Serial Protocols Compared

John Patrick - May 31, 2002

Serial Protocols Compared

Serial buses dot the landscape of embedded design. From displays to storage to peripherals, serial interfaces make communications possible.

Many serial communication interfaces compete for use in embedded systems. The right serial interface for your system depends on several key factors. In this article I will describe seven of the most common serial interfaces, to help you decide which bus is right for your next project.

Why serial?

There are many different reasons to use a serial interface. One of the most common is the need to interface with a PC, during development and/or in the field. Most, if not all PCs have some sort of serial bus interface available to connect peripherals. For embedded systems that must interface with a general-purpose computer, a serial interface is often easier to use than the ISA or PCI expansion bus.

A benefit of serial communications is low pin counts. Serial communications can be performed with just one I/O pin, compared to eight or more for parallel communications. Many common embedded system peripherals, such as analog-to-digital and digital-to-analog converters, LCDs, and temperature sensors, support serial interfaces.

Serial buses can also provide for inter-processor communication-a network, if you will. This allows large tasks that would normally require larger processors to be tackled with several inexpensive smaller processors. Serial interfaces allow processors to communicate without the need for shared memory and semaphores, and the problems they can create.

This isn't to say that parallel buses have no use. For operational fetches, address and data buses, and other microprogram control, parallel buses have always been the clear winner. "Memory-mapping" peripherals has been a technique commonly used for systems with address and data buses. This tendency allows parallel access to off-chip peripherals. However, with many 8-bit microcontrollers (let alone 8-pin) with no external address/data bus available for designs, memory-mapping is not an option.

Terminology

Before we get into the individual interface details, we should define several terms:

- On an asynchronous bus, data is sent without a timing clock. A synchronous bus sends data with a timing clock.
- Full-duplex means data can be sent and received simultaneously. Half-duplex is when data can be sent or received, but not at the same time.
- Master/slave describes a bus where one device is the master and others are slaves. Master/slave buses are usually synchronous, as the master often supplies the timing clock for data being sent along in both directions.
- A multi-master bus is a master/slave bus that may have more than one master. These buses must have an arbitration scheme that can settle conflicts when more than one master wants to control the bus at the same time.
- Point-to-point or peer interfaces are where two devices have a peer relation to each other; there are no masters or slaves. Peer interfaces are most often asynchronous.
- The term multi-drop describes an interface in which there are several receivers and one transmitter.
- Multi-point describes a bus in which there are more than two peer transceivers. This is different from a multi-drop interface as it allows bidirectional communication over the same set of wires.

RS-232

TIA/EIA-232-F (typically referred to as RS-232) is a common interface that can be found on almost every personal computer. RS-232 is a complete standard, not only including electrical characteristics, but physical and mechanical characteristics as well, such as connection hardware, pin-outs, and signal names. A point-to-point interface, RS-232 is capable of moderate distances at speeds up to 20Kbps. While not specifically called out in the specification, speeds of greater than 115.2Kbps are possible, provided that connections are short and proper grounding is used. Cable lengths of 30 feet are common, and cables of over 200 feet can be attained with low-capacitance cable.

An RS-232 bus is an unbalanced bus capable of full-duplex communication between two receiver/transmitter pairs, named data terminal equipment (DTE) and data communication equipment (DCE). Each one has a transmit signal that is connected to the receive signal on the other end. As such, there is a pin difference between the two sides. (Your PC is a DTE, while the connected peripheral is DCE.)

Each transmitter sends data by varying the voltage on the line. A voltage higher than 3V is a binary zero, while a voltage less than --3V is a binary one. Between these voltages, the value is undefined. To convert from logic levels (0 and 5V) to these levels and back, an RS-232 conversion IC, such as the 1488, 1489, or ubiquitous MAX232, can be used.

Typical RS-232 communication consists of a start bit, data bits, parity bits (if any), and stop bit(s). When communicating with PCs, the typical format is eight data bits, no parity, and one stop bit (8N1). Seven data bits, even parity, and one stop bit (7E1) is also common. A start bit is often a zero and a stop bit is often a one, as shown in Figure 1. The official specification does not delineate any communications protocol, including the use of start/stop bits.

