OREGON INSTITUTE OF TECHNOLOGY

**Computer Systems Engineering Technology Department**

***CST 204 - Introduction to Microcontrollers***

**Lab 2 – LCD Display and Digital Clock App v2**

This lab has two parts.

* Part 1 will implement the timing signals to control the LCD (liquid crystal display) by bit bashing the RS, RW’, and E signals. The Data Bus signals will be written and read with respect to these timing signals.
* Part 2 will be a digital clock display application. It will execute under the real-time executive with the 10 ms loop time.

All of the Lab 2 functions will build off of the Lab 1 framework.

* The LED HEARTBEAT application that was created in Lab 1 will continue to execute concurrently with the new Lab 2 function.
* Again, the concurrency comes through the round-robin execution of the “apps” or tasks.
* Each app should limit its execution time so that the sum total of all execution times does not exceed the 10 ms loop time.

Lab 2 procedure.

* Using Windows File Explorer, copy the **lab1.X** project folder into a NEW **cst204:\labs\lab2\** folder. All of the files should go over verbatim.
* Run MPLAB X and open the **cst204:\labs\lab2\lab1.X** project. **Within the Projects Tab**, right-click rename the **lab1** project to **lab2**, and, check the “Also Rename Project Folder” box. This will result in **cst204:\labs\lab2\lab2.X**. It is not enough to just copy the files.
* The migrated files from **lab1** should be:
  + Within the **cst204:\labs\lab2\lab2.X\source** folder (Added to **Source Files** Project Folder)
    - app\_heartbeat.S
    - config\_bits.S
    - data.S
    - hardware.S
    - main.S
  + Within the **cst204:\labs\lab2\lab2.X\library** folder (Added to **Libraries** Project Folder)
    - lab1\_libs.a

(Note: Library directory path must have been specified under File ⇒ Project Properties (lab2) ⇒ Categories: XC32 (Global Options) ⇒ xc32-ld ⇒ Option categories: Libraries ⇒ Library directories)

* + If for some reason copying does not work out, create a new **lab2.X** project and manually copy the files from the **lab1.X** project and also add them to the new project/
* Modify the following files to add the required items.
  + **hardware.S** will add the assembly code to configure the GPIO pins that interface to the LCD.
    - **gpio\_config** – Add configurations to the *existing function* for the following pins using the following sequence:

1) Establish default data value = 0 (**PORTxCLR** SFR),

2) Establish NON-open-drain configuration (**ODCxCLR** