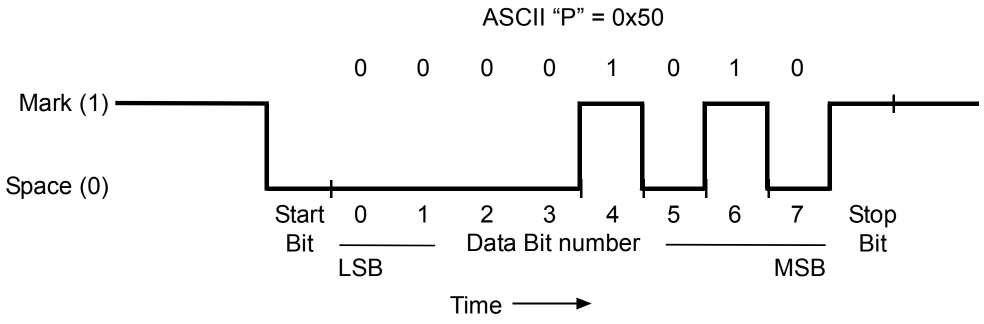
CHAPTER 12

Legacy Digital I/O Interfacing Standards



The EIA RS-232C standard is widely used in PCs on the COM ports for serial data transmission.

12 Legacy Digital I/O Interfacing Standards

Historically, several common digital interface standards have developed over the years to interface computers to their peripheral devices. This chapter will introduce several of the older standards and briefly describe how they function in a hardware design. Each standard has a unique set of hardware and performance tradeoffs. Many devices and ICs are available that use these standards. These interfaces are present in most PCs and are found in many embedded systems including FPGA boards.

12.1 Parallel I/O Interface

The parallel printer interface standard was developed by Centronics in the 1970s and is a widely used standard for transferring 8-bit parallel data. Most PCs have a parallel port. Data is transferred in parallel using eight data bits and standard digital logic voltage levels. Additional status and control bits are required for the sender and receiver to exchange handshake signals that synchronize each 8-bit data transfer. Typically, the parallel printer port is interfaced to two 8-bit I/O ports on a processor. One I/O port is used for 8-bit data transfers and one I/O port for the status and control bits that are used for handshake signals.

The transfer of an 8-bit data value is shown in Figure 12.1. First, the computer waits for the printer's busy signal to go Low. Next, the computer outputs the eight data bits and the computer then sets strobe Low for at least 0.5us. The computer then waits for the printer to pulse Ack Low. The computer must wait for Ack Low before changing the data or strobe lines. The printer may go Busy after it raises Ack. The printer handshake lines are also used to force the computer to wait for events like a slow carriage return or page feed on a mechanical printer or errors like a paper out condition. Sometimes a timeout loop is used to detect conditions like paper out. The UP3 board has a standard printer parallel port connector. With the appropriate hardware, it can be used to communicate with a standard printer.

In addition to printers, some special purpose devices also use the individual parallel port bits in a number of different ways to output digital logic bits to control external hardware. The ByteBlaster adapter you use to program the FPGA is one such example.

The original parallel interface supported only unidirectional data transfers from a computer to a printer. Recent parallel port standards such as IEEE 1284 ECP and EPP support bidirectional and faster data transfers between an external device and the computer. In these newer modes, another control bit from the computer specifies the data transfer direction and tri-state gate outputs are used in both the computer and printer to drive the data lines bidirectionally.

Parallel cables will only work for relatively short distances. The RS-232C standard in the next section supports longer cables with fewer wires, but it also has lower bandwidth and data transfer rates.

