Soft USB Controller Design Challenges

Implementing peripheral solutions for an emerging bus standard such as USB is challenging. Peripheral developers must be responsive to external, co-developed variables. For USB, these variables 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 create 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 seems an easy matter to download code over the USB and then execute the code to function as a USB peripheral device. However, 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 is 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 (*Figure 1*).

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. The special measures required to implement the soft feature are then apparent.

The Basic USB Interface

Inside every USB peripheral is a Serial Interface Engine (SIE) (*Figure 2*). The SIE:

- Serializes and deserializes 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 following two examples illustrate some added complexity.

What the SIE Does

Figure 3 provides a simple example of what the SIE does. USB traffic is shown at the top of the illustration, with time traveling from left to right. This USB transaction represents a USB Bulk data transfer.

A USB transaction consists of data packets identified by special codes called Packet IDs (PIDS). The bulk transfer 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 and the device's ACK.

The two data PIDS, DATA0 and DATA1, provide data security beyond CRC checking to guard against corrupted handshakes, 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 ac-

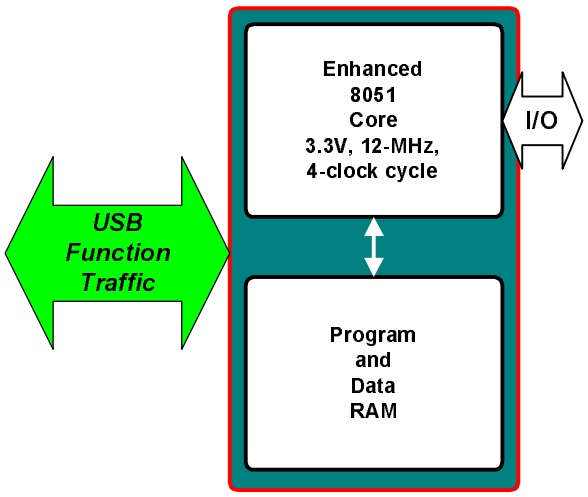

Figure 1. The Objective: SOFT

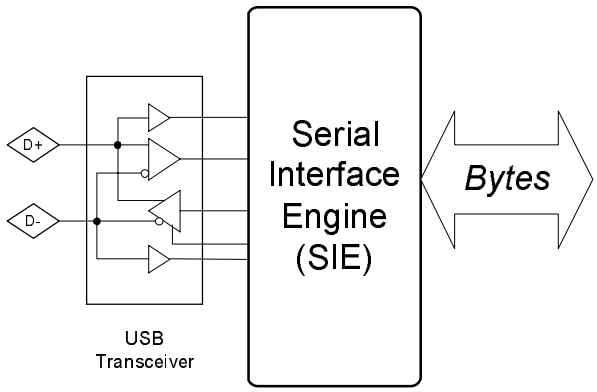

Figure 2. The Basic USB Interface

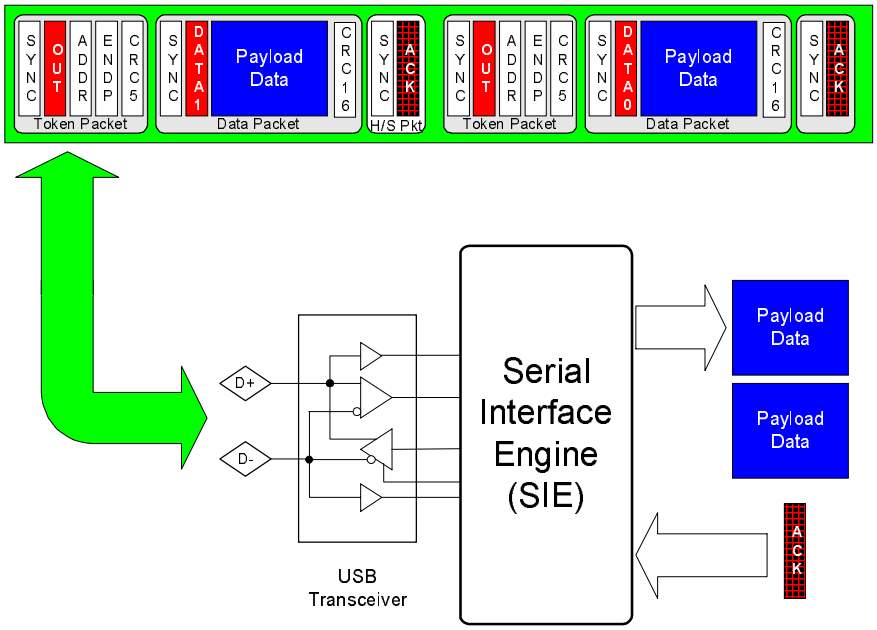

