Kettering University

Microcomputers I

Lab Exercise 7

Software Multiplexing and Demultiplexing

Multitasking

7-segment-display Device Driver

Software Priority Encoders

Interrupt-driven Systems

4 Textbooks and 7 (8) Students

Spring 2022

Prelab (10%): Go over this handout rigorously, do Assignments **1, 2**, and **Error! Reference source not found.** (optional), then upload one handout (prelab) per group to Blackboard in **.pdf** by **11:59 pm** on the **Tuesday** before your lab day.

Lab report: Upload one lab handout (report) per group to Blackboard (in **.pdf**) by **11:59 pm** on the **Sunday** following the lab day and **after** you have done all the assignments, answered all the questions, and shown your lab work to the lab instructor **individually**. A demo sign-up sheet will be posted if necessary.

In the lab report, please correct your prelab incorrect answers, if any.

Assignments Error! Reference source not found. through **Error! Reference source not found.** are optional. You may get 25% extra credit for your work. To get extra credit, you also need to correctly answer the lab instructor's questions.

If you manually scan your prelabs or lab reports for submission purposes, you may scan only the relevant pages of the handout, the pages that should be graded.

Names: Dylan Lozon, Kayla Jones

Objectives

- Get hands-on experience with multiplexing and demultiplexing in software.
- Better understand the concept of BUS.
- Become familiar with the concept of multitasking.
- Better understand interrupts and how useful they are.
- Learn how to prioritize different choices in software.

What to hand in

- Upload this handout (report) in **.pdf**, and after you have done all the assignments and answered all the questions.
- Show your functional digital systems in Assignments Error! Reference source not found. and **Error! Reference source not found.** (optional) to the lab instructor *individually*, and before you turn in your report.
- Also be prepared to *individually* answer the lab instructor's questions regarding today's lab exercise and what your group did.

Note

- When single-stepping, reduce the delays, if any, to almost zero, or put a breakpoint after the delay so that you can use the run button.
- Press the reset button on the trainer board before you upload your code.
- Close the "True-Time Simulator & Real-Time Debugger" window *before* you run the debugger again if you use the microcontroller.
- Take CodeWarrior to the HCS12 Serial Monitor mode should you use the trainer board.
- Use single-stepping (F11) for troubleshooting/testing purposes.
- Use F5 to run your code in one step (and stop when you reach an infinite loop) or up to the next breakpoint.
- Write your programs with proper *indentation* as well as *explanatory* and *short* comments.
- In your comments, use *meaningful/descriptive* names for the register and memory locations that you use.
- Your flowchart (the *formulation* of the project) should not be too close to the word description of the problem, nor too close to the assembly program.
- When you right-click on the Assembly pane (in True-Time Simulator and Real-Time Debugger window), a pop-up menu with ten choices will appear. Here are 3 frequently used choices:
  - **Address …** enter the address of the instruction that you want to display
  - **Display > Code** to display machine code next to each assembly instruction
  - **Format** to select the base of numbers (you usually use Hex)

During your presentation, suppose that you are selling your product (software). When I ask you if your product works, please do not tell me you don't know; this is one of the worst possible answers! If you are ready to demo, you should be able to prove that your product does work!

You should work closely with your lab partner. You are also urged to talk to other students; teach them or learn from them, as this will enhance your performance; however, do NOT copy from them!

Assignments

1. **Prelab: Reading assignment**

There are 4 identical textbooks and 7 students, 1 through 7. Each student has her/his own **active-high** request line. The request line of student i drives pin i of PORT H. For example, the request line of student 3 is tied to pin PH3. To borrow a textbook, students need to assert (**pull up**) their request lines using the DIP switch on the trainer board. In today's lab, first, draw a flowchart and then write a program to design a digital system that looks at these 8 input lines, and displays (on the four 7-segment displays) the numbers (1 through 7) of the 4 requesting students who have the highest priorities among all the requesting students. Let us assume that **student No 7 (PH7) has the highest priory**. PH0 is driven by pushbutton 0 to generate an interrupt as you will see shortly.

Example: If students 6, 5, 3, 2, and 1 assert their request lines, then digits 6, 5, 3, and 2 should be displayed on the four displays from left to right, respectively. In case of fewer requests than 4, the unused displays should turn off.

Remember: You probably designed and implemented an 8-student 2-textbook version of today's project in your Digital Systems I lab but in pure hardware.

