Chapter 14

I²C Interface

INTRODUCTION

The utilization of several peripheral devices in microcontroller-based systems has made it almost impossible to connect a reasonable number of these devices to a low pin-count controller. For example, if we were to connect the following I/O devices to a microcontroller:

- Parallel 8-bit A/D converter.
- Parallel 8-bit D/A converter.
- 4x4 hex keypad.
- Alphanumeric LCD in 8-bit mode.

We would usurp at least 36 I/O pins, which is too large for a low pin-count microcontroller. The question that arises here is how can a small MCU accommodate many peripherals without having to add additional hardware. An answer to this question was formerly attempted by introducing the serial peripheral interface (SPI) covered in Chapter 13. Many SPI-type peripheral devices surfaced on the market as a result. Although SPI interfacing offers high speed data transfer between an MCU and a peripheral, it has the following drawbacks:

- Each peripheral component requires a so-called Slave-Select \( SS \) pin which has to be asserted by the master in order to initiate communication between both ends of the communication link. For instance, if 10 SPI-type peripheral devices were tied to the serial bus, the MCU would have to spare 10 control signals to select amongst them. This is definitely the antithesis of a serial bus.
- SPI interface puts forth some constraints such as clock polarity and clock phase between communicating devices. This is contradictory to the plug-and-play philosophy adopted nowadays when adding new hardware to a system.
- The SPI was not originally designed to configure internal registers in a peripheral device and hence it cannot distinguish between data and commands. It was mainly envisioned to transfer high

DOI: 10.4018/978-1-68318-000-5.ch014
speed data between a microcontroller and data acquisition devices. This was the driving force behind seeking a different communication approach between a microcontroller and peripheral devices.

The I²C bus (Inter-Integrated Circuit), which is 2-wire serial bus, was conceived to provide full-blown network capability between a microcontroller and the peripheral devices clustered around it. This bus was developed by Philips Semiconductors in the mid-1980s and has been updated ever since. This chapter lays out the fundamental underpinnings of the I²C bus and emphasizes several applications pertaining to it.

THE I²C DATA FORMAT

As in the case of SPI interfacing, I²C communication is synchronous and is initiated by the master as well. However, there is no such thing as clock polarity and clock phase nor is there a slave select pin. There is one timing requirement relating the clock position to the data and is adopted as a standard by all I²C chip manufacturers. On top of that, data is transferred in both directions between the master and the slave using only one transmission line. Figure 1, shows a system consisting of a network of I²C devices. SDA and SCL are the open-drain data and clock lines respectively and hence they require pull-up resistors. The open-drain scheme is responsible for ridding the transmission lines SDA and SCL from any bus contention. As a matter of fact, a device not using the bus, electrically disconnects itself from it (high impedance) thereby allowing another one to pull it low (logic ‘0’) or release it (logic ‘1’). A device that is not “talking” on the bus is in a listening mode waiting to be addressed by the master. Each peripheral device tied to the serial bus is associated with a unique device address provided by the manufacturer. The master communicates with one slave at a time by transmitting the device address on the bus. The interrogated device acknowledges the master’s request by pulling the line low. This is how communication is started between the master and one of the slaves. The subsequent sections elaborate further on the I²C protocol, designed to organize bus communication.

The I²C data format prohibits a level change of a data bit while the clock is high, unless signaling is implied. The high or low levels of the of the data line SDA can only change when the clock signal SCL is low as shown in Figure 2 (top). A setup time and hold time must be satisfied according to the specifications provided in the data sheet of the I²C peripheral device. When the data bit changes while the clock line is high (see Figure 2 (bottom)), one of the following signaling conditions takes place:

- **START Condition:** It initiates data transfer between the master and the slave. Starting with a released bus (SDA = SCL = logic high), the master pulls SDA to ground via an internal transistor. This so-called START condition produces a falling edge of SDA while SCL is high.
- **STOP Condition:** It is used to terminate a transaction between the master and the slave. Starting with a pulled-down bus (SDA = SCL = 0), the master releases SCL first then SDA. This generates a rising edge of SDA while SCL is high.
- **RESTART Condition:** In essence, it is a STOP condition followed by a START condition. It is used to terminate a transaction and restart a new one. This signaling scheme is generally used to change the direction of data flow between the master and the slave.