Lab 6: Parallel I/O Protocols and Timing Loops

Summary:

Develop assembly language programs involving delay loops and I/O ports and test these functions using evaluation board hardware.

Learning Objectives:

- Gain experience with assembly programming using timing loops
- Learn hardware and software protocols of standard port I/O
- Explore practical use of microcontroller I/O ports

Resources and Supplies:

- CML12S-DP256 development board
- CML12S-DP256 MON12 Manual
- PC with WinIDE Development Environment and MON12 monitor program

Important Reminders:

- It is your responsibility to save the programs you create.
- Pre-lab assignments must be completed before coming to the lab.

Background:

CML12S-DP256 Development Board and User Interface Hardware Overview

The development board was described in Lab 5. Please read through this once again to discover features you may have missed reading it the first time.

Loop Delays

Refer to the Background section of Lab 5 for a discussion on loop delays. Remember that the 68HCS12 of the CML12S-DP256 development board operates at a *clock frequency of 8 MHz.*

Microcontroller Registers

The microcontroller contains many registers to control and interface with I/O devices such as data ports. For the MC9S12DP256B, a memory-mapped microcontroller, these registers are mapped to memory addresses $0000 - $02FF. Appendix C of the ECE331 textbook lists each of these registers. Lab 5 introduced you to the registers associated with Port K, which is connected to LEDs on the development board. The ports used in this lab and their connections to I/O devices on the project board are:

Port K (outputs, bits 0-3)	LEDs
Port M (output, bit 3)	7-segment display enable
Port T (outputs, bits 0-3)	7-segment display
Port AD (inputs, bits 0-3)	4-position DIP switch

Port K (LEDs)

The AXM-0295 project board includes 4 LEDs useful for visual display of program outputs. The LEDs are wired to the lower nibble (4 bits) of the 8-bit Port K of the microcontroller, as shown in Figure 1. The HC12 has two registers associated with Port K, a *data* register where values can be read from or written to, and a *data direction* control register that determines if each bit of the port is an input or and output. When a bit of the *data direction* register is set high (1), the corresponding bit of the *data* register is an output. If it is set low (0), the *data* register bit is an input. Thus, to set Port K to output signals to the LEDs connected to its lower nibble bits, a value of $XF would need to be stored to the Port K *data direction* register (where X means don't care for the upper 4 bits).

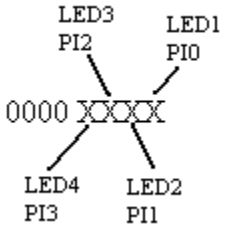

Figure 1. Port K data register.

The Port K registers are located at the following memory addresses:

$0032	Port K data register
$0033	Port K data direction register

These registers can be read or written to (using load or store instructions) just like any other memory address, but writing to the data register only affects bits set as outputs.

Ports M and T (7-segment display)

The AXM-0295 project board includes a 7-segment display useful for visual display of program outputs. It will display decimal values 0 – 9 corresponding to binary inputs 0000 – 1001. The 7-segment display inputs are connected to the lower nibble (bits 0-3) of Port T data register, as shown in Figure 2. The corresponding bits of the Port T data direction register must be set high (1) to configure them as outputs. The 7-segment display also utilizes an enable signal that is wired to bit 3 of Port M, as shown in Figure 3. To enable the 7-segment display, Port M bit 3 must first be set as an output in the Port M *data direction* register, and then a high value (1) must be written to the Port M *data* register.

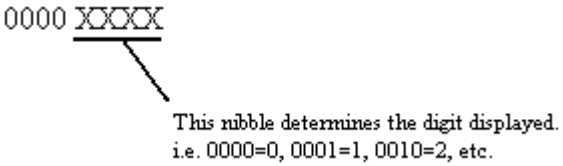

Figure 2. Port T data register

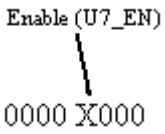

Figure 3. 7-segment display enable bit of the Port M data register.

The Port M and T registers are located at the following memory addresses:

$0240	Port T data register
$0242	Port T data direction register
$0250	Port M data register
$0252	Port M data direction register

It is good practice to ensure your programs configure a data direction register before read/write of the corresponding data register.

Port AD (input DIP switches)

The AXM-0295 project board has a 4 position DIP switch, enabling users to set 4 input bit values (0 or 1) that can be read by the microcontroller. The DIP switch outputs are connected to Port AD. To use Port AD, it must be enabled. Setting the Port AD input enable register ($008D) to $FF will work for this lab. Port AD data register can then be read at $008F. Note that the DIP switch outputs a 'high' in the OFF position and a 'low' (ground) in the ON position.

