Lab 6 Prelab

**Name**: Riley Lawson

**Lab Partner Name (if you worked together and are submitting the same document or mostly the same answers): Drake Dodson**

**Lab Section**: 9

**Submit your prelab document as a PDF file in Canvas under the corresponding prelab assignment. Every student submits their own prelab. Lab partners are allowed to work on the prelab together and submit the same document (if there is actual collaboration on the document). For full credit, the prelab must be submitted prior to the start of lab. Text responses should be typed or printed neatly. You can draw a sketch by hand, or you can use a drawing tool. Try to have started a rough draft of the prelab when you come to class on Tuesday.**

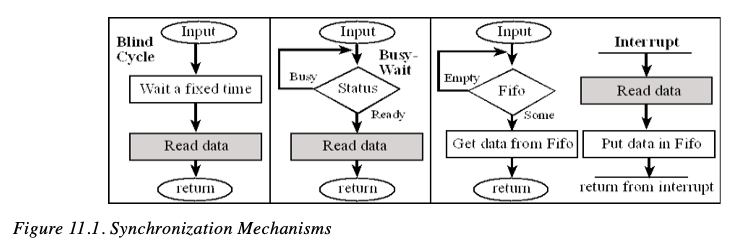
1. Sensor scan data

**Read the section in the lab manual, “More Information about the Sensor Data.” Both the infrared (IR) and sonar (PING) sensors are used to measure distance to objects. The sensors are very different in many ways. Briefly describe one way that they differ.**

The IR sensor uses an infered light to bounce off of objects to get the location of that particular object but can only use it within a 9-80 cm distance. Whereas a PING sensor uses sound to locate object, but must have a return signal and can be used through air and water.

1. I/O synchronization

**Read section 11.1, I/O Synchronization, in the VYES book, stopping before Interactive Tool 11.3** [**http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C11\_SerialInterface.htm**](http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C11_SerialInterface.htm) **(note that the interactive tools in this chapter may only work in the Firefox browser, and if so, ignore them). Read about the synchronization mechanisms shown in figures 11.1 and 11.2.**



**Let’s consider some of the UART registers used when programming these mechanisms. You can assume we are using UART1 as in Lab 6.**

**UART Data (UARTDR): UARTx\_DR\_R**

**UART Flag (UARTFR): UARTx\_FR\_R**

**UART Interrupt Mask (UARTIM): UARTx\_IM\_R**

**UART Raw Interrupt Status (UARTRIS): UARTx\_RIS\_R**

**UART Masked Interrupt Status (UARTMIS): UARTx\_MIS\_R**

**UART Interrupt Clear (UARTICR): UARTx\_ICR\_R**

1. **Which register would be read in the “Read data” box in Figure 11.1?**

UART1\_DR\_R

1. **Which register would be tested in the “Status” decision box in the figure?**

UART1\_FR\_R

1. **Which register indicates that there has been a trigger for an interrupt, such as receiving a character into the UART (one of the conditions necessary to get to the “Interrupt” in the figure)?**

UART1\_RIS\_R

1. **Which register is set during initialization to allow (or enable) an interrupt signal from the UART (one of the conditions necessary to get to the “Interrupt” in the figure)?**

UART1\_MIS\_R

1. **UART Interrupt Mask Register**

**Refer to the UARTIM register description in the Tiva datasheet. Refer to the macro definition for the UART1 Interrupt Mask register in the header file, tm4c123gh6pm.h.**

1. **Write the full #define macro definition as given in the header file.**

#define UART1\_IM\_R (\*((volatile unsigned long \*)0x4000D038))

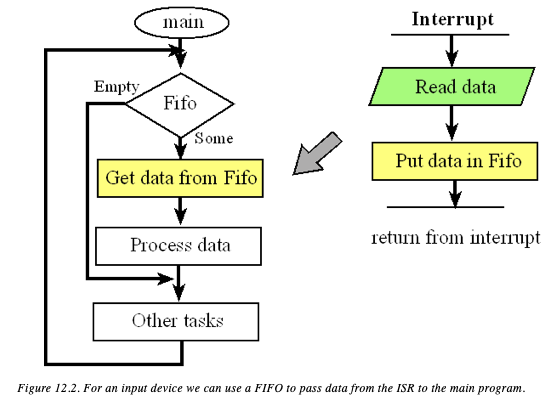
1. **Write the line of code for the IM register to enable RX interrupts from the UART1 device.**

UART1\_...

UART1\_IM\_RXIM |= 0x00000010;

1. **Interrupt behavior**

**Instead of polling (i.e., waiting in a loop) to receive data, an interrupt approach can be used. This lets the main program do other tasks. A separate interrupt handler function is executed whenever a character is received by a UART. The main program doesn’t have to busy-wait and instead just gets data when it needs it. This is shown below in Figure 12.2 from the VYES book. For example, think about an analogy of a classroom: suppose the instructor is like a processor executing a main program of teaching, including receiving student questions. Briefly explain the difference between polling and interrupts with this or another analogy. For example, explain how the instructor receives questions in relation to doing other tasks when a) polling, or b) using interrupts.**

When polling, the device calls the CPU and continously checks the status of the FIFO and tests to see if they require anything (synchonous), whereas an interrupt calls the CPU and lets it know that it needs to be interrupted but doesn’t force it to be (asynchronous).