**Kettering University**

**Microcomputers I**

Lab Exercise 6

**I/O Programming**

**State Machines in Software**

**Switch Debouncers**

Stopwatch

Spring 2022

**Prelab (10%):** Go over this handout rigorously, do AssignmentsError! Reference source not found. **through** Error! Reference source not found., and 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.

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

**Note:** If you are not in my lecture section (Section 01), please let me know ASAP but before the lab time if anything in this handout has not been yet covered in your lecture.

**Purpose of this lab exercise:** (Please use your own words.)

 We created a timer for the purpose of getting familiar with programming physical hardware and debouncing inputs on said hardware. Additionally, this exercise serves to familiarize us with software-based state machines.

**Assignments**

Prelab ends here.

1. Briefly but clearly explain the reason for each delay block in your flowchart:

 We only have one delay, and it runs at the very start of the loop. This serves two purposes:

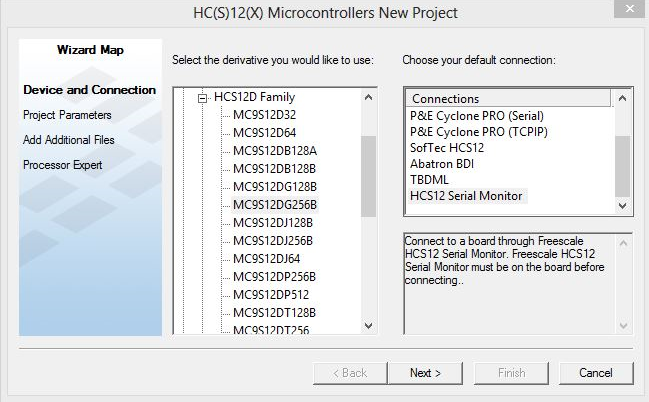
1. This allows us to loop the program 1000 times to count one second rather than delaying the program for one second. (If we delay one second, then no inputs can be accepted during that one second.)
2. This builds in a 1ms debouncer.
3. There is a major difference between how you have done lab assignments so far and how will do them from this week on:

You have *simulated* the execution of instructions so far, while from this week on you will run your programs on *real hardware*, i.e., you will let the *microcontroller* *execute* *instructions*. To make this happen, you need to take Code Warrior to HCS12 Serial Monitor mode as you will see.

Similar to simulation, you can still use CodeWarrior to see the contents of memory and registers after the HCS12 (real hardware) executes an instruction, and also change their contents before HCS12 executes another instruction!

Make a lab06 folder on your desktop. This section walks you through the complete cycle of how to use the trainer board (instead of the simulator), although several steps are identical to what you did for simulation purposes in the past weeks:

Double click on the CodeWarrior icon. A blank window opens up. Go to File > New Project … The New Project window opens up as shown in Figure 1. Under the **HCS12D Family** tab, choose MC9S12DG256B. In the Connections pane, select **HSC12 Serial Monitor**. Click Next.