$008D	Port AD enable register
$008F	Port AD data register

Pre-lab Assignment:

- Read this entire lab assignment so you know what to expect in the lab. Also refer to the *CML12S-DP256 MON12 manual* if you have any questions about the monitor program.
- Complete the steps described in the Pre-lab sheet near the end of this document. Each student must complete his/her own pre-lab before coming to the lab and hand it in to the lab TA at the beginning of the lab.

Laboratory Assignment:

This lab consists of two parts and an optional step in the second part. A check-off sheet is included at the end of this lab document.

- Print the check off sheet. Where indicated, you must record your results on the check-off sheet. After you successfully finish each part of the lab, show the TA your results and ask him to sign the check-off sheet.

Part 1: Using Parallel I/O Ports and the Project Development Board

For Part 1, you must write a program and test its operation on the microcontroller project board. The program should do the following:

- begin at $4000
- setup all necessary ports
- create an infinite loop that
  - reads the binary setting of the 4-bit DIP switch on the project board
  - displays the value on the 7 segment display
  - after a 1 second delay, displays the binary value on the LEDs

Remember, the 7 segment display can correctly display only digits 0 to 9 (0000 to 1001).
Note that the DIP switch outputs a 'high' in the OFF position and a 'low' (ground) in the ON position.

Preparation:

1. Write the assembly code of the switch input display program described above and save it.
2. Debug your code by compiling (Build) it and checking the *.LST* file for syntax errors. You may also try (optional) to simulate the code, but you would have to use memory modify commands within the simulator to act like inputs from the DIP switch.
3. Compile your final error-free ASM code to generate a *.S19* record. Print your final *.LST* file using the text editor and include this in your lab report.

Alternative: If you prefer, you can replace steps 1-3 above by using any text editor to write your code and the ***AsmIDE*** utility program to compile your code, as described in Lab 5.

4. From the PC on your lab bench, launch the ***AxIDE*** utility from the 68HC12 program folder under the 'Start' menu.
5. Connect the CML12S-DP256 board to your computer using the wiring bundle attached to your computer.
6. Connect power to the CML12S-DP256 board.
7. Press the **reset button** on the CML12S-DP256 board. In your terminal window on the PC you should see a message to "PRESS KEY TO START MONITOR…". Press the ENTER key and you should see:

   Axiom MON12 - HC12 Monitor / Debugger V256.5
   Type "Help" for commands.

>_

Note: Refer to page 10 of the *CML12S-DP256 MON12 manual* for additional information on any MON12 command or type 'help' at the prompt for a list of commands. The most useful command are:

MD <address>	***Memory display***
MM <address>	***Memory modify***
RD	***CPU Register display***
RM	***CPU Register modify***
G	***Go = run program***
T	***Trace = step***
BR <address>	***Set Breakpoint***

8. Type '**load**' at the prompt and press the ENTER key.
9. Select the '**Upload**' option. Then click on the '**Browse**' button and select the ***.s19*** file that was created from the assembly process in step 2. Once you have selected the file, click 'OK'.

Inspection:

10. Verify that the program is stored correctly in the memory using the MD command to display the contents of memory. ***MD <address>*** will display the contents of the given memory address. Using the information in the CML12S-DP256 manual, see if you can display the entire block of memory that holds your program with a single command. Record the contents of memory addresses $4000-$4006 on the check off sheet.
11. Comment on the check-off sheet how you can use these values to verify your program is correctly stored on the microcontroller.

Testing:

12. Use the 'Modify CPU Register Contents' command (***RM***) to set the Program Counter to point at address $4000.
13. Use the 'Begin/Continue execution of user code' command (***G***), and press ENTER to start the program execution. The program should run continuously if you correctly created an infinite loop.
14. Change the value on the switch and notice the displayed value on the 7 segment display. If the switch value is not displayed correctly on the 7 segment display, debug your program and repeat the steps to test its execution. Remember that the DIP switch outputs a 'high' in the OFF position and a 'low' (ground) in the ON position.

Note: You can use the **RD** command at any time to display the contents of the CPU registers. You can use the **T** command to trace through your program one instruction at a time and the **BR** command to setup break points.

15. When you are satisfied that the program is operating correctly, record a comment on your observations in the check off sheet. Also comment on whether or not the 7 segment display change to the new value *immediately* after you change the switches or after a brief pause.
16. Ask the TA to check a demonstration of your program. Ask the TA to check off Part 1 on your lab check-off sheet. Remember to print your final *.LST* file and include it in your report.

Part 2: Adding Counter Capability

Note: The last step below optional.

Modify your code from Part 1 to do the following:

- read the value from the 4-bit DIP switch.
- display the value on the 7-segment display.
- starting at the value on the 4-bit DIP switch, count up and display each new value after a 1 second delay.

