Keywords: Isolated USB, Medical applications, Isolated applications

APPLICATION NOTE 3891

Isolating USB

Sep 22, 2006

Abstract: If you must isolate a device that also connects to a PC, the USB interface is a natural choice. USB's extensive industry support and simple structure (only four wires in a USB cable) make it a popular PC interface. A USB controller that attaches to your embedded system using the SPI interface is easy to electrically isolate. The SPI interface can run at any speed and consists of simple, unidirectional signals.

Introduction

The Universal Serial Bus (USB) has become the standard way to connect peripherals to personal computers. If you need to electrically isolate a device that is to be connected to a PC, USB is a natural connection interface because of its extensive industry support. Two obvious isolation applications are medical, where PC-based instruments are attached to patients, and industrial, where large supply rail offsets can occur.

USB Signaling Basics

USB operates at three speeds:

- Low speed, 1.5Mbps
- Full speed, 12Mbps
- High speed, 480Mbps

This article discusses optical isolation of a full-speed (12Mbps) USB connection. A 12Mbps device operates with enough bandwidth for useful data transfers, and employs a data rate that is manageable for designs that use inexpensive optocouplers.

The USB connector contains four wires: two to supply power (V_{BUS} and GND) and two to move the USB data (D+ and D-). The V_{BUS} wire provides 5V of power up to 500mA. The D+ and D- signals are bidirectional, operating at a signaling rate of 12Mbps (83ns per bit cell). The D+ and D- signaling voltage is 3.3V.

The USB Isolation Challenge

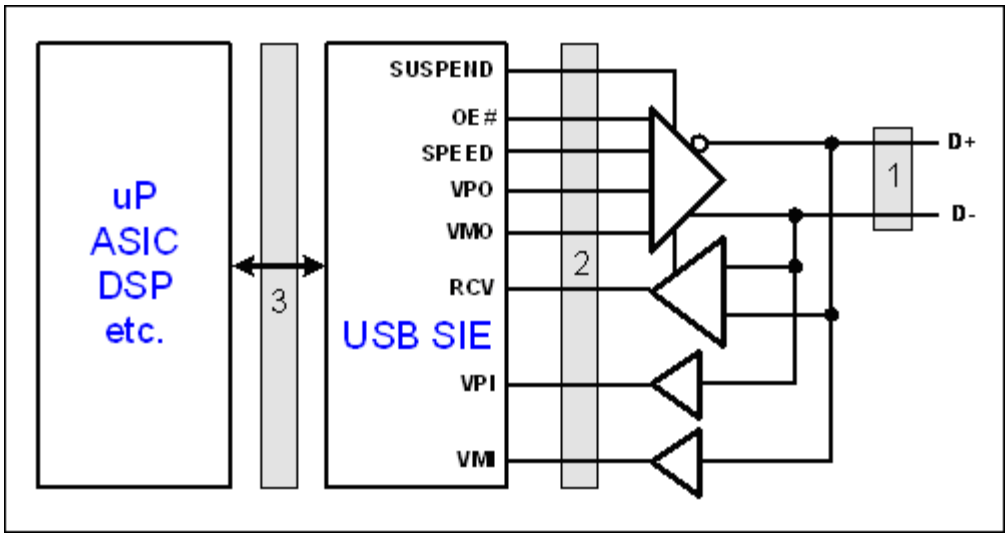


Figure 1. There are three possible interfaces where a USB peripheral could be isolated: site 1, the USB bus itself; site 2, the transceiver interface; and site 3, the application interface. In most current designs the SIE and transceiver are integrated, making interface (2) inaccessible.

USB peripherals are built using the block diagram shown in **Figure 1**. Considering this figure from right to left, a USB transceiver connects to the D+ and D- lines and either drives or receives data under the control of an OE (Output Enable) control pin. The middle block, a USB Serial Interface Engine (SIE) translates the bus signals (as seen and sent by the transceiver) into data bytes and USB signals for use by the application that implements the USB peripheral. The leftmost block is the application circuitry, which might be a microprocessor, an ASIC, or a Digital Signal Processor (DSP).