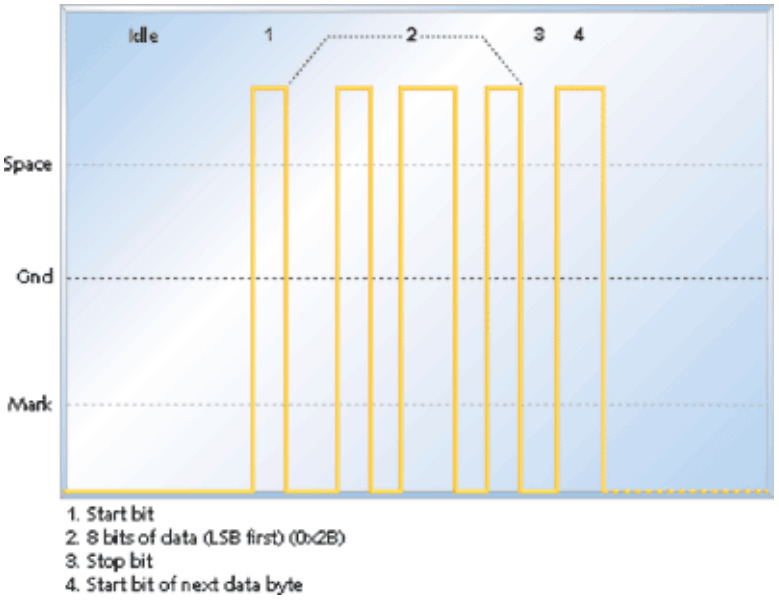

Figure 1: RS-232

Many embedded systems that use the RS-232 bus either interface with PCs or PC peripherals such as modems. Other systems use RS-232 so that bus traffic can be monitored easily with an inexpensive protocol analyzer or a PC equipped with two serial ports.

Almost every microcontroller vendor has products that include hardware support for RS-232, called Universal Asynchronous Receiver Transmitters (UARTs). UARTs are often interrupt-driven and capable of speeds up to 115.2Kbps with little software overhead, although this varies by architecture.

RS-422 and RS-485

TIA/EIA-422-B (typically referred to as RS-422) and TIA/EIA-485-A (typically referred to as RS-485) are balanced, twisted-pair interfaces capable of speeds up to 10Mbps and distances up to 4,000 feet. Being differential buses, each uses signals from 1.5V to 6V to transmit the data. (With a differential, balanced bus, noise immunity is increased over a comparable single-ended, unbalanced bus such as RS-232.)

The RS-422 interface is a multi-drop interface, giving unidirectional communication over a pair of wires from one transmitter to several receivers, up to 10 unit loads (UL). If the devices receiving the data wish to communicate back to the transmitter, the designer must use a separate, dedicated bus between each receiver and the transmitter. (Using this return bus will allow full-duplex transmissions.) For that reason, RS-422 is seldom used between more than two nodes.

The RS-485 interface, on the other hand, is a bidirectional communication over one pair of wires between several transceivers. The specification states that the bus can include up to 32 UL worth of transceivers. Many manufacturers produce fractional-UL transceivers, thereby increasing the maximum number of devices to well over 100.

The RS-422 and RS-485 interfaces often use the same start bit/data/stop bit format of RS-232. In fact, several converters exist to go from RS-232 to RS-485 and back. Do keep in mind, however, that RS-232 is a full-duplex interface, while RS-485 is half-duplex.

Several microcontroller manufacturers provide built-in UARTs that boast special RS-485 abilities.

I^2C

The Inter-Integrated Circuit bus (I^2C) is a patented interface developed by Philips Semiconductors. (In order for an IC manufacturer to implement the I^2C bus in hardware, they must obtain licensing from Philips.)

The I^2C bus is a half-duplex, synchronous, multi-master bus requiring only two signal wires: data (SDA) and clock (SCL). These lines are pulled high via pull-up resistors and controlled by the hardware via open-drain drivers, giving a wired-AND interface.