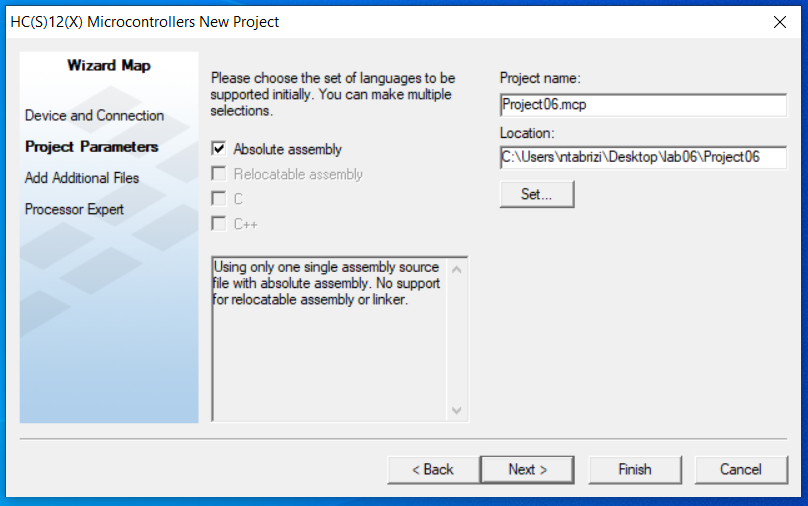
1. **New Project window**

The *second* **New Project** window opens up as shown in Figure 2. Use the **Set …** button to point to folder lab06 in the **Location** field. In the **Project name** field type your (arbitrary but meaningful) project name, say, **Project06.mcp**. Note that your project name is appended to the **Location** field. Uncheck all the checkboxes and then check the **Absolute assembly** checkbox. Click on **Finish** (NOT Next!). CodeWarrior’s main window opens up as shown in Figure 3.

Here is your project file:

lab06\Project06\Project06.mcp.

To open the project later, you should open this file. (You may need to change the file selection filter.)

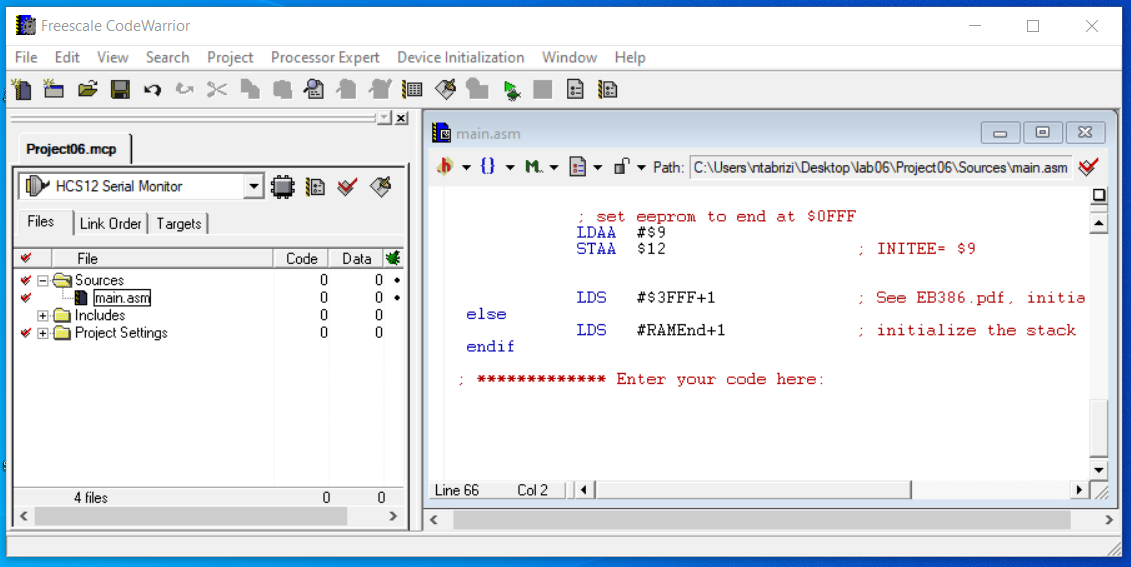


1. **Second New Project window**

Open folder lab06\Project06\Sources and replace file main.asm with the same-name file main.asm that is posted on Bb. Back to CodeWarrior main window and in the file hierarchy on the left pane, double click on main.asm (under Sources) to open the file.

1. Translate the flowchart that you completed in **Error! Reference source not found.** into assembly language, and using the built-in editor of CodeWarrior, type your program in main.asm and save it.

Select “HCS12 Serial Monitor” from the dropdown list on the left pane of the CodeWarrior main window. See Figure 3. Connect the development board to your desktop as well as the power supply if it is not already connected. Press the reset button on the trainer board before you upload your program



1. **CodeWarrior main window**

**Close the** “**True-Time Simulator & Real-Time Debugger”** **window(s), if any,** and then hit the green debug arrow (see Figure 3) to enter the debug mode. When the Monitor Setup window pops up as shown in Figure 4, click OK. The “True-Time Simulator & Real-Time Debugger” window opens up after you have fixed possible syntax errors. This way your machine code is uploaded into the microcontroller.