The gray rectangles, marked 1, 2, and 3, show three possible places to place optocouplers to electrically isolate a USB device from the host computer.

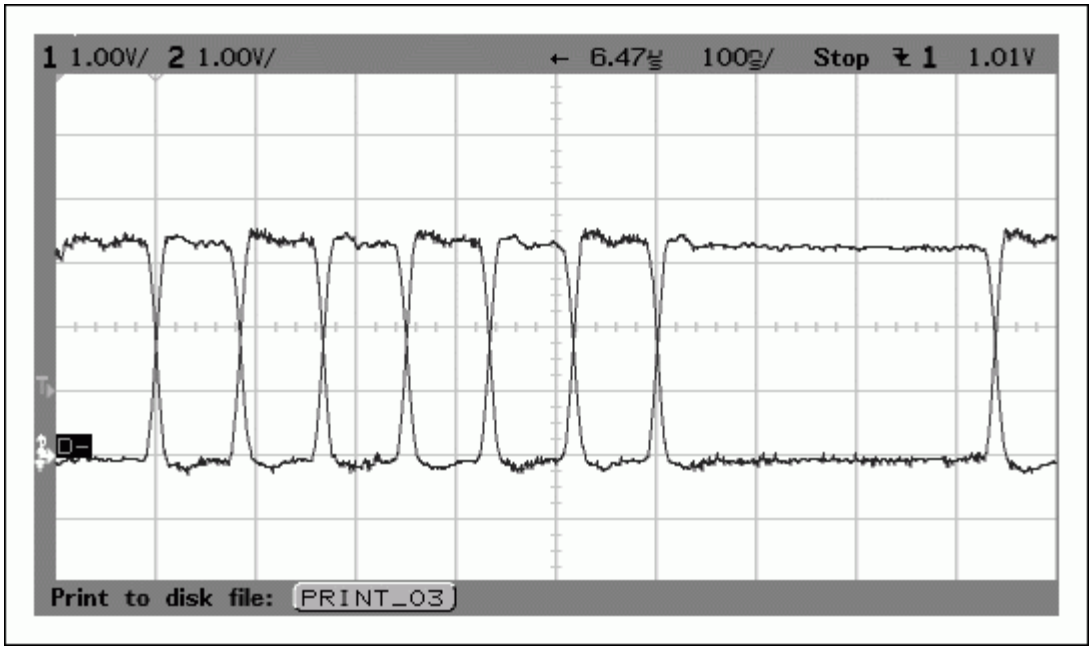


Figure 2. This scope trace shows the USB D+ and D- signals (site 1 in Figure 1) at the beginning of a packet. The 83ns bit time and close matching of rise/fall times make it difficult to maintain signal fidelity through optical isolators. D+ and D- are bidirectional, further complicating the isolation.

Site 1

There are several reasons why optical isolation is not practical on the USB bus wires, as seen in **Figure 2**:

1. The signaling rate is 12MHz, too high for cost-effective isolators.
2. The D+ and D- signals must be carefully matched for propagation delay and skew, a match that is difficult to achieve with optical isolators.
3. The situation is further complicated because the bus is bidirectional, but isolators are unidirectional. Moreover, in a peripheral with an integrated transceiver, the OE signal (which indicates direction) is not accessible.

Site 2

A USB peripheral that uses an external transceiver exposes the transceiver interface. These unidirectional signals could, therefore, be considered for optical isolation. However, this site has the same problem as site 1: too high a data rate, and indeed more signals running at 12MHz to isolate. VPO, VMO, RCV, VPI and VMI all operate at the 12MHz rate, and would need to be carefully matched for delay and skew. Furthermore, this interface is rarely accessible in modern USB designs that incorporate the SIE and transceiver in the same chip.

Site 3

This is the most promising place to do isolation. This interface can operate slower than the USB data and lines rate, and the interface can be constructed entirely with unidirectional signals. An ideal interface would, in fact, use a small number of unidirectional signals that operate at a data rate much lower than the USB 12MHz signaling rate.

An ideal interface that meets these requirements is the SPI (Serial Peripheral Interface), originally defined by Motorola and now widely available in many semiconductor types. SPI is a very popular interface due to its simplicity and performance.