I^2C uses an addressable communications protocol that allows the master to communicate with individual slaves using a 7-bit or 10-bit address. Each device has an address that is assigned by Philips to the manufacturer of the device. In addition, several special addresses exist, including a "general call" address (which addresses every device on the bus) and a high-speed initiation address.

During communication with slave devices, the master generates all clock signals for both communication to and from the slave. Each communication begins with the master generating a start condition, an 8-bit data word, an acknowledge bit, followed by a stop condition or a repeated start. Each data bit transition takes place while SCL is low, except for the start and stop conditions. The start condition is a high-to-low transition of the SDA line while the SCL line is high. A stop condition is a low-to-high transition of the SDA line while the SCL line is high (see Figure 2). The acknowledge bit is generated by the receiver of the message by pulling the SDA line low while the master releases the line and allows it to float high. If the master reads the acknowledge bit as high, it should consider the last communication word not received and take appropriate action, including possibly resending the data.

A start condition, a one data, a zero data bit, and a stop condition

Figure 2: I^2C

I^2C has a rather interesting feature called clock stretching, which is done when the slave device is unable to process the bit and wishes for more time. When this happens, the slave pulls the SCL line low. Since the signal behaves as a wired-AND, when the master releases the SCL line while the slave is "stretching" the clock, the master should notice that the line stays low. Upon seeing this, the master waits until the slave has processed the data bit and released the line. Once released by the slave, the SCL line floats back high, signaling to the master to send the next data bit..

The I^2C bus has three speeds: slow (under 100Kbps), fast (400Kbps), and high-speed (3.4Mbps), each downward compatible. Philips has specified a recommended wiring arrangement should the signals need to leave the circuit board.

I^2C bus distances are often limited to on-board communications, although I have heard of developers using I^2C successfully over distances of 50 feet! The true limit to I^2C distances is the bit-rate and capacitance of the bus. As such, for off-board communications, I^2C is practically limited to under 10 feet for moderate speeds.

For more details on I^2C, read David and Roee Kalinsky's "Beginner's Corner: I^2C" (August 2001).

SPI

The Serial Peripheral Interface (SPI) is a synchronous serial bus developed by Motorola and present on many of their microcontrollers.

The SPI bus consists of four signals: master out slave in (MOSI), master in slave out (MISO), serial clock (SCK), and active-low slave select (/SS). As a multi-master/slave protocol, communications between the master and selected slave use the unidirectional MISO and MOSI lines, to achieve data rates over 1Mbps in full duplex mode. The data is clocked simultaneously into the slave and master based on SCK pulses the master supplies. The SPI protocol allows for four different clocking types, based on the polarity and phase of the SCK signal. It is important to ensure that these are compatible between master and slave.

In addition to the 1Mbps data rate, another advantage to SPI is if only one slave device is used, the /SS line can be pulled low and the /SS signal does not have to be generated by the master. (This capability is, however, dependent on the phase selection of the SCK.)

A disadvantage to SPI is the requirement to have separate /SS lines for each slave. Provided that extra I/O pins are available, or extra board space for a demultiplexer IC, this is not a problem. But for small, low-pin-count microcontrollers, a multi-slave SPI interface might not be a viable solution.

For more detail on SPI, read David and Roee Kalinsky's "Beginner's Corner: Serial Peripheral Interface" (February 2002).

Microwire

Microwire is a three-wire synchronous interface developed by National Semiconductor and present on their COP8 processor family.

Similar to SPI, Microwire is a master/slave bus, with serial data out of the master (SO), and serial data in to the master (SI), and signal clock (SK). These correspond to SPI's MOSI, MISO, and SCK, respectively. There is also a chip select signal, which acts similarly to SPI's /SS. A full-duplex bus, Microwire is capable of speeds of 625Kbps and faster (capacitance permitting).

Microwire devices from National come with different protocols, based on their data needs. Unlike SPI, which is based on an 8-bit byte, Microwire permits variable length data, and also specifies a "continuous" bitstream mode.

Microwire has the same advantages and disadvantages as SPI with respect to multiple slaves, which require multiple chip select lines. In some instances, an SPI device will work on a Microwire bus, as will a Microwire device work on an SPI bus, although this must be reviewed on a per-device basis.

