ECE263

Lab 4

Spring 2013

Parallel Ports & LED Display

Revised: 20 Feb 2013 DAL

(intentionally left blank)

Object:

In this lab you will design and code an assembly language program that will display numbers on a 4-digit LED display using time division multiplexing.

Material:

CodeWarrior software package
Dragon 12 Plus Development Board
P&E USB Multilink
CPU12 Reference Guide (CPU12RG)
Agilent 16801A Logic Analyzer
One flying lead probe

Background:

When embedded systems need to display numeric data, seven segment LED displays are often used since they are simple, cheap and can be used in a wide range of lighting conditions. The LED displays can be driven using seven segment decoders such as the 74LS47 that will convert binary data to the appropriate format and then sink or source the required current. These decoders take up space and add more expense to the product therefore many designs use port pins of the microcontrollers to drive the LED displays directly and perform the binary to seven segment conversion in software. In order to cut down on the number of pins that the microcontroller must drive, a technique called time division multiplexing is used to drive the LEDs as explained below. Doing the decoding in software also gives the designer more flexibility in the look of the characters, esp. for the non-numeric codes.

The LED display on the Dragon 12 Plus contains four separate seven segment LEDs each of which has eight pins attached to the anode of the individual diodes for each segment (plus decimal point) and one pin attached to all eight cathodes in common. To illuminate a specific number the user must drive the appropriate anodes (segments) high and drive the cathode (enable) low. With a four digit display, it would normally take 4x8 signals for the anodes and 4x1 signals for the cathodes, or 36 signals total. Using time division multiplexing, that number of pins required can be reduced to 12: 8 for the anodes of all of the displays and 4 for the cathodes of each individual display.

Time division multiplexing is normally associated with communications systems and it is the concept of sending multiple messages over a single wire (or set of wires) so that they appear to be occurring simultaneously. It is accomplished by separating each message into small segments and then sending one segment from each message one after the other in time sequence as illustrated below:

```
A4—A3—A2—A1—A0----
                  \
B4—B3—B2—B1—B0-------→ A4—B4—C4—A3—B3—C3—A2—B2—C2—A1—B1— etc.
                  /
C4—C3—C2—C1—C0-----
```

When applied to the display, the individual digits will be illuminated one at a time in sequence with a short “on time”. After the last digit is illuminated the process starts over again at the beginning and repeats. The flash rate of the sequencing (i.e. the duration of the on time vs. the number of digits) is adjusted to be fast enough so that it appears to the human eye that all of the digits are illuminated simultaneously. It is also important to keep the flash rate within a restricted range because it has been shown that some frequencies of flashing lights can cause seizures in some people with specific medical conditions. For most purposes a flash rate of 60HZ is used which translates to 1/60 HZ = 16.66 msec for the entire sequence. With 4 LED digits to cycle through, it means that each digit should be enabled (illuminated) for approximately 4 msec

On the Dragon 12 Plus board the eight anodes of the four LED displays are wired in common and are all connected to bits of PORT_B as illustrated in Figure 1.

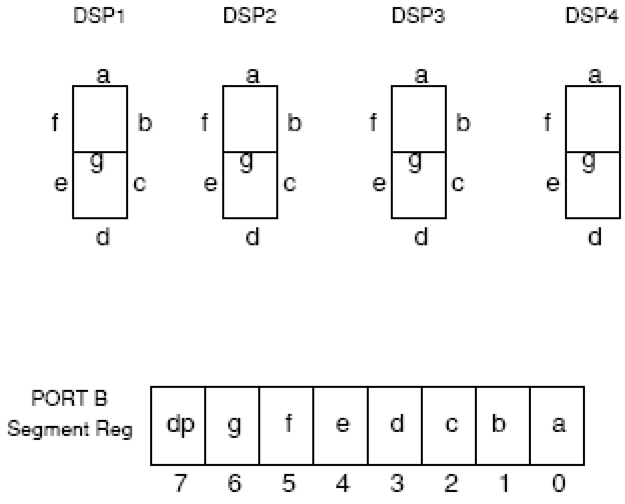


Figure 1. LED Segments (Anodes)

In order to display a hexadecimal character (nibble), it must be converted into the seven segment code that represents the digit on the display. Table 1 shows which segments must be illuminated for each of the sixteen possible hex digits. You must finish filling in the columns labeled Hex with the appropriate code by using Figure 1

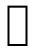
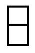
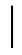
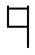
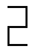
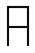
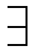
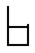
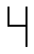
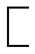
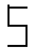
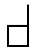
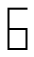
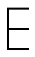
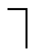
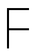
Nibble	Display	Hex	Nibble	Display	Hex
0			8		
1		06	9		
2			A		
3			B		
4			C		
5			D		
6			E		
7			F		

Table 1. Seven Segment Decodes

The cathodes of the four LED digits are individually connected to four bits of PORT_P as shown in Figure 2 so that each digit may be illuminated by itself.

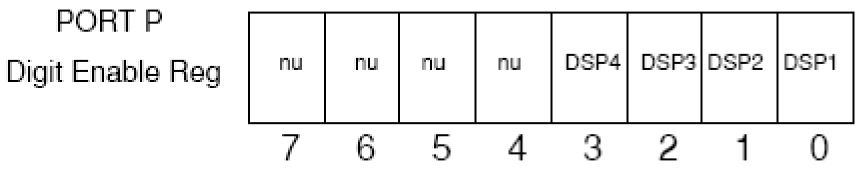


Figure 2. LED Enables (Cathodes)

To display a 1 on the left most digit of the LED display, a $06 would be sent to Port B and then a binary %xxxx 1110 would be sent to Port P. The 0 in bit position 0 would drive the cathode for DSP1 low thereby causing segment b and segment c of that digit to illuminate. Since the other three cathodes are high, all of the segments of those digits would stay off. Finish filling in the columns in Table 2 using Figure 2.