SPI Signaling Basics

Table 1. SPI Signals

SPI Signal	Description	Comments
SS#	Slave Select	Selects the chip for data transmission, supplied by the master
MOSI	Master Out Slave In	Unidirectional data pin
MISO	Master In Slave Out	Unidirectional data pin
SCLK	Serial Clock	Serial clock, supplied by the master

Table 1 shows the four SPI interface signals. SPI is a master-slave interface, where the master initiates and conducts transactions to a single slave. The master provides the slave select (SS#) signal and the serial clock (SCLK) to synchronize data transfers. The SPI interface has four clocking modes, reflecting two mode signals called CPOL (clock polarity) and CPHA (clock phase). These signals are represented in the form (CPOL, CPHA).

Figure 3 illustrates an SPI data transfer between a microprocessor and an SPI slave device. Figure 3 uses SPI mode (0,0), which is the most common. In mode (0,0), the clock is low in its inactive state, and the SPI master makes the MOSI data available before the first SCLK positive edge. SPI data changes on the SCLK falling edge, and is sampled on the rising edge for both master and slave devices.

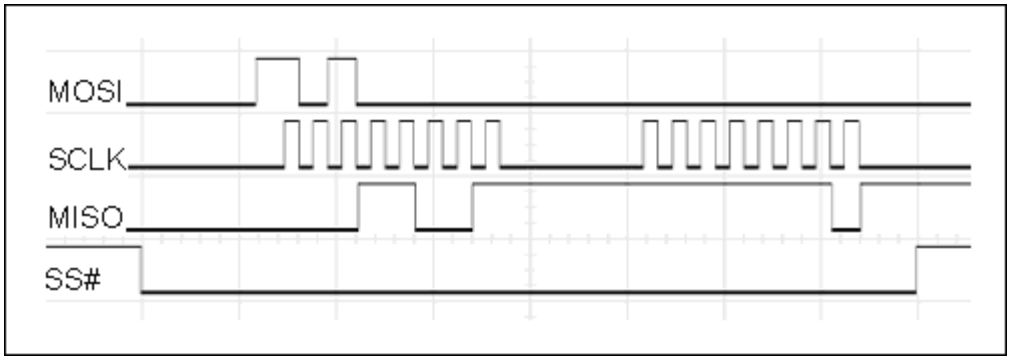


Figure 3. An SPI interface operating in mode (0,0). The same interface can operate in mode (1,1) if the SCLK signal is made active-low (quiescent state is high). These low-frequency signals are easy to optically isolate.

The SPI interface is easy to implement on any microprocessor, even one that does not contain a hardware SPI unit. All it takes is four general-purpose IO (GPIO) pins to construct the Figure 3 signals, and subroutines to read and write bytes by directly toggling the IO pins.

Figure 4 shows the difference in data rates between SPI and USB, when a USB peripheral controller uses the SPI interface. In this figure, the controlling microprocessor functions as a USB keyboard and periodically blinks an LED. The SPI traffic in **Figure 5** represents one LED blink (toggle an output bit); the USB traffic is the USB host asking for keyboard data.

The difference in data rates between these two buses is dramatic. Clearly the lower frequency, unidirectional SPI signals are easier to isolate than the 12MHz bidirectional USB bus signals. The isolation solution becomes very simple with the SPI signals, which can be tailored to run at any frequency to suit the characteristics of the optical isolators.