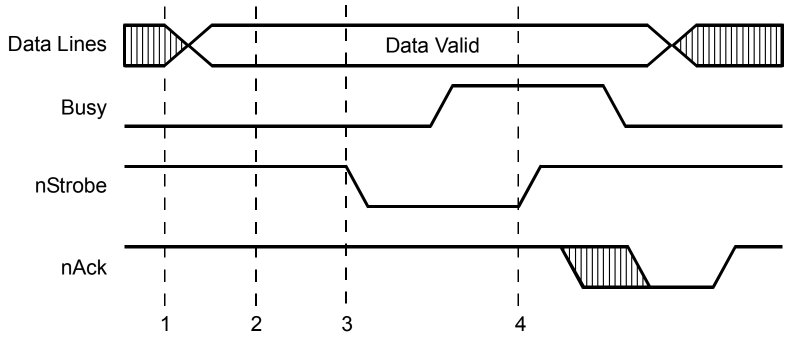


Figure 12.1 Parallel Port transfer of an 8-bit data value

12.2 RS-232C Serial I/O Interface

The Electronics Industry Association (EIA) RS-232C Serial interface is one of the oldest serial I/O standards. In Europe, is it also called V.24. 8-bit data is transmitted one bit at a time serially. Most PCs have an RS-232C serial COM port. Serial interfaces have an advantage in that they require fewer wires in the cable than a parallel interface and can support longer cables. In RS-232C's simplest implementation, only three wires are used in the cable. One wire for transmit data (TD), one for receive data (RD) and one for signal ground (GND). Individual bits are clocked in and out serially using a clock signal. The frequency of this bit clock is called the serial interface's baud rate. (Baudot was a French engineer that developed an early serial interface for the telegraph.) Since two different signal wires are used for receive and transmit, serial devices can be transferring data in both directions at the same time (full-duplex). The ASCII character code is typically used on serial devices, but they can also be used to transfer 8-bit binary values.

The baud rate clock is not synchronized by using a signal wire connected between the sending and receiving devices, rather it is asynchronous and is derived by a state machine watching the serial data bit transitions occurring at the receiver. For this to function correctly, the transmitter and receiver must be setup to operate at the same clock or baud rate. Even though they have the same clock rate, the clock phase must still be synchronized between a serial transmitter and receiver by examining the incoming serial data line. The hardware logic circuit needed for this common serial interface is called a Universal Asynchronous Receiver Transmitter (UART).

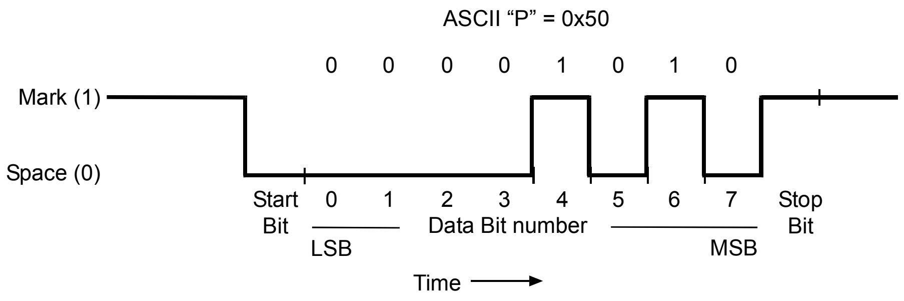


Figure 12.2 RS-232C Serial interface transmission of an 8-bit data value

Figure 12.2 shows the transmission of a single ASCII character over an RS-232C serial interface. The serial bit has two states. Mark is the high state (>3V) and Space is the low state (<-3V). Older generation serial devices will have around +12V and -12V levels for Mark and Space. Note that for the proper RS-232 voltage levels, a standard digital logic output bit will have to have its voltage levels converted for use in a serial interface. Special ICs are normally used for this RS-232C voltage conversion. To reduce the need for additional circuits, these ICs also generate the required DC supply voltages from the standard digital logic DC power supplies. This special IC chip is already present on the UP3's serial interface. FPGA logic elements can be used to build the UART hardware function.

The idle state is High (Mark). Whenever the interface starts sending a new 8-bit data value, the line is dropped Low (Space) for one clock cycle (baud rate clock). This is called the start bit. The eight data bits are then clocked out during the next eight baud clocks in low to high bit order. The highest data bit is sometimes used as a parity bit for error detection, when only seven data bits are used instead of eight. After the data bits are clocked out, the bit goes high for one clock. This is called the Stop bit. Sometimes at low baud rates, two Stop bits are present. Note that at least 10 clocks are required to transfer an 8-bit data value.

Typically, UARTs transfer 8-bit data values in and out to other internal logic using an 8-bit parallel I/O port interfaced to a processor. Extra UART status bits can be read by the processor that indicate another 8-bit data value can be sent to the UART or another 8-bit data value is available to read in from the UART. Since serial transmission is very slow compared to a processor's clock, these status bits must be checked in software or hardware for their proper state or the processor will send/receive data faster than the UART can produce or consume it. Other status bits can also be used to detect various error conditions.

