We are currenty aware of issues with the interactives on this page and are working to fix them as soon as possible. Thank you for your patience.

---

Before we begin define serial communication, let's begin by introducing some performance measures. As engineers and scientists we are constantly making choices as we design new product or upgrade existing systems. A **performance measure** is a quantitative metric that the goodness of the system. The metrics and synchronization algorithms presented in this section will apply to all I/O communication.

**Latency** is the time between when the I/O device indicated service is required and the time when service is initiated. Latency includes hardware delays in the digital hardware plus computer software delays. For an input device, software latency (or software response time) is the time between new input data ready and the software reading the data. For an output device, latency is the delay from output device idle and the software giving the device new data to output. In this book, we will also have periodic events. For example, in our data acquisition systems, we wish to invoke the analog to digital converter (ADC) at a fixed time interval. In this way we can collect a sequence of digital values that approximate the continuous analog signal. Software latency in this case is the time between when the ADC conversion is supposed to be started, and when it is actually started. The microcomputer-based control system also employs periodic software processing. Similar to the data acquisition system, the latency in a control system is the time between when the control software is supposed to be run, and when it is actually run. A **real-time** system is one that can guarantee a worst case latency. In other words, the software response time is small and bounded. Furthermore, this bound is small enough to satisfy overall specification of the system, such as no lost data. **Throughput** or **bandwidth** is the maximum data flow in bytes/second that can be processed by the system. Sometimes the bandwidth is limited by the I/O device, while other times it is limited by computer software. Bandwidth can be reported as an overall average or a short-term maximum. **Priority** determines the order of service when two or more requests are made simultaneously. Priority also determines if a high-priority request should be allowed to suspend a low priority request that is currently being processed. We may also wish to implement equal priority, so that no one device can monopolize the computer. In some computer literature, the term "soft-real-time" is used to describe a system that supports priority.

The purpose of our interface is to allow the microcontroller to interact with its external I/O device. One of the choices the designer must make is the algorithm for how the software synchronizes with the hardware. There are five mechanisms to synchronize the microcontroller with the I/O device. Each mechanism synchronizes the I/O data transfer to the busy to done transition. The methods are discussed in the following paragraphs.

---

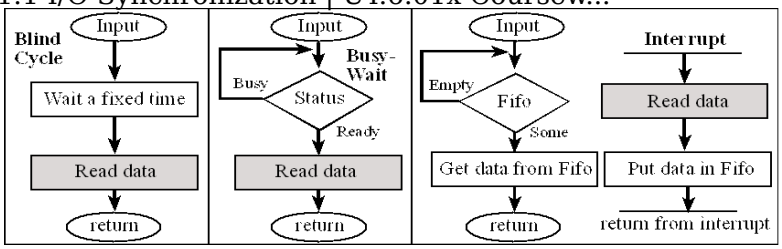



*Figure 11.1. Synchronization Mechanisms*

**Blind cycle** is a method where the software simply waits a fixed amount of time and assumes the I/O will complete before that fixed delay has elapsed. For an input device, the software triggers (starts) the external input hardware, waits a specified time, then reads data from device. Blind cycle synchronization for an input device is shown on the left part of Figure 11.1. For an output device, shown on the left part of Figure 11.2, the software writes data to the output device, triggers (starts) the device, then waits a specified time. We call this method **blind**, because there is no status information about the I/O device reported to the software. It is appropriate to use this method in situations where the I/O speed is short and predictable. We can ask the LCD to display an ASCII character, wait 37 µs, and then we are sure the operation is complete. This method works because the LCD speed is short and predictable. Another good example of blind-cycle synchronization is spinning a stepper motor. If we repeat this 8-step sequence over and over 1) output a 0x05, 2) wait 1ms, 3) output a 0x06, 4) wait 1ms, 5) output a 0x0A, 6) wait 1ms, 7) output a 0x09, 8) wait 1ms, the motor will spin at a constant speed.