Figure 3. Example 1, A USB Bulk Transfer

knowledged. If either side fails to read a correct handshake, it does *not* flip its data toggle, causing a mismatch with the next data PID. This initiates a retry. All of this is handled automatically by the SIE.

A USB Control Transfer

Figure 4 illustrates a more complex USB operation: the SIE helps to process USB protocol information. The protocol layer responds to standard USB requests. The protocol layer can be implemented in logic or with the aid of a CPU. *Figure 4* shows a USB transaction called a CONTROL transfer.

CONTROL transfers consist of two or three stages, SETUP, STATUS, and an optional DATA stage. This example uses a DATA stage. The “Intelligence” block first decodes the host request using the eight Setup Data bytes from the SIE. In this example, the host has requested data from the peripheral (such as a “Get_Descriptor” request). 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.

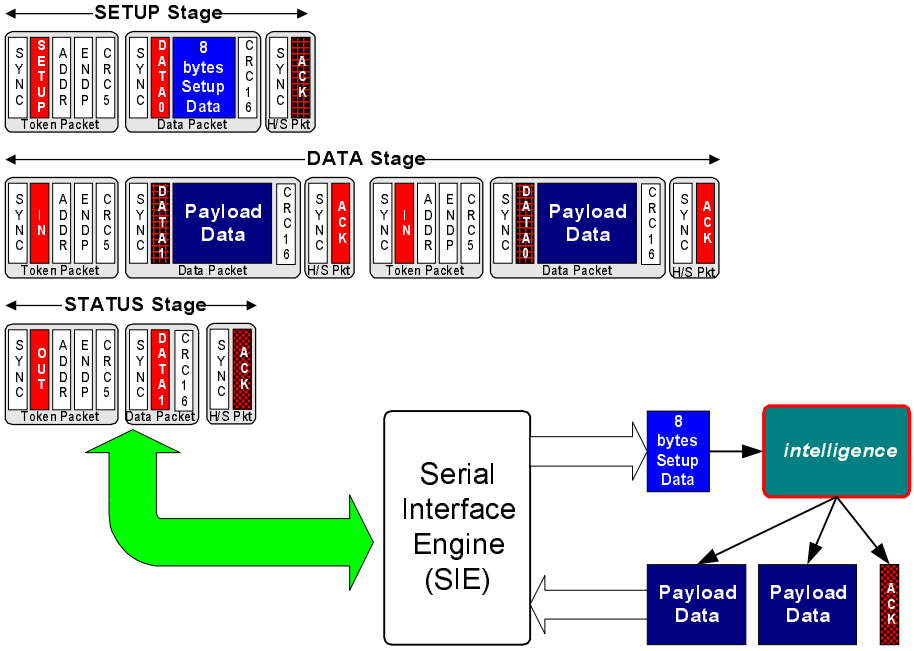

Figure 4. Example 2, A USB CONTROL Transfer

When first attached to USB, a device answers a series of host requests through a process called “enumeration”. During enumeration, the device tells the host about its capabilities and requirements. The CONTROL transfer shown in *Figure 4* is typical of the USB traffic during enumeration.

In a soft controller, the RAM, which holds the program code, 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 can be incorporated into the SIE logic (*Figure 5*). This “Enhanced SIE” contains hard-coded descriptor tables to identify it as a “Generic” device. These descriptors instruct the operating system to load the correct driver to operate the device. The generic device contains default USB endpoints and alternate settings as shown in *Table 1*.