Both SPI and Microwire are generally limited to on-board communications and traces of no longer than 6 inches, although longer distances (up to 10 feet) can be achieved given proper capacitance and lower bit rates.

1-Wire

Dallas Semiconductor's 1-Wire bus is an asynchronous, master/slave bus with no protocol for multi-master. Like the I^2C bus, 1-Wire is half-duplex, using an open-drain topology on a single wire for bidirectional data transfer. However, the 1-Wire bus also allows the data wire to transfer power to the slave devices, although this is somewhat limited. Though limited to a maximum speed of 16Kbps, bus length can be upwards of 1,000 feet, given the proper pull-up resistor.

For more detail on the 1-Wire bus, read H. Michael Willey's "One Cheap Network Topology" (January, 2001).

Bit banging

Should you not have hardware support for any of the above, it is possible to use general-purpose I/O pins. The act of software controlling a serial communication is often referred to as "bit banging," as the software is truly "banging away" at the adopted "serial port."

Bit banging requires the software to be cognizant of the exact timing required for each bit, for it must toggle an output line for every bit change (as well as monitor the receive pin for incoming data, if such interface is full-duplex). Luckily for embedded developers, quite a few bit-banging routines are available on the Internet for every serial bus described here, and for use in almost every microcontroller architecture. In fact, several microcontroller manufacturers have developed and published their own such routines.

Catching the right bus

As you can see, there is a multitude of serial communication buses to choose from. (And we didn't even discuss wireless, networks, Firewire, and USB protocols.) Your choice in a serial bus should not only meet the needs of the product today, but also be available as well as viable for the life of the product. I hope this has helped you decide which serial interface is proper for your current embedded design.

Table 1 Protocol comparison

Name	Sync /Async	Type	Duplex	Max devices	Max speed (Kbps)	Max distance (Kbps)	Pin count(1)
RS-232	async	peer	full	2	20(2)	30(3)	2(4)
RS-422	async	multi-drop	half	10(5)	10,000	4,000	1(6)
RS-485	async	multi-point	half	32(5)	10,000	4,000	2
I^2C	sync	multi-master	half	-7	3,400	<>	2
SPI	sync	multi-master	full	-7	>1,000	<>	3+1(8)
Microwire	sync	master/slave	full	-7	>625	<>	3+1(8)
1-Wire	async	master/slave	half	-7	16	1,000	1s

Notes

-1 Not including ground.

-2 Faster speeds available but not specified.

-3 Dependent on capacitance of the wiring.

-4 Software handshaking. Hardware handshaking requires additional pins.

-5 Device count given in unit loads (UL). More devices are possible if fractional-UL receiv

-6 Unidirectional communication only. Additional pins needed for each bidirectional commu

-7 Limitation based on bus capacitance and bit rate.

-8 Additional pins needed for every slave if slave count is more than one.

John Patrick is an embedded software engineer for L-3 Communications/Electrodynamics, Inc. He holds BS and MS degrees in electrical engineering from the University of Oklahoma. When not spending time with his new family, John can be found in the garage working on one of his robots. He can be reached via e-mail at j.s.patrick@ieee.org.

References

1. "Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange," TIA/EIA-232-F Standards, Electronics Industries Association Engineering Department.

2. "Electrical Characteristics of Balanced Digital Interface Circuits," TIA/EIA-422-B Standards, Electronics Industries Association Engineering Department.

3. "Standard for Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems," TIA/EIA-485-A Standards, Electronics Industries Association Engineering Department.

4. "The I^2C Specification," Version 2.1, Philips Semiconductors.

5. Aleaf, Abdul, "Microwire Serial Interface," Application Note AN-452, National Semiconductor.

6. Goldie, John, "Summary of Well Known Interface Standards," Application Note AN-216, National Semiconductor.

7. Nelson, Todd, "The Practical Limits of RS-485," Application Note AN-979, National Semiconductor.

8. Wilson, Michael R., "TIA/EIA-422-B Overview," Application Note AN-1031, National Semiconductor.

9. Goldie, John, "Ten Ways to Bulletproof RS-485 Interfaces," Application Note AN-1057, National Semiconductor.

Return to June 2002 Table of Contents