Help

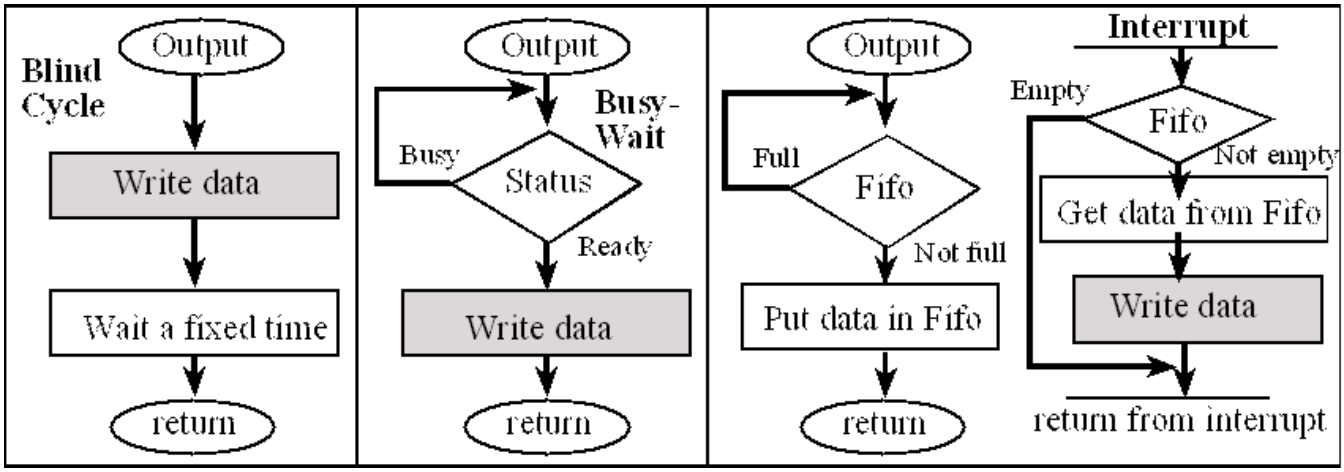
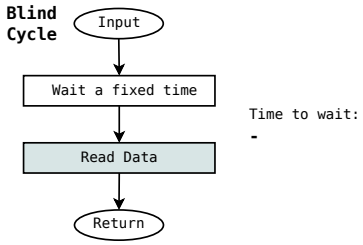


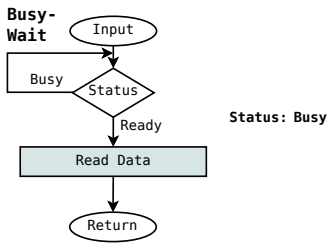

*Figure 11.2. The output device sets a flag when it has finished outputting the last data.*

**Enter an amount of time to wait (1-10):** 5 **Start**



**Busy Wait** is a software loop that checks the I/O status waiting for the done state. For an input device, the software waits until the input device has new data, and then reads it from the input device, see the middle parts of Figures 11.1 and 11.2. For an output device, the software writes data, triggers the output device then waits until the device is finished. Another approach to output device interfacing is for the software to wait until the output device has finished the previous output, write data, and then trigger the device. Busy-wait synchronization will be used in situations where the software system is relatively simple and real-time response is not important. The UART software in this chapter will use busy-wait synchronization.

**Start** **Ready** Click to simulate I/O device becoming ready.




Help

An **interrupt** uses hardware to cause special software execution. With an input device, the hardware will request an interrupt when input device has new data. The software interrupt service will read from the input device and save in global RAM, see the right parts of Figures 11.1 and 11.2. With an output device, the hardware will request an interrupt when the output device is idle. The software interrupt service will get data from a global structure, and then write to the device. Sometimes we configure the hardware timer to request interrupts on a periodic basis. The software interrupt service will perform a special function. A data acquisition system needs to read the ADC at a regular rate. Interrupt synchronization will be used in situations where the system is fairly complex (e.g., a lot of I/O devices) or when real-time response is important. Interrupts will be presented in Chapter 12.

**Start**