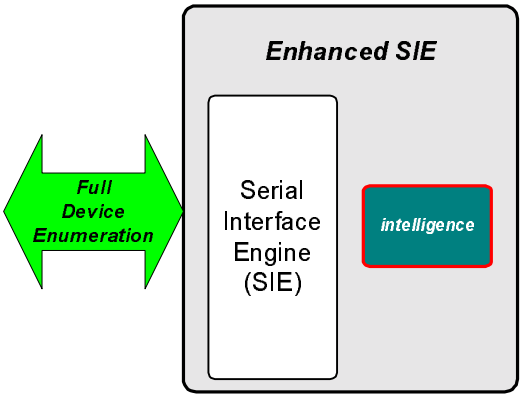

Figure 5. EZ-USB Enhanced SIE

Table 1. Default 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 the developer can program and study USB transfers starting with a fully functional USB device.

AN2131 Memory Map

The default endpoints shown in *Table 1* actually represent a subset of the 31 endpoints available in the AN2131. The full set of AN2131 endpoints is shown in *Figure 6*.

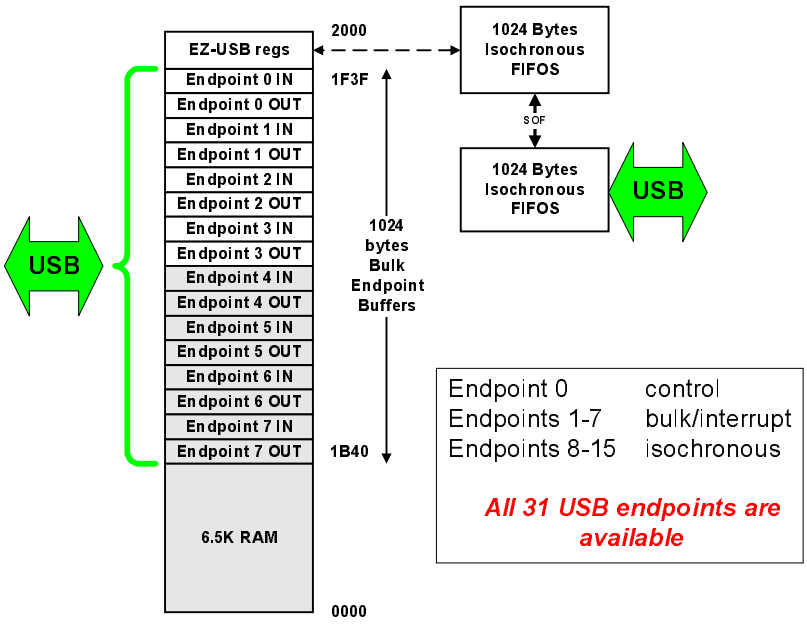

Figure 6. AN2131 Memory Map

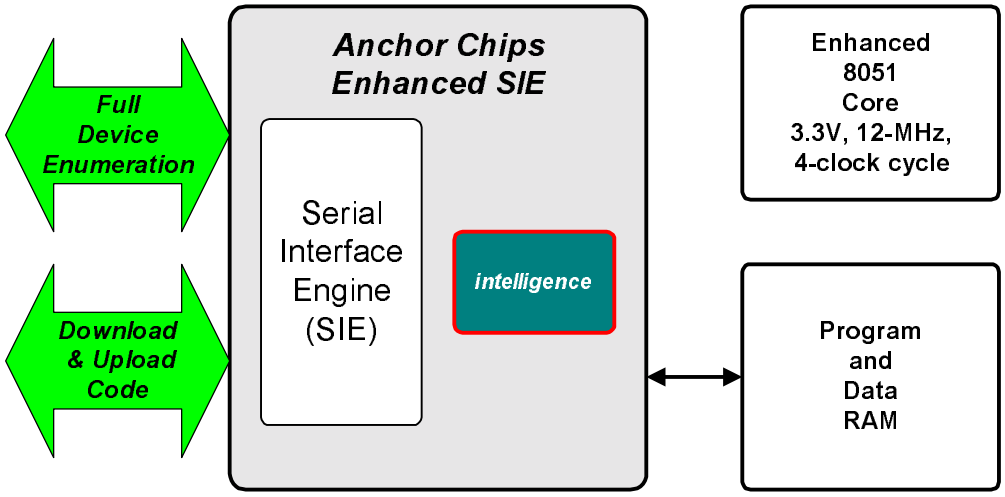

Figure 7. Advanced SIE Enumerates and Loads Code

Advanced SIE Enumerates and Loads Code

For the soft application, 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 (*Figure 7*). 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 set-up bytes that define the "Download" USB request are shown below:

Byte	Field	Value	Meaning
0	bmRequest	0x40	Vendor Request, OUT
1	bRequest	0xA0	"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 (*Figure 8*).

The ReNumeration™ Process

There's a hitch. USB allows a device to enumerate only *once*. The three steps shown in *Figure 9* 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 device needs to enumerate a second time, or ReNumerate™ (*Figure 10*). When the final device driver loads, the device contains all firmware and descriptors, and our soft controller is in business.

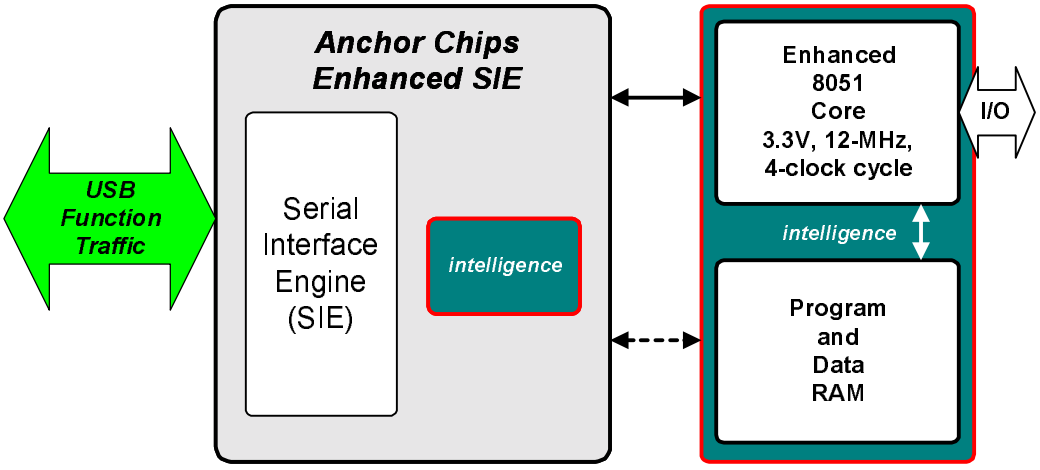

Figure 8. Final USB Device

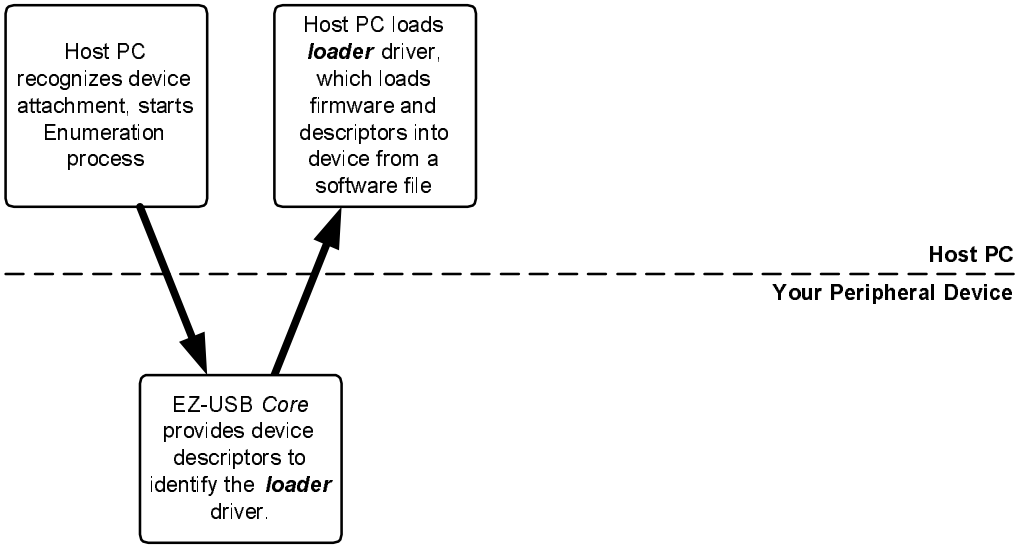

Figure 9. The Enumeration Process

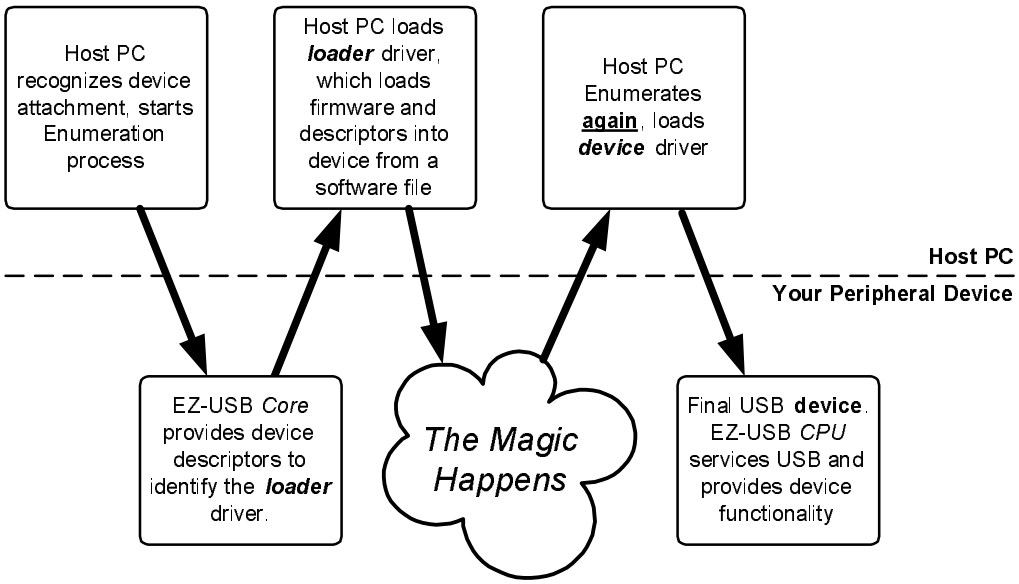

Figure 10. The ReNumeration Process

The "magic" is simple (*Figure 11*). A USB hub detects a high-speed device by the presence of a 1500 Ω pull-up resistor connected to the D+ line. (The hub has a 15-kΩ pull-down to keep the line low when nothing is connected.)

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