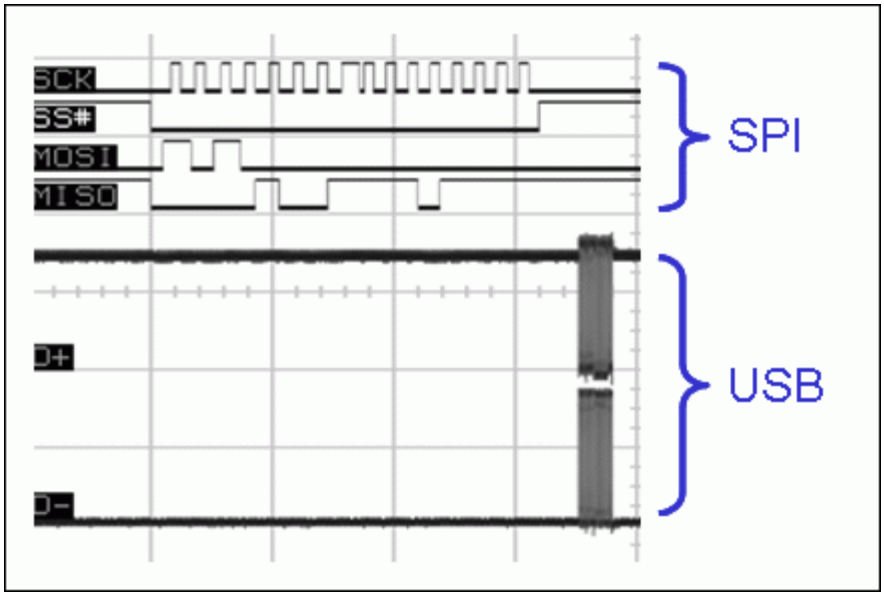


Figure 4. SPI data rate compared with USB signaling rate. The SPI signals lend themselves to easy transmission through optocouplers.

How does the system that produced Figure 4 reconcile the wide difference in data rates between the SPI bus that operates the USB controller and the USB signaling rate? One of the advantages of USB is that it is "self-throttling" with built-in flow control. It accomplishes this control of data flow by using a handshake called "NAK" (Negative Acknowledge), whereby a peripheral tells the host asking for data that it is not ready with data and that the host should try again later.