The first instruction of your code is now located in memory @ address $400E. You may still use F11 to execute one instruction at a time. Remember that when the microcontroller executes an instruction, you may check the registers and memory locations affected by that instruction to see if your exception matches your observation. CodeWarrior also allows you to change the contents of memory or registers before an instruction is executed!

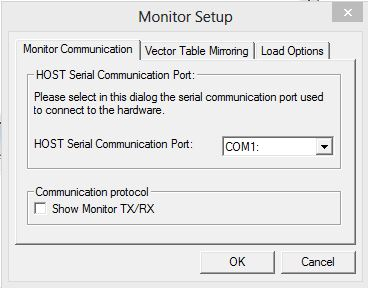
**Remember from previous weeks**:

F5 is the run command. You may also reach the run command through the menu bar on the “True-Time Simulator & Real-Time Debugger” window:

Run > Start/Continue (F5)

To force the CPU to stop at the end of your program (when you use the Run command) just put the following instruction at the end of your program:

here: bra here

****

1. **Monitor Setup window**

Or you may replace the above bra instruction with any (arbitrary) instruction, and then put a *breakpoint* on it. The CPU will stop instruction execution as soon as it hits a breakpoint. To put a breakpoint on an instruction, go to the assembly pane, use the mouse pointer to point to the instruction, and right-click. In the popup menu, select “Set Breakpoint”. To remove a breakpoint, select “Delete Breakpoint” on the same popup menu.

1. Test your stopwatch, and then **show it to the lab instructor.**
2. Copy your *indented* and *commented* assembly code and paste it into the space provided below:

 ; \*\*\*\*\*\*\*\*\*\*\*\*\* Enter your code here:

 ; Initialization

  bset DDRJ, $2     ; configure pin PJ1 as output

  bclr PTJ,  $2     ; enable LEDs to light up

  movb #$FF, DDRB   ; configure port B as output

  movb #0,   PORTB  ; turn LEDs OFF

  movb #$F,  DDRP   ; configure port P3-0 as output

  movb #$0F, PTP    ; turn 7-segment displays OFF

  movb #0,   DDRH   ; configure PORT H as input

  ldd #0

 ; note: REG-A stores the state and REG-B is the content of the timer

 ; infinite loop

 endCase:              ; when endCase is reached, we start the loop again

 stab PORTB

 jsr wait1ms

 ; switch statement evaluates current state

    cmpa #0

     beq case0             ; case 0: timer is stopped and can be reset

    cmpa #1

     beq case1             ; case 1: timer is stopped, but button is being pushed to start timer

    cmpa #2

     beq case2             ; case 2: timer is active until button is pushed

    cmpa #3

     beq case3             ; case 3: button is being pushed to stop timer

    bra case0              ; any case outside our range evaluates to 0

    case0:

       brset PTH, 2, noReset     ; evaluate timer button iff reset button is not pushed

          ldab #0                 ; reset timer

          bra endCase

       noReset:

          brset PTH, 1, endCase   ; if timer button is not pushed, loop

          ldaa #1                 ; else, change state to 1

          bra endCase

    case1:

       brclr PTH, 1, endCase      ; if timer button is being held, loop

       ldx #1000                 ; else, set X to 1000 and change state to 2

       ldaa #2

       bra endCase

    case2:

       brset PTH, 1, count        ; if timer button is pushed, change state to 3

          ldaa #3

          bra endCase

       count:                     ; else, count

          dbne X, endCase        ; if X != 0, loop

          ldx #1000               ; else, set X to 1000 and increment the timer

          incb

          bra endCase

    case3:

       brclr PTH, 1, endCase      ; if timer button is being held, loop

       ldaa #0                    ; else, change state to 0

       bra endCase

 wait1ms:

    pshy

    ldy   #8000

 loop:                      ; 1 ms delay

    dbne   Y, loop

    puly

    rts

    end