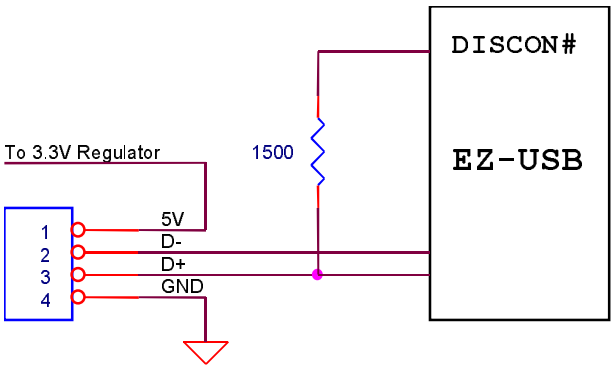

Figure 11. The "Magic" is Simple

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. *Figure 12* illustrates a typical device request called “Get_Descriptor”.

Most USB peripheral chips handle the “Get_Descriptor” in the manner shown in *Figure 12*. 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

Because the enhanced SIE already 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 (from its internal descriptor tables) into a control register. The Enhanced SIE does the rest (*Figure 13*). Watch Those VID-PID-DIDs.

There are some details to keep straight. In 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 (see *Figure 14*).

AN2131 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). An AN2131 Fast Transfer mode monitors transfers between the 8051 accumulator and endpoint FIFOS and buffers. When enabled, USB data is transferred directly to the AN2131 data bus, and fast strobes FRD# and FWR# are generated (*Figure 15*).