SPI Data Rates and USB NAKs

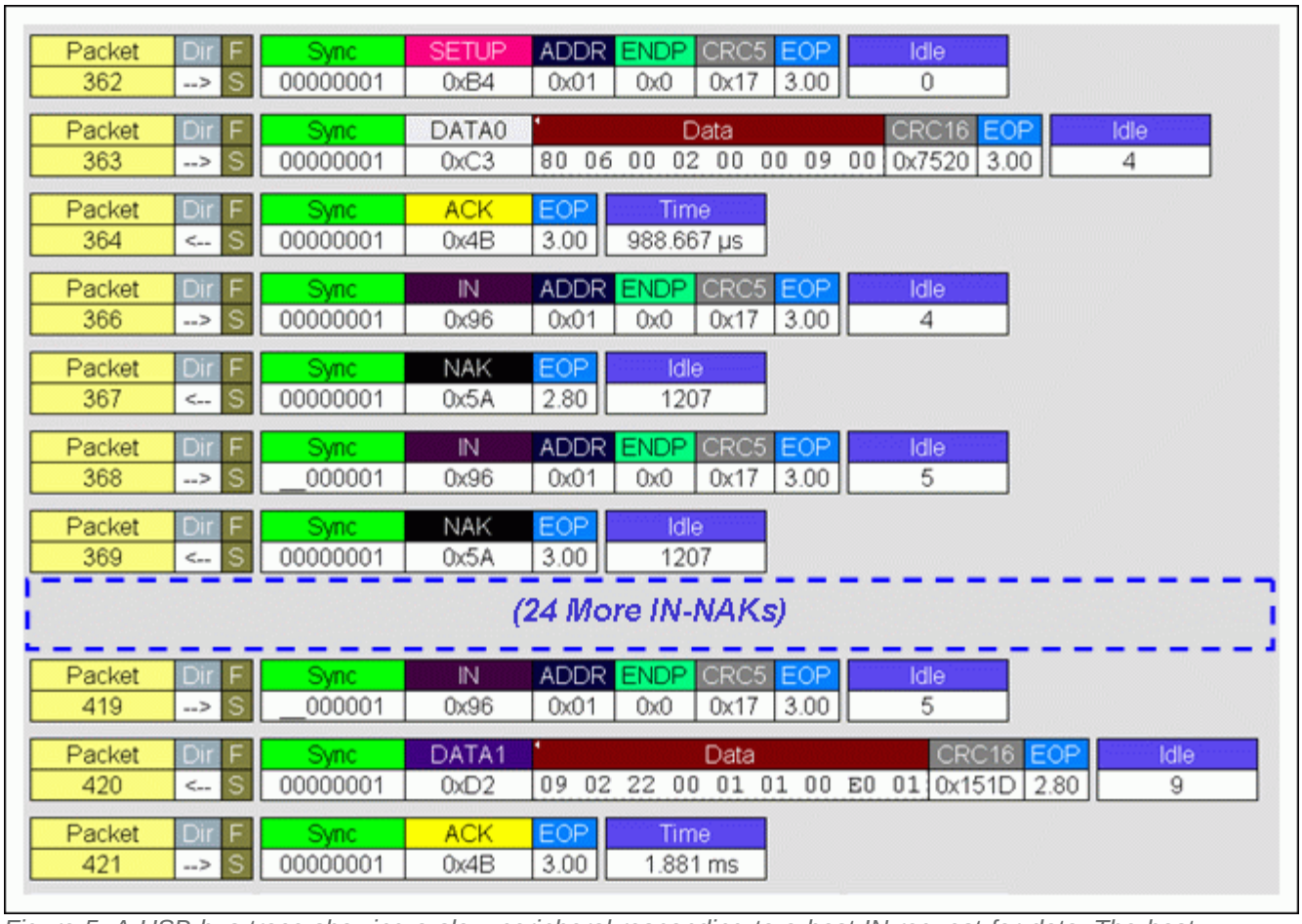


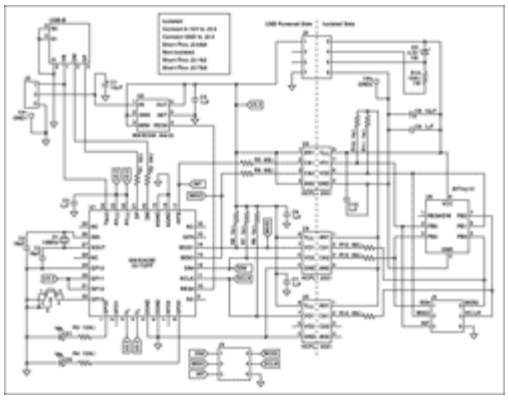
Figure 5. A USB bus trace showing a slow peripheral responding to a host IN request for data. The host requests the data in packets 362-364. The peripheral is ready with the data in packet 420. The intervening IN-NAK packets demonstrate USB flow control. The peripheral responds with NAKs until it is ready with data.

Figure 5 shows the USB flow-control mechanism in action. Starting with packet 362, the host issues a `Get_Descriptor-Configuration` request. The 09 in the second to last byte of packet 363 indicates that the host requests 9 bytes of data from the peripheral. The peripheral acknowledges receipt of the request in packet 364, and then gets busy decoding the request and loading the requested data into its endpoint 0 data FIFO. A slow peripheral will take some time to answer this request, and the relatively slow SPI bus speed further adds to the response time.

After 988.667 microseconds (after packet 364), the host starts asking for the requested data in packet 366. The peripheral does not have the data yet, so the USB hardware automatically responds with the NAK handshake, indicating "I am busy, try again later." The host tries again in packet 368, and gets the same NAK answer from the peripheral. This IN-NAK process continues until packet 419, when the peripheral at last has loaded the requested data and armed its endpoint zero for the data transfer. Now instead of NAK, the peripheral responds with the 9-byte data packet in packet 420, which the host acknowledges in packet 421.

The IN-NAK pairs (dotted rectangle in Figure 5) can occur any number of times, which means that there is no lower limit on the rate at which the SPI interface can operate. This allows a designer to fine-tune the SPI data rate to suit optocoupler choices for practical and cost-effective designs.

An Isolated USB Design Example



More detailed image (PDF, 204kB)

Figure 6. Schematic diagram of an isolated USB design. The left side is powered by the USB bus itself, and the right side uses an isolated power supply. Providing the isolation at the SPI interface makes the design simple and independent of stringent USB bus timing.

