Device Drivers and Interfaces

By David Janowsky, ECE '20

Abstract

This article will discuss how peripheral devices are used and communicate with a computer. It will begin with an explanation of what a device driver is and then compare two common serial interfaces.

Introduction

Modern-day computers and their processors are lightning fast and can perform tasks in seconds that would take humans all day. Though, without the ability to provide input and view output this would all be meaningless for the users. Another often overlooked component necessary for most computer programs and human-computer interactions are the peripheral devices: keyboards, mice, hard drives, sensors, printers, etc. The computer does not inherently know how to communicate with these devices, and thus a device driver is necessary. Device drivers are short programs that allow the computer access to the peripheral [6]. One of the more common functions of device drivers is the ability to read/write data to and from the device. The commands are often communicated over a serial interface, sending individuals '1's and '0's in sequence. This paper will examine the role device drivers play in modern computer architectures and investigate two types of serial interfaces: Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C).

Device Drivers

Purpose

Drivers are the communication system used between external hardware and the internal system of a computer. They act as a translation module, taking code or data from the computer and then sending commands to the device that it can understand. Alternatively, it can go in the reverse direction as well, taking output from the device and modifying it to a format the computer can understand. As such, every driver must be written specifically for the device and operating system [8]. The driver for a device being used on a Windows PC will not work when using that device on Linux.

Commonalities

Most drivers implement the same functionalities regardless of device. Therefore, part of the job of a driver is to abstract the communication with the device. Users need not concern themselves with how the device works, just what the device can do. For most devices this means a proper driver allows a user to send read or write commands from the computer to the device without understanding the details under the hood. The driver should not restrict how the commands are used but must ensure commands are properly executed and do not corrupt the device [3]. An example of this a device that can send and receive data, yet not at the same time. If the device in receiving data and the user issues a send command, the driver must block this command, and only let it through once it is possible to do.

Application

For the Magnetic Navigation project, drivers must be written for a magnetometer and an accelerometer. The device must relay a status to the drivers so it can

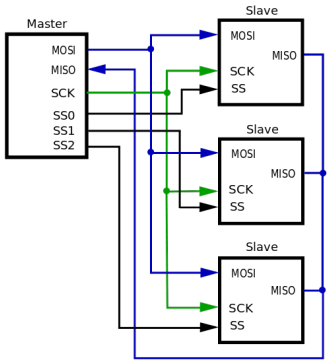

Figure 1. Example SPI Setup (Public Domain) [9]

take appropriate action. Both sensors must notify the driver that new data is ready to be read, and then the driver should send the data to the CPU for further processing or storing. The driver is notified by constantly checking a specific memory of each sensor, waiting for the value to change. [10]

Serial Interfaces

The driver is not enough on its own to effectively communicate with a device. There also needs to be an agreed upon method of communication, or a protocol. Some of the simplest methods are serial interfaces, where the driver and device send and receive messages by sending one bit at a time over a wire. Here, two specific interfaces are explored, the Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I²C).

SPI

SPI is most often used in a setup with 4-wires that connect the host computer (master) to the device (slave). Many devices can be connected as slaves to the same master. For the sake of simplicity, this paper assumes only one slave is present. Two wires are control lines and two transmit data. While there are many names for the wires, some of the most common are presented here. One control line is SCK or SCLK, a clock signal generated by the host and sent to the device. The other control line is SS for Slave Select. When there are multiple slaves connected to the same host, as shown in Figure 1, there is a select line for each device. For the two data lines, one sends to signals to the host and the other sends signals to the device. Often, they are labeled MOSI for Master-Out Slave-In and MISO for Master-In Slave-Out. The timing characteristics for when the value being sent is changed and when it is recorded is shown in Figure 2. [5]

I²C

For I²C (sometimes written as I2C or IIC), only 2 lines and a connection to ground or power are required, shown in Figure 3. With fewer lines to send signals, the protocol is quite complex and restrictive. Data must always be transferred 8 bits at a time, the clock frequency must be in a specific range, etc. The two lines are labeled SDA for Serial Data and SCL for Serial Clock. When no transmission is being sent, these lines should be driven to a high voltage. Every device that connects to these two lines can at any time be either a master (send a signal) or a slave (receive a signal) and has a unique identifier. [7]

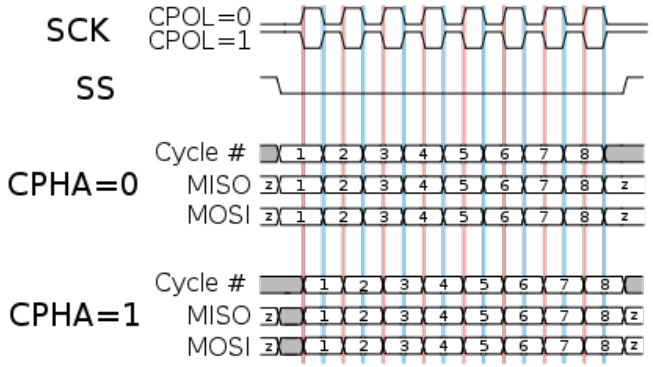

Figure 2. Timing diagram of SPI data transfer (CC BY-SA 3.0) [2]

Every communication begins with a START condition and ends with a STOP condition. After the START condition, the device acting as the master sends out the identifier of the device it wants to communicate with. That device then sends a short signal back to the master indicating the transmission can begin [7]. The timing characteristics of I²C messages is shown in Figure 4

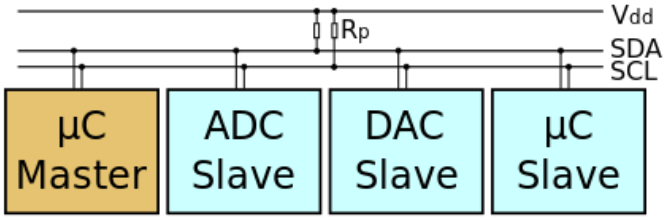

Figure 3. Example I²C Setup (CC BY-SA 3.0) [1]

Comparison

Having explored some of the basics of both SPI and I²C, the advantages and disadvantages of them can be compared. If speed is the priority, SPI is to be preferred. When using SPI, the speed of the transmission is only limiting by what the master and slave can support, and data can be sent in both directions simultaneously. I²C only allows data transfer at specific rates and can only communicate in one direction at a time. However, I²C is superior if space or hardware efficiency is desired. Only two wires are required to connect anywhere from two to 128 devices, whereas SPI requires four lines just to connect a pair of devices. [5]

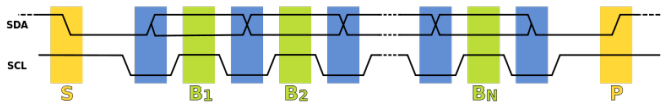

Figure 4. Timing diagram of I²C data transfer (Public Domain) [4]

Conclusion

Device drivers are imperative for communication between a computer and its external devices. Without drivers, input and output devices such as keyboards, mice, and printers would not work as expected, if at all. The communication is often facilitated over a serial interface, commonly one of SPI or I²C.

References

1. Cburnett. (2006) *Sample Inter-Integrated Circuit (I²C) schematic with one master (a microcontroller) and three slave nodes (an analog-to-digital converter (ADC), a digital-to-analog converter (DAC), and a microcontroller)*. https://commons.wikimedia.org/wiki/File:I2C.svg

2. Cburnett. (2006) *SPI bus timing diagram*. https://commons.wikimedia.org/wiki/File:SPI_timing_diagram.svg

3. Corbet, J. (2005). *Linux device drivers*. O'Reilly & Associates, Sebastopol, 3rd ed.

4. Floryan, M. (2007). *A sequence diagram of data transfer on the I²C bus*. https://commons.wikimedia.org/wiki/File:I2C_data_transfer.svg

5. Leens, F. (2016). Introduction to I²C and SPI protocols. URL: https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/

6. Morley, D. (2004). *Understanding computers: today and tomorrow*. Thomson/Course Technology, Boston, Mass., 10th ed.

7. NXP Semiconductors. (2014). UM10204 I2c-bus specification and user manual.

8. Somasundaram, G. and Shrivastava, A., editors (2009). *Information storage and management: storing, managing, and protecting digital information*. Wily Pub., Indianapolis, Ind.

9. *SPI Verbindung in Stern Schaltung*. (2007). https://commons.wikimedia.org/wiki/File:SPI_Stern.svg

10. Stringham, G. (2010). Chapter 7 – design. In Stringham, G., editor, *Hardware/Firmware Interface Design*, pages 149 – 170. Newnes, Boston.