  Suggestion: Set the higher bits of register where you will store the DIP switch value to '0' (e.g. if you store in accA, set the highest 4 bits of accA to '0') after you store it. This may help in the next step when you compare it with a value.
- when you reach the maximum value for the 7-segment display (1001), have your system count back down to the value on the DIP switch, displaying the value at each count after a 1 sec delay.
- when the value goes back to the value on the DIP switch either
  - stop (SWI), or
  - (**Optional -**if you have time and want to try something a little more interesting), stop when there is no change the DIP switch, but when the DIP value is changed, read the new value and follow the steps above counting up and back again.
    To complete this task, you may want to create an infinite main loop that compares the current DIP switch value with the previous one. If it is same, it stays in the loop and does nothing. Otherwise, it goes to the loop that counts up and down and then returns to the main loop.
    This optional task is for fun and experience. No bonus points will be awarded.

Preparation & Testing:

1. Write the assembly code for the program described above and save it. Compile and remove all syntax errors. Upload your program to the microcontroller and then inspect, test, and debug your hardware/software system until it is functioning correctly.
2. Test your system for different starting values on the DIP switch. Remember to reset the program counter before running the program each time.
3. When you are satisfied that the program is operating correctly, record a statement of your observations in the check off sheet. Be specific enough that your statement verifies that all required features of your system were properly observed.
4. Ask the TA to check a demonstration of your program. Ask the TA to check off Part 2 on your lab check-off sheet. Remember to print your final *.LST* file and include it in your report. Even if you did not complete this task, show your .LST file and comment on what was and was not working in your lab report.

Final Tasks and Notes

- Turn off anything that you have turned on.
- Disconnect and return the microcontroller interface board to the lab closet.
- Tidy up your lab bench

Discussion Points

As explained in the *Lab Report Guide*, you should address these discussion points in a designated section of your report.

1. What is the maximum delay time that can be created using your delay loop structure from Part 1?
2. In Part 1, step 14, you should have observed a delay most of the time you entered a new digital input code before it was displayed on the 7 segment display. Explain why this is likely to occur in the algorithm for Part 2.
3. Consider a program that would turn on LED1 (only) then wait 2 seconds and turn on LED2 (only) then wait 2 seconds and turn on LED3 then wait 2 seconds and turn on LED4. The program would then wait 5 seconds and then flash all LEDs on then off 3 times with a 0.5 second delays between flashes. The program would then wait 5 seconds and start the whole cycle over. Describe this algorithm using pesudocode or a flow chart using ***subroutines*** for all delays and using loops whenever possible to minimize code length.

Subroutines: In case subroutines have not been discussed in class by the time you do this lab, here's the basic concept. A subroutine is a sequence of instructions stored outside the main program loop that can be called upon to perform a specific action (such as a delay). Subroutines are called using a Jump to Subroutine (JSR) instruction, and all subroutines must end with a Return from Subroutine (RTS) instruction that causes the program to return to the point immediately after the JSR. Thus, the main loop would contain only the JSR instruction (typically JSR LABEL, where LABEL is the label that defines the starting point of the subroutine you want to call). When coding, subroutines are generally places after the main program (often stopped at a SWI instruction) but before the END directive. A program can have and call multiple subroutines.

PRE-LAB 6

Due: At the beginning of lab.

Student Name: _____ Lab. Section (time): _____

Make sure you read the lab document before you start the pre-lab, especially the Background section.

1. Write the code for delay subroutine that will create a delay of approximately one second. Briefly explain how you computed this value.

2. Write the all of the code necessary (including port setup) to:
   a. Send a value of your choice to the LEDs
   b. Display a '7' on the seven-segment display. Don't forget the enable on Port M.
   c. Read a 4-bit value from the DIP switch inputs.

3. Write a program segment that will count up from a given value (0 ~ 9) to 9 and then count back to this given value.

4. Plan both programs needed for Part 1 and Part 2 of this lab by creating either a flowchart or pseudocode for the assigned tasks. Show these to the TA before beginning your lab and then attach them to your lab report.

LAB 6 CHECK-OFF SHEET

Student Name: _____ Lab. Section (time): _____

Complete this sheet as you complete the lab. Remember to have the TA check off each section of the assignment. This sheet must be included in your lab report.

Part 1: Using Parallel I/O Ports and the Project Development Board

Step 10. The contents of memory in addresses

$4000_____ $4003_____

$4001_____ $4004_____

$4002_____ $4005_____

Step 11. Comment on verifying program is correctly loaded.

_____

Step 15. Comment on whether the switch value was correctly displayed on the 7 segment display and if it was immediate or not.

_____

_____

Part 1: TA sign off
Part 1: *Using Parallel I/O Ports and the Project Development Board* Initial_____

Part 2: Adding counter capability

Step 3. Statement verifying system is working correctly.

_____

_____

_____

Part 2: TA sign off: Adding counter capability Initial_____

Step 3 (with option). Statement verifying system is working correctly if you do the optional step.

_____

_____

_____

Part 2 (optional step): TA sign off: Adding counter capability with infinite loop Initial_____