Figure 6 is a circuit built around an inexpensive microprocessor, the Atmel® AtTiny13 (U6), HCPL-2531 optocouplers (U3-5), and the MAX3420E (U1), a USB peripheral controller with an SPI interface to its register set. Even though U6 does not contain a hardware SPI unit, the SPI interface is easily managed by "bit-banging" some GPIO pins. U1 provides four general-purpose input and four general-purpose output pins to replace (and add to) the pins used by U6 to implement the SPI interface. This design uses two output pins to drive LED indicators D1 and D2, and one input pin to connect the pushbutton PB1. Since U1 contains its own IO pins that are controlled by the SPI interface, these IOs are inherently isolated from U6 and do not require individual isolation.

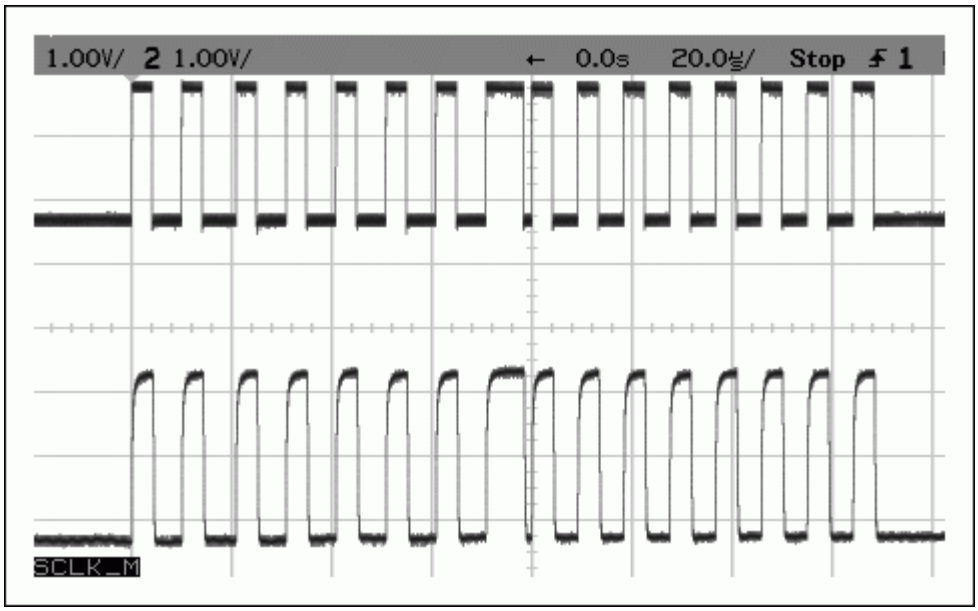


Figure 7. The SPI SCLK signal on both sides of the isolated interface. Top: $ISOV_{CC}$ = 8.3V, ISOGND = 5V. Bottom: Powered by USB: V_{CC} = 3.3V, GND = 0V.

Figure 7 shows scope traces of the SCLK signal on both sides of the isolated interface. The baseline for both

traces is set at the bottom of the screen. The top trace shows the SCLK signal as generated by U6, but offset by 5V.