A UARTs transmitter uses a shift register clocked by the baud rate clock to convert the 8-bit parallel data to eight serial bits. Start and stop bits are automatically added by the UART’s hardware.

At the other end of the serial cable, another UART’s receiver uses the Stop and Start bits to reset its internal state machine that is attempting to synchronize its receive clock phase to the incoming serial bit data. This state machine synchronizes the receive clock phase whenever it sees an edge on the incoming serial line. Note that several consecutive bits could be the same value inside the eight data bits, so there is not an edge transition on every single clock.

A UART typically uses an internal clock that is eight or sixteen times the baud rate to watch for edges on the incoming serial data line. UARTs also use this faster clock and a counter to attempt to sample the data bits in the middle of each bit’s time frame to minimize the possibility of reading in an incorrect value near an edge. Since long wires are allowed on an RS-232C serial interface, there will likely be noise and ringing present whenever the serial bit changes. Clocking in the bit in the middle of its time frame greatly increases the reliability of the interface. A second shift register is used for serial to parallel conversion in the UART’s receiver circuit.

Some serial devices also require additional hardware handshake lines to stop and start the flow of a new 8-bit data value over the serial interface. These handshake lines require additional signal wires in the cable used to connect the serial device. Some of the more commonly used handshake lines are RTS (request to send), CTS (clear to send), DCD (data carrier detect), DSR (data set ready), and DTR (data terminal ready).

There are two types of serial devices defined in the RS-232C standard, data terminal equipment (DTE) and data carrier equipment (DCE). A standard RS-232C serial cable is designed to connect a DTE device to a DCE device. When connecting two serial devices of the same type, a special null modem cable or adapter is needed. A null modem exchanges the TD and RD signal lines at one end of the cable along with several connections on the handshake lines. If you experience problems when connecting a new serial device, the various handshake and null modem cable options can be quickly checked using a low-cost in-line RS-232C analyzer breakout box.

The UP3 board contains a RS-232 serial connector, and it has the required voltage conversion IC needed for serial data transmission.

12.3 SPI Bus Interface

The serial peripheral interface (SPI) bus created by Motorola in the 1980s is used primarily for synchronous serial communication between a host processor and peripheral ICs. Four signal lines are used: Chip Select (CS), Serial Data Input (SDI), Serial Data Output (SDO), Serial Clock (SCLK). CS and SCLK are outputs provided by the master device. The slave devices receive their clock and chip select inputs from the master. If an SPI device is not selected, its SDO output line goes into a high impedance state (tri-state). The number of serial bits transferred to the slave device varies from device to device. Each slave device contains an internal shift register used to transfer data.

Two types on connections between master and slave devices are supported as seen in Figure 12.3. In a cascaded connection, all slaves in the chain share a single chip select line driven by the master. The master device outputs SDO and it connects as an input to a slave device's SDI input. A slave's SDO output connects to another slave's SDI input. The serial data cascades through all of the slaves and the final slave in the chain connects its SDO line to the master's SDI input to complete the chain. In this configuration, the slave devices appear as one larger slave device, the data output of one device feeds into the input of another device, thus forming one large shift register.

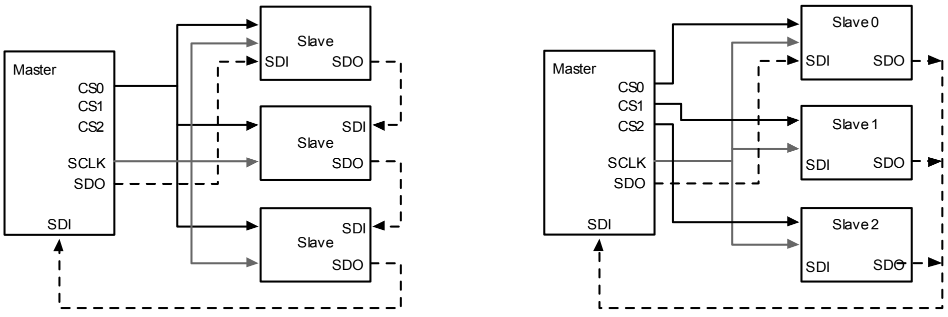


Figure 12.3 The two SPI slave device configuration options.