Fast Transfers to an External FIFO

Figure 16 shows an isochronous transfer of 1008 bytes, transferred from an OUT endpoint to an external FIFO, and then looped back to an IN endpoint. Thanks to the fast transfer mode, the time to transfer 1008 bytes in or out of the chip is shorter than the time for the 1008 bytes to arrive or be sent over the USB.

Expanding the AN2131Q

The AN2131Q has a non-multiplexed address bus, an 8-bit data bus, and three 8-bit IO ports (*Figure 17*). Each IO pin has an alternate function, for example the Fast Read (FRD#) and Fast Write (FWR#) strobes shown in *Figure 16*.

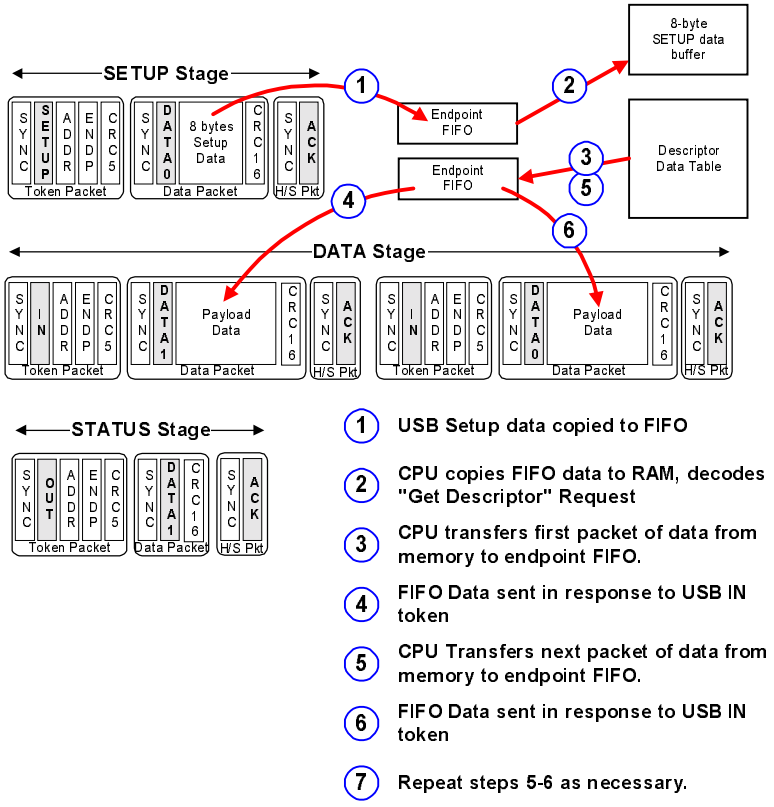

Figure 12. Get_Descriptor—Conventional Method

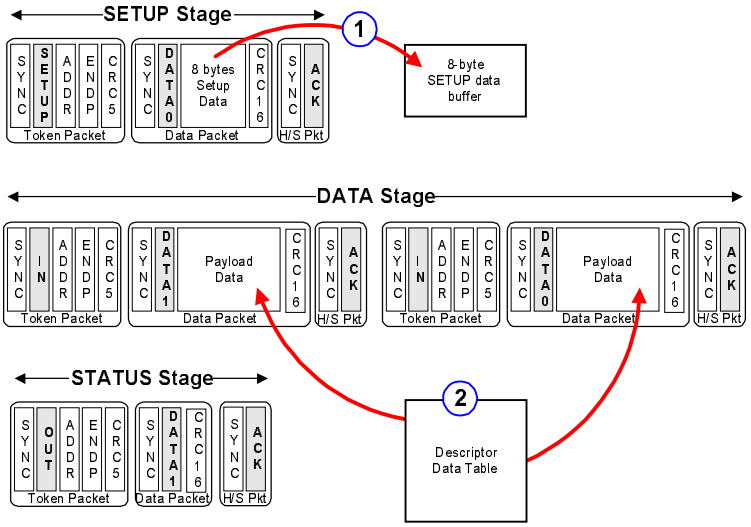