Figure 1 shows a big picture of the system: your program should read the 7 input lines driven by the 7 DIP switches, determine the numbers of the asserted lines, convert them to 7-segment code words, place the results in a 7-byte buffer, read the first 4 locations of the buffer (four data producers on the transmitter side), multiplex them on PORT B, and then demultiplex them on the 4 displays (four data consumers on the receiver side). You learned the multiplexing/demultiplexing algorithm in class. Take a close look at the combination of Transmitter, Bus, and Receiver.

Remember: In a simple language, a bus is a set of related lines shared by two or more data producers and/or two or more data consumers.

Note: You may change your algorithm to use a 4-byte buffer (instead of a 7-byte one).

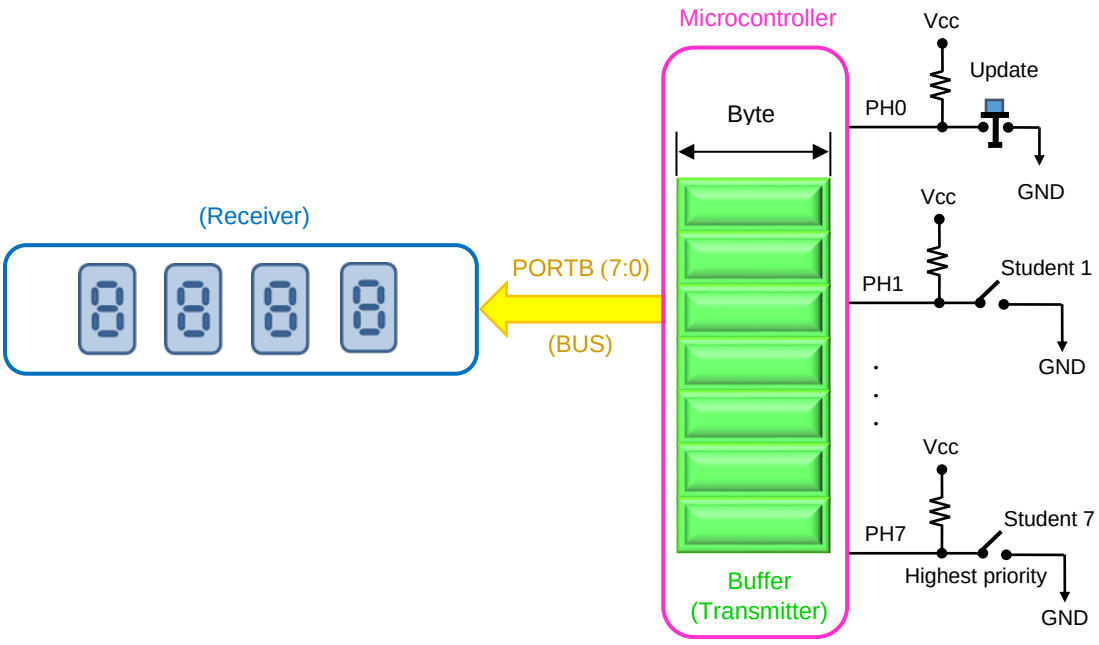


Figure 1. Big picture of our digital system: 7 students and 4 textbooks

More details are illustrated in Figure 2. Pay close attention to the beautiful concept of *multitasking* used in this system:

- Task 1 checks the 7 input lines and updates the buffer accordingly.
- Task 2 displays (on the four 7-segment displays) the four digits stored in the first 4 bytes of the buffer.

These two tasks are obviously performed sequentially, as there is only one processor that takes care of both of them; however, because of the super high speed of task execution (compared to our time constants), they look concurrent!

