configures the soft USB controller as a loader, capable of downloading the final device personality into internal RAM. But once the RAM is loaded with the descriptors and code that define the final device, it's too late to connect to USB as the final device.

The ReNumeration™ Process



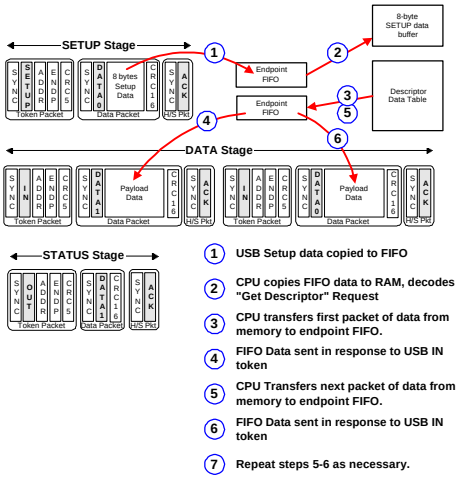
The device needs to enumerate a second time, or ReNumerate™. Then the final device driver is loaded, the device contains all firmware and descriptors, and our soft controller is in business.



The “magic” is simple. A USB hub detects a high-speed device by the presence of a 1500 ohm pullup resistor connected to the D+ line. (The hub has a 15 Kohm pulldown which keeps the line low when nothing is connected.)

The DISCON# pin either drives to the 3.3V rail or floats, under control of a CPU register bit. This emulates a physical disconnect and reconnect while maintaining power to the device. Once re-connected, the USB device enumerates using the downloaded code and descriptors. The entire enumerate-ReNumerate™ process happens in about one second.

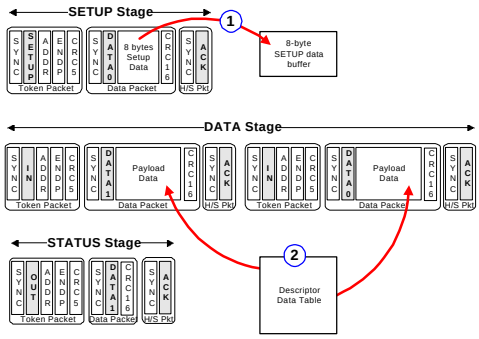
Get Descriptor-Conventional Method



Once ReNumerated, the CPU can take advantage of the enhanced SIE to simplify the firmware needed to service USB device requests. The figure illustrates a typical device request called “Get_Descriptor”.

Most USB peripheral chips handle the “Get_Descriptor” in the manner shown. The CPU transfers the eight setup bytes from an endpoint FIFO to RAM to decode the request (1,2). Then the CPU fetches the requested data from internal RAM, packetizes it, and loads it into an endpoint FIFO for USB transmission (3-6). The CPU must keep track of the three stages of the control transfer, usually by maintaining a firmware state machine.

Get Descriptor-Enhanced SIE Method

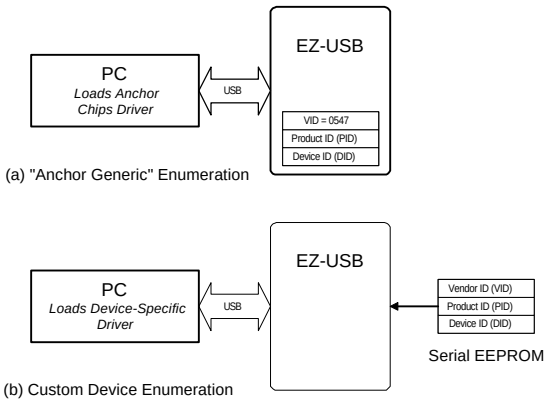


1. EZ-USB core copies Setup data directly to RAM, eliminating the FIFO-to-RAM copy step. 8051 decodes the "Get Descriptor" request.
2. 8051 sets pointer to descriptor table in RAM, EZ-USB core does entire multi-packet transfer.

Because the enhanced SIE contains logic to handle the Get_Descriptor request, the CPU can take advantage of this added hardware to respond to its own requests.