The second SPI configuration option supports independent slave devices, each device has its own unique chip select input line coming from the master. The master's SDO output connects to each slaves SDI input. The slave's SDO tri-state outputs are connected together and to the master's SDI input. Only the selected slave's SDO output is driven, the others are tri-stated.

Multiple masters are also supported in SPI. Several SPI modes are supported with serial data being valid on either the rising edge or the falling edge of the clock. Serial clock rates can range from 30 kHz to 3 MHz depending on the devices used. Most commonly, devices place new data on the bus during the falling clock edge and data is latched off the bus on the rising edge after it stabilizes, but you will need to check data sheets for specific master and slave devices to confirm this since some devices use the opposite clock edges.

Some Motorola literature may use different names for the SPI signals. CS may appear as SS, SDI as MOSI, and SDO as MISO. In National Semiconductor products, SPI is also known as Microwire. SPI devices are also available in several different voltage supply levels ranging from 2.3 to 5 volts. Since SPI uses a common clock, the hardware interface is simpler than RS-232C serial.

12.4 I^{2}C Bus Interface

The Inter IC (I^{2}C) bus is a widely used standard developed by Phillips in the 1980s for connecting ICs on the same circuit board. Many small ICs now include I^{2}C pins to transfer data serially to other ICs. For lower bandwidth signals, a serial interface has an advantage in that it requires fewer interconnect lines. The I^{2}C bus uses two signal wires called SCL and SDA. SCL is the clock line and SDA is the 1-bit serial data & address line. A common ground signal is also needed. The SCL and SDA lines are open drain. This means that the output is only driven Low, never High. An external pull-up resistor pulls the lines High whenever there is not a device driving the lines Low.

In an FPGA with tri-state output pins, you can simulate open drain outputs by tri-stating the output whenever the bit should go High and only driving the output signal Low. Even though there are multiple devices on the I^{2}C bus, only one pull-up resistor is used for the entire I^{2}C bus.

Devices on the I^{2}C bus are masters or slaves. The slaves are the devices that respond to bus requests from the master. Each slave is assigned its own unique 7-bit I^{2}C bus address. Since both address and data information is transferred over the bus, the protocol is a bit more involved than SPI. When the master needs to talk to a slave, it issues a start sequence on the I^{2}C bus. In a start sequence, SDA goes from High to Low while SCL is High. To stop an I^{2}C sequence, the master sends a stop sequence command. In a stop sequence, SDA goes from Low to High while SCL is High. Start and stop sequences are the only times a change may occur in SDA while SCL is High.

The master drives the SCL clock line to transfer each new I^{2}C serial bit. To force a wait, a slave device can drive SCL Low. Therefore, before each new I^{2}C SCL clock, the master checks to see if SCL is being forced Low by a slave. If it is, the master must wait. SCL clocks are typically up to 100 kHz with 400 kHz available on some new devices.

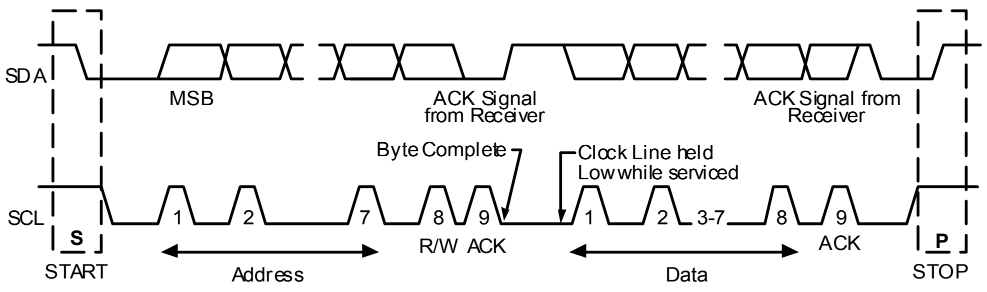


Figure 12.4 I^{2}C interface serial transmission of an 8-bit data value

All address and data transfers contain eight bits with a final acknowledge (ACK) handshake bit for a total of nine bits. All address and data transfers send the High bits first, one per SCL clock bit High. In an address transfer, the

7-bit address is sent and the eighth bit is a R/W bit (0=read, 1=write). Some IC datasheets just append this final R/W bit to the address field and show an 8-bit address field (with even 8-bit addresses for read and odd for write).