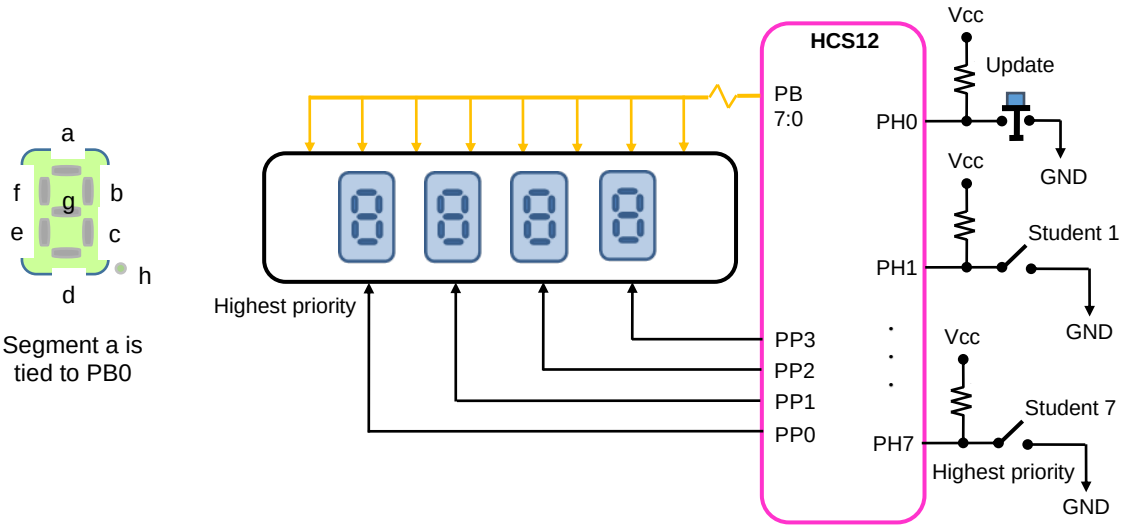


Figure 2. Seven-segment displays, DIP switches, and a pushbutton used in this system

Note: The members of each group should work on the prelab *independently* and then compare their results.

2. Prelab: Use interrupts to design and implement the system. Your infinite loop should keep reading the buffer and displaying the 4 digits unless an interrupt is generated by pushbutton 0. Your interrupt service routine should update the buffer. Pay close attention to task scheduling. It is interesting, is it not?

Complete the partial flowcharts shown in Figure 3 and **Error! Reference source not found.** to describe your infinite loop and ISR, respectively:

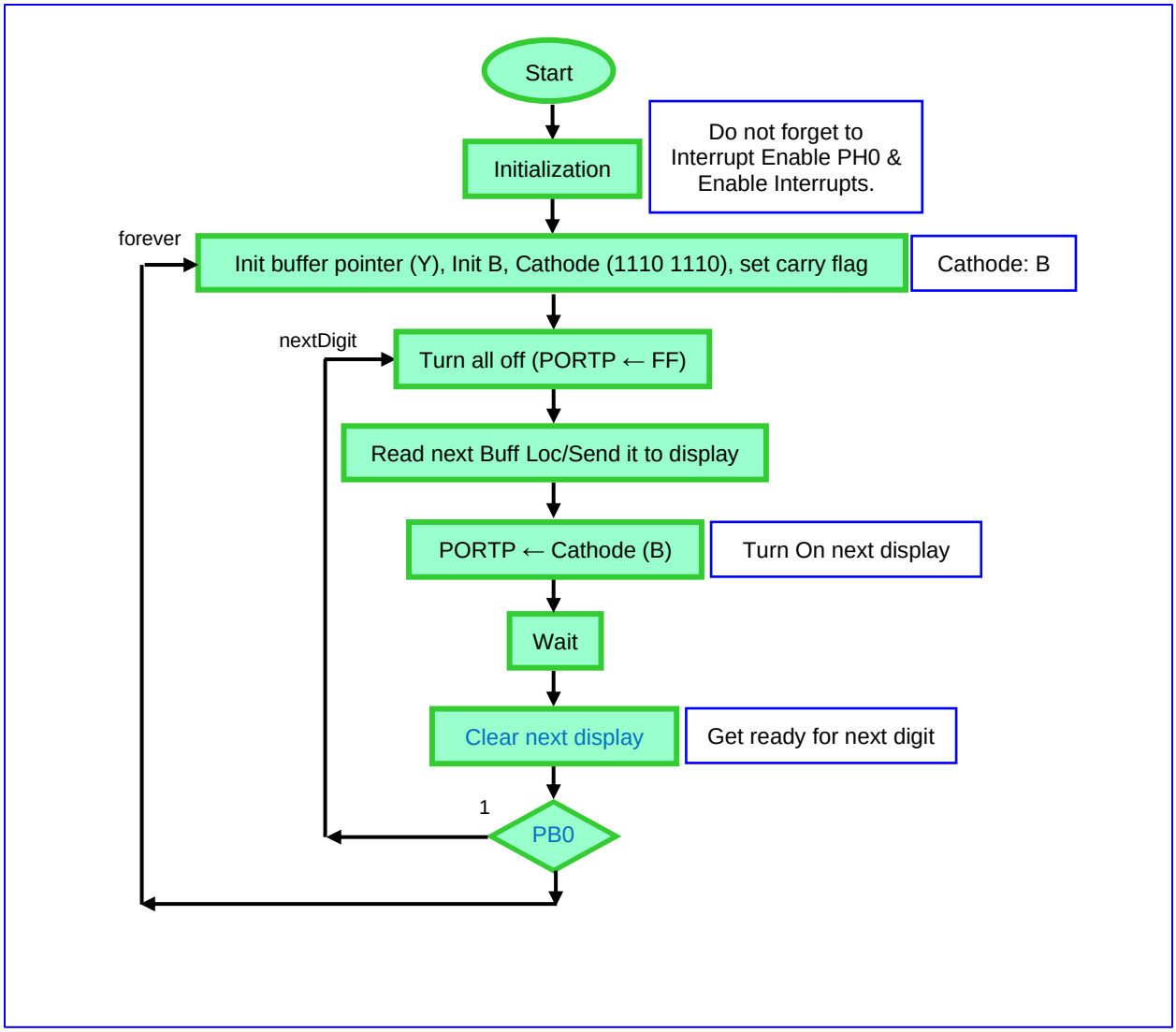
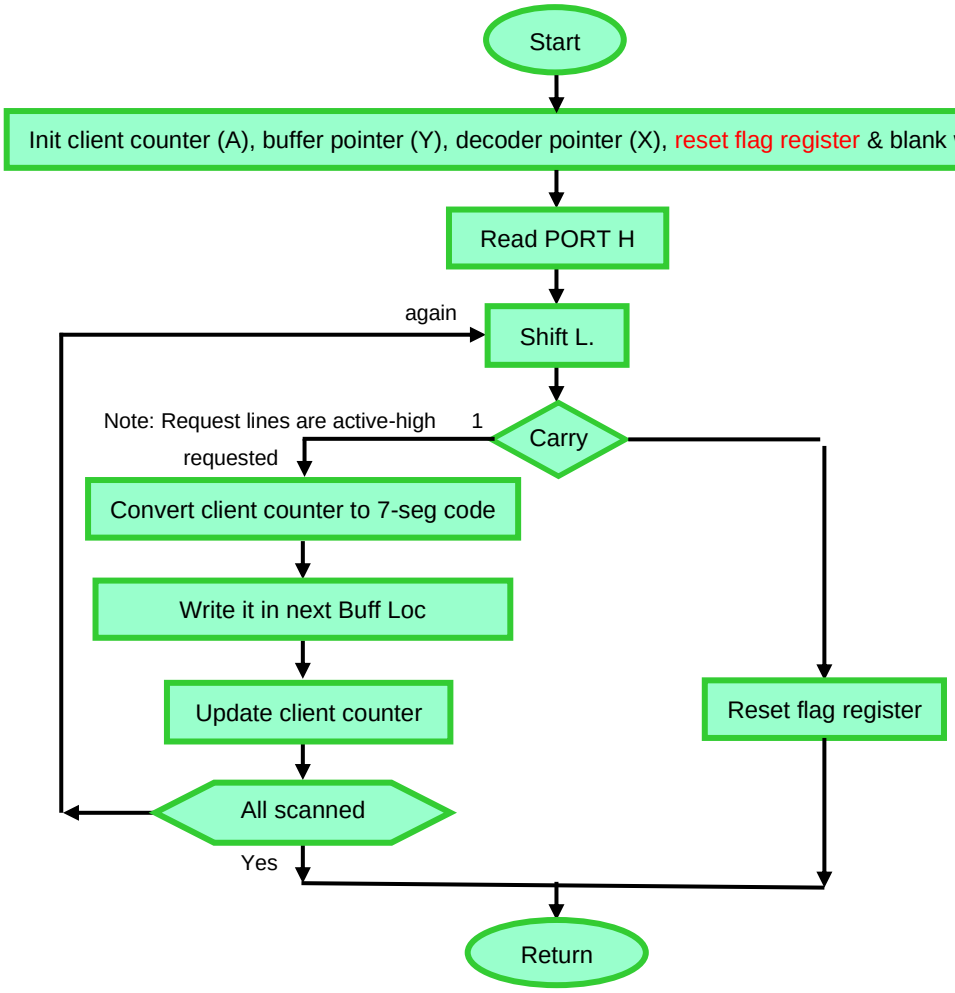


Figure 3. Complete the partial flowchart to model your interrupt-driven infinite loop, which displays the buffer



Remember $FFCC is address of address of ISR

```
        org     $FFCC ; initialize vector address
        dc.w    display_ISR
```