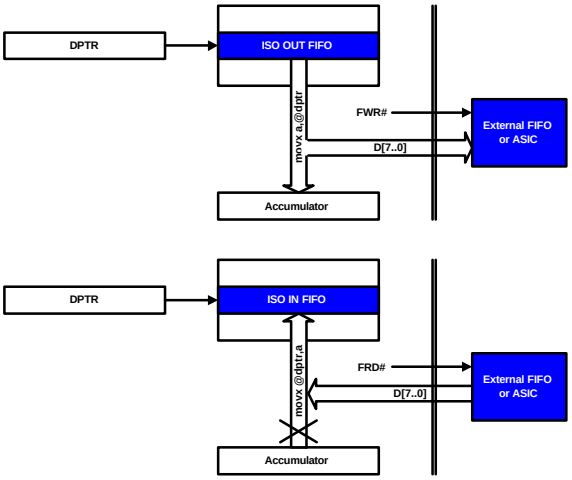
As in the conventional method, the CPU decodes the request, although it accesses the eight setup bytes directly in memory, saving the FIFO-to-memory transfer. Then the CPU simply loads the address of the requested descriptor table into a control register. The Enhanced SIE does the rest.

Watch Those VID-PID-DIDs



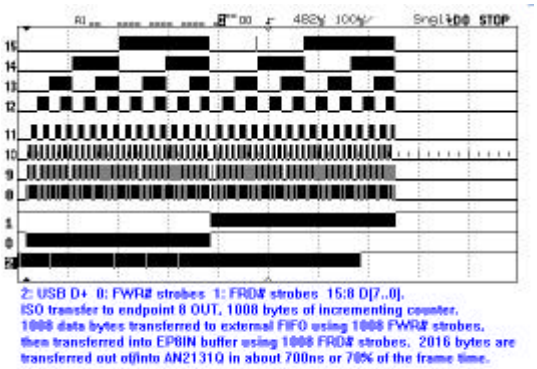
In Anchor Generic mode, six bytes of descriptor information “tag” the device to an OS driver. To allow different vendors to customize their own drivers, a small (16 byte) EEPROM attaches to the EZ-USB chip to provide custom VID-PID-DID information. This allows the device manufacturer to write a single driver which incorporates the loader.

AN2131Q Fast Transfer Modes



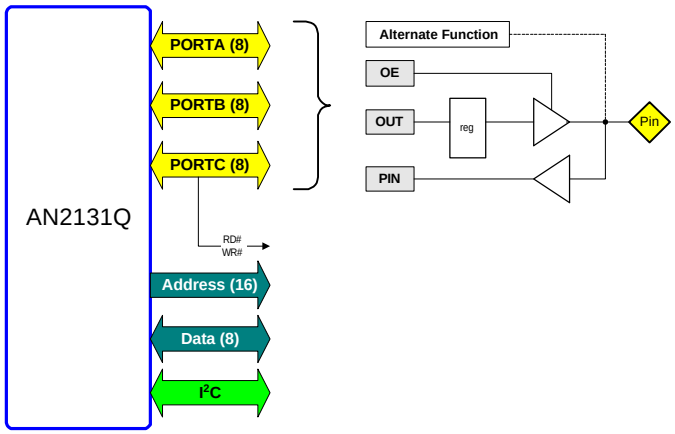
It's important to insure that the CPU keeps up with USB rates when it transfers data to and from the peripheral (for example an external FIFO). A "turbo" mode monitors transfers between the 8051 accumulator and endpoint FIFOS and buffers. When enabled, USB data is transferred directly to the AN2131Q data bus, and fast strobes FRD# and FWR# are generated.

Fast Transfers to an External FIFO



Here is an isochronous transfer of 1008 bytes, transferred from an OUT endpoint to an external FIFO, and then looped back to an IN endpoint.

Expanding the AN2131Q



The AN2131Q has a non-multiplexed address bus, an 8-bit data bus, and three 8-bit IO ports. Each IO pin has an alternate function, for example the Fast Read (FRD#) and Fast Write (FWR#) strobes shown in the previous figure.