DIGIT	Bit Position	Binary	Hex
DS1	0	xxxx 1110	xE
DS2	1		
DS3	2		
DS4	3		

Table 2. Digit Enable Codes

Preparation:
For this lab you will write a program that will display the contents of a 16-bit variable on the 4-digit LED display on the Dragon 12 Plus board. See the specification for the program attached at the end of this write-up for details on how to write the program. At the start of the lab, show the instructor or the TA the pre-lab that should contain: 1. the flow chart and/or pseudo code that documents the design of the program and 2. a hard copy of the assembly source code for the program.

Remember, the pre-lab is to be the combined effort of both lab partners.

Procedure:
Working with your lab partner, complete the following steps:

1. Create a new project and enter your program as *main.asm*. Generate the executable object file using the *Make* facility and then download the program on the Dragon 12 Plus.
2. In the memory window set the address to location $1000 which is the start of the RAM space. NUM_2_DISPLAY should be the firs variable in this space so, using the edit mode, set the content of memory locations $1000 and $1001 to $5A and $6F respectively.
3. In the Data window, make sure your variable(s) appear by name and have the values you just assigned in Step 2 above.
4. Now we are going to set up the Logic Analyzer. Get one “flying lead” pod from the box at the front of the room.

5. In the Bus/Signal Name column replace *MY BUS 1* with *PortB* and leave the check marks in the boxes under *Slot A Pod 1*.
6. Using the leads of the pod, attach lead labeled 0 through 7 to the pins labeled PB0 through PB7 on the Dragon 12Plus – lower left on pins surrounding the white proto block.
7. Now add four more signals labeled DSP1, DSP2, DSP3 and DSP4 and check the boxes of probes 8 through 11 respectively.
8. Attach leads 8 through 11 of the pod to the pins labeled PP0 through PP3 respectively.
9. Click OK to finish the setup.
10. We now have to set up the trigger condition. In the column labeled Simple Trigger on the main window, click on the box next to the bus/Signal labeled DSP1. In the drop-down window select falling edge.
11. The analyzer is set up so now start the debug process using the *True Time Simulator/Debugger*.
12. Start debugging your program as you normally would using single step and breakpoints as appropriate. Use the logic analyzer to verify the data that you are sending to the LED display and to verify the correct timing.
13. You should see a series of low going pulses on the traces labeled DSP1 through DSP4 and you should see a series of numbers on the bus labeled PORTB. The numbers should correspond to the converted seven segment data you sent to the LED display.
14. Use the cursors to take timing measurements.

Note: the following questions must be answered and included in your lab report.

15. Are the correct numbers being display on the LED digits?
16. Is the timing of the system correct?
17. Are the two bytes of NUM_2_DISPLAY getting updated as you expect?
18. Are the two bytes of NUM_2_DISPLAY changing at the rate you expect?
19. Print out some screen shots of the sequences to show that you have generated the correct timing for the time division multiplexing. Include these prints with an explanation of each one in the lab report.
20. Make sure that you program tests the conversion of all possible nibbles ($0 through $F) for each digit position.
21. Demonstrate the working program to the TA or the instructor.
22. Get the Verification Sheet signed off after demonstrating the operation of your program.

Lab 5: Programming Assignment

As stated previously, your program is going to display the contents of the sixteen bit variable named NUM_2 _DISPLAY which you will place at the start of the variable section of the RAM, i.e. location $1000. Each hexadecimal character (nibble) of NUM_2_DISPLAY will be displayed on one of the four seven segment LED displays on the Dragon 12 Plus board. Your program will have to get the data from the appropriate nibbles and convert the data to a seven segment code as explained earlier. You will do one nibble at a time and you will wait approximately 4 msec before getting the next nibble and displaying it on the next digit of the display. After all 4 nibbles/digits have been done your program will start over at the beginning and do it again. Your program must always get the data for the nibble to convert from the variable location rather that do a one time conversion of all of the nibbles and them merely keep outputting the converted data.

This program will use the program template that you designed for Lab 3. The 4 msc timing will be maintained by doing all of the display updating within TASK_1. When you enter the task, you need to retrieve the contents of a counter that will keep track of which nibble you are working on. If the count is 0, for example, you would get the most significant nibble of the most significant byte of NUM_2_DISPLAY, convert it to seven segment format, write it to PORT_B, and then set PORT_P bit 0 to the low state so that DIGIT 0 will display the expected segments. Before you exit the task you must increment the counter and test for boundary conditions so that the next time TASK_1 is called you will display the next digit.

> Note: you will need to modify the existing code in the template program from Lab 3 that relates to the diagnostic bit banging to Port B. Since the LED display of the Dragon Board is attached to Port B you need to either delete those statements or change them so they write to another port, for example Port M.

The other thing your program for Lab 4 must do is to use Task_3 to modify the contents of NUM_2_DISPLAY. Every time TASK_3 is entered, you are to increment the contents of the upper byte ($1000) and decrement the lower byte ($1001) of NUM_2_DISPLAY. This is intended to simulate the displayed number, i.e. NUM_2_DISPLAY being constantly changed by the operation of the program. For example, NUM_2_DISPLAY might be displaying the number of gallons of gasoline that are flowing into your gas tank from the gas pump.

Memory Map:

PORT NAME	**ADDRESS**
Port A	$0000
Port B	$0001
DDRA	$0002
DDRB	$0003
Port M	$0250
DDRM	$0252
Port P	$0258
DDRP	$025A