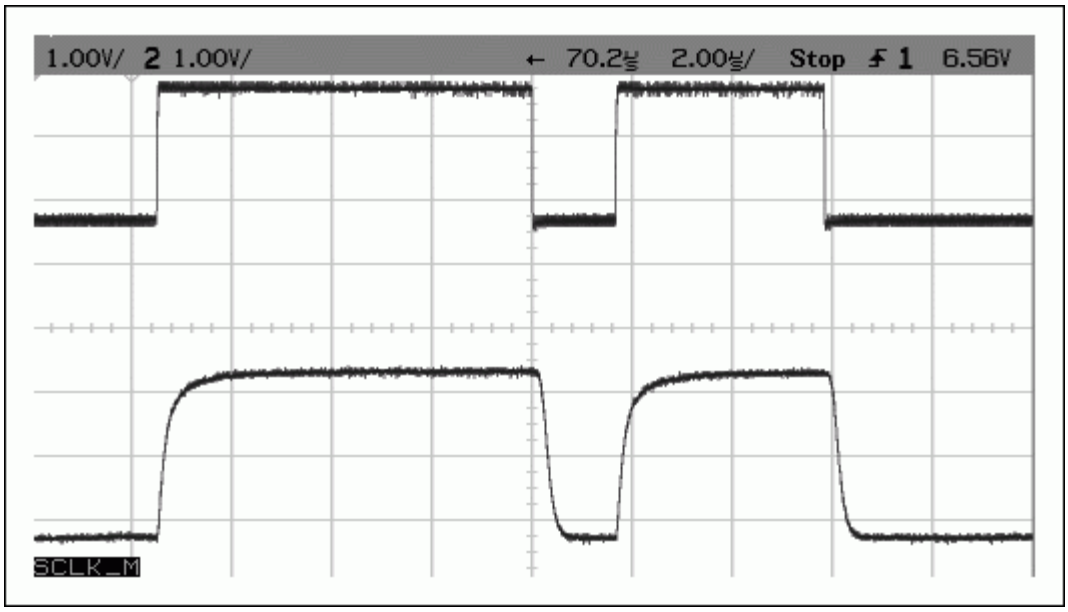


Figure 8. SCK isolated (top) and at the MAX342E (bottom), shown with expanded scale.

Figure 8 is an expanded scale version of Figure 7, and demonstrates the optocoupler performance. The optocouplers have a throughput delay of about 0.5µs with the resistor values chosen for this design. The short SCLK pulse in the center results from a portion of the U6 code that drives the SCLK IO pin. This portion of U6 code is shown in **Figure 9**.

```
; ------------------------------------------------------------------
; Read a MAX3420E register. Uses MAX_Reg(preserved), Updates MAX_Dat.
; ------------------------------------------------------------------
rreg:
      clT
rr2:  mov         dat,MAX_Reg ; 000rrrrr
      lsl         dat         ; 00rrrrr0
      lsl         dat         ; 0rrrrr00 (R)
      lsl         dat         ; rrrrr000  (write bit is clear--b1)
      bld         dat,0       ; rrrrr00T  (T=ACKSTAT bit)
      rcall       send_byte
;
; Now read the MISO data
;
      SCK_LO
r4:   SCK_HI                  ; ready the next input bit
      sec                     ; speculatively set CY
      sbis        PINB,MISO   ; skip if set
      clc
      SCK_LO
      rol         MAX_Dat     ; shift CY into the data byte
      dec         bitcount
      brne        r4
      SS_HI                   ; all done
      ret
;
```

Figure 9. AtTiny13 assembly code to read a MAX3420E register. The SPI interface timing can be fine-tuned here to offer optocoupler cost/performance tradeoffs.

Just before the `r4` label, the SCK signal is driven low and then immediately high again. (`SCLK_LO` and `SCLK_HI` are assembler macros that make it easy to assign IO pins for particular circuit-board layouts without changing the code.)

By inserting a few NOP instructions between these two statements, the narrow pulse in Figure 8 can be lengthened, thereby creating the possibility for using slower (that is, lower cost) optocouplers. This process illustrates the flexibility offered by the SPI interface for optically isolated applications.

Conclusion

Electrically isolating USB has been a challenge due to the high speed, bidirectional nature, and stringent matching requirements of the USB data signals. The isolation problem becomes simpler if the interface (Site 3 in Figure 1) between the USB controller and the application processor is isolated, because this interface can run at any rate. The lower signaling rate of Site 3 makes it well-suited for low-cost optocoupler solutions. As with any isolation design, the fewer lines requiring isolation, the better to save cost. The SPI interface is an ideal candidate for isolation because it uses only four low-speed, unidirectional signals. Because the MAX3420E connects to any controller using the simple SPI interface, it provides an ideal method to add USB functionality to an embedded system, with the added advantage of providing a simple isolation solution. Isolated USB host applications are also possible with the MAX3421E, which functions either as peripheral or host, and which uses an SPI interface identical to the MAX3420E.

A similar article appeared in the July, 2006 edition of *EDN*.

Atmel is a registered trademark and registered service mark of Atmel Corporation.

Related Parts		
MAX3420E	USB Peripheral Controller with SPI Interface	Free Samples
MAX3421E	USB Peripheral/Host Controller with SPI Interface	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 3891: http://www.maximintegrated.com/an3891
APPLICATION NOTE 3891, AN3891, AN 3891, APP3891, Appnote3891, Appnote 3891