Figure 13. Get_Descriptor—Enhanced SIE Method

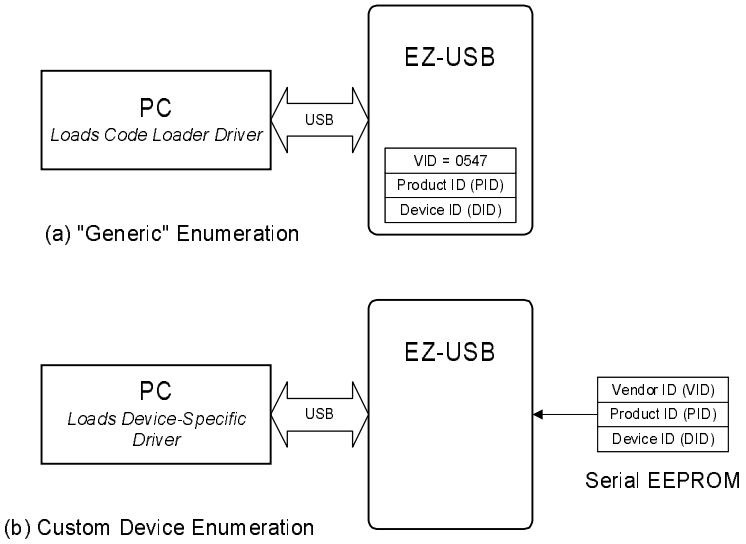

Figure 14. Watch Those VID-PID-DIDs

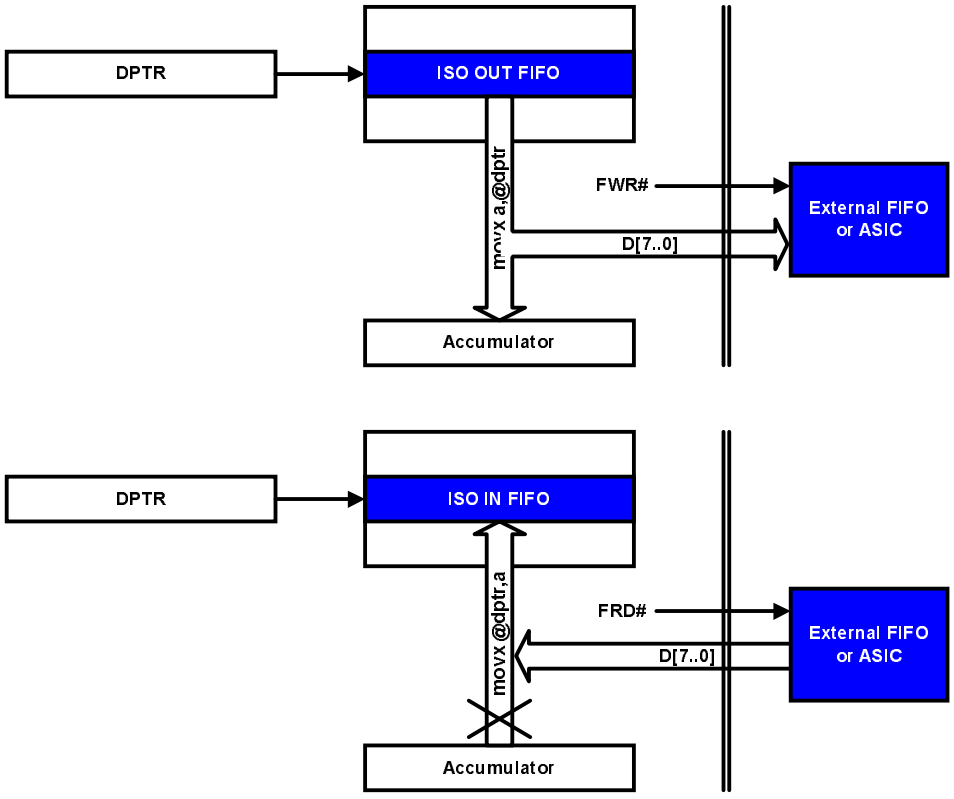

Figure 15. AN2131 Fast Transfer Modes

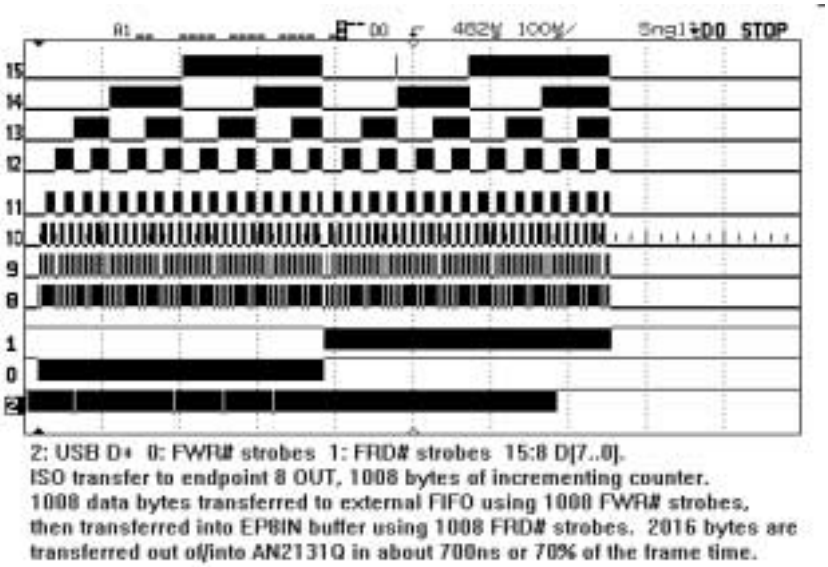

Figure 16. Fast Transfer to an External FIFO

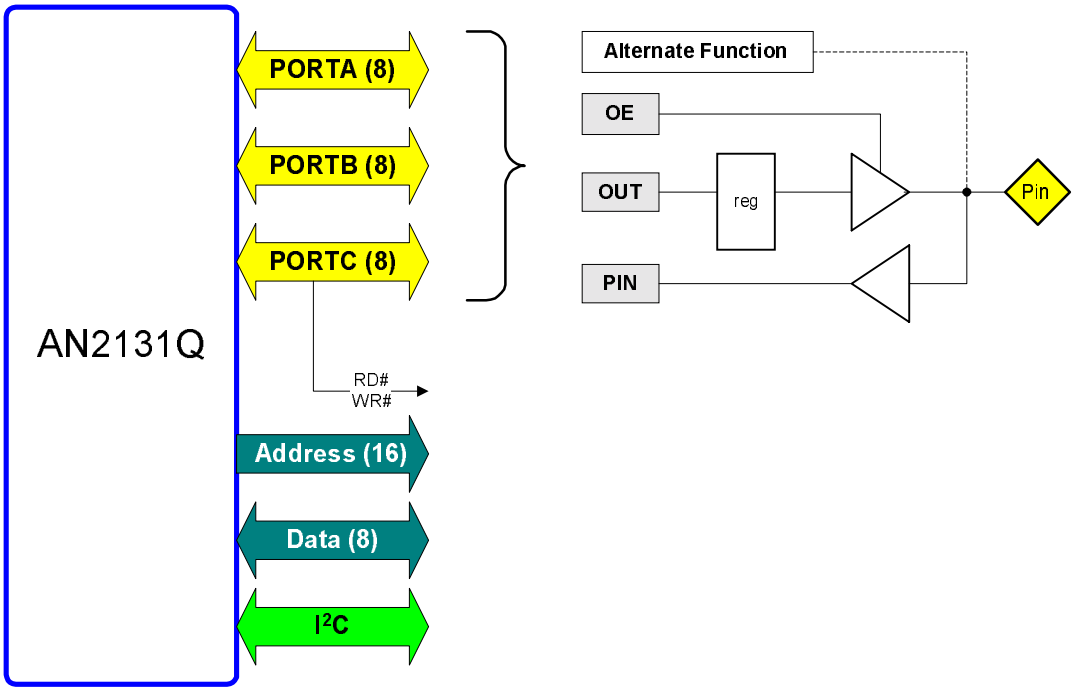

Figure 17. Expanding the AN2131