The last bit in all data and address transfers, bit nine, is an ACK from the slave. The slave normally drives ACK Low on the last SCL cycle to indicate it is ready for another byte. If ACK is not Low, the master should send a stop sequence to terminate the transfer.

As seen in Figure 12.4, when a master wants to write data to a slave device, it issues the following bus transactions:

1. Master sends a start sequence.
2. Master sends the 7-bit I^2C address (high bits first) of the slave with the R/W bit set Low.
3. Master sends the 8-bit internal register number to write.
4. Master sends 8-bit data value(s). Highest bits first.
5. Master sends a stop sequence.

When a master wants to read data from a slave device, it issues the following bus transactions:

1. Master sends a start sequence.
2. Master sends the 7-bit I^2C address of the slave (high bits first) with the R/W bit set Low.
3. Master sends the 8-bit internal register number to read.
4. Master sends a start sequence.
5. Master sends the 7-bit I^2C address of the slave (high bits first) with the R/W bit set High.
6. Master reads the 8-bit data value(s). Highest bits first.
7. Master sends a stop sequence.

In the full I^2C standard, multiple bus masters are also supported with collision detection and bus arbitration. Collision occurs when two masters attempt to drive the bus at the same time. Arbitration schemes must decide which device can drive the bus when multiple masters are present. Some of the newest I^2C devices can support a high-speed 3.4 MHz clock rate, 10-bit addresses, programmable slave addresses, and lower supply voltages. The System Management Bus (SMB) bus developed by Intel in 1995 that is used for temperature, fan speed, and voltage measurements on many PC motherboards is based on the I^2C bus. On the UP3 board, the real-time clock chip and the serial EEPROM chip use an I^2C bus interface. Many new TVs, automobiles, and other consumer electronics also contain I^2C interfaces between chips for control features.

SPI and I^2C both offer good support for communication with low-speed devices. SPI is better suited to applications that need to transfer higher bandwidth data streams without the need for explicit address information. Some of the most common SPI examples are analog-to-digital (A/D) and digital-to-analog (D/A) converters used to continuously sample or output analog signals. Since addressing is required for I^2C, it requires more hardware, but with advances in VLSI technology these additional hardware costs are minimal. In 2005, one FPGA vendor calculated that a single I/O pin on an FPGA package costs as much as 50,000 transistors inside the chip.

12.5 For Additional Information

The books *Parallel Port Complete* and *Serial Port Complete* by Jan Axelson published by Lakeview Research (www.lvr.com) contain complete details on using parallel and serial ports. The full I^2C specification is available from Philips Semiconductors (www.phillipssemiconductor.com) and SMB at (www.smbus.org). The Motorola MC68HC11 data manual (www.freescale.com) and various National Semiconductor manuals (www.national.com) have more information on SPI. Analog Devices (www.analogdevices.com) makes a wide variety of A/D and D/A converters with SPI and parallel interfaces.

12.6 Laboratory Exercises

1. Interface a printer with a parallel port to the UP3 board's parallel port. Connect the two devices using a printer cable. Design logic using a state machine or a processor core for the FPGA to transfer data and handle the handshake lines. You may want to use an older printer so that any problems with your design will not damage the printer. Be careful not to generate tri-state bus conflicts on the parallel data lines by making sure you drive the data direction bit to the proper state. Have the UP3 print a short ASCII message on the printer ending with an ASCII form or page feed to print the message on a page. A form feed may be needed to cause the printer to print since most printers store characters in an internal page buffer.

2. Interface the FPGA board's serial port (DE1, DE2 or UP3) to a PC serial port using a serial cable. Run a serial communications program on the PC. Send a short message to the PC from the FPGA and display the data from the PC on the FPGA board's LCD panel or seven-segment LEDs.

3. Design an I^2C interface for the UP3 board's real-time clock chip. Display the time from the chip on the UP3 board's LCD display. Don't forget to check the UP3 board's jumper settings and battery for the real-time clock chip. The data sheet for the clock chip contains address and data formats.

4. Obtain an IC chip with an SPI or I^2C interface and design an interface for it on the FPGA board. Chips with SPI interfaces include analog-to-digital converters, digital-to-analog converters and various sensor modules. Header I/O and power connections are available on the FPGA boards with 5V or 3V logic levels. Consult the board’s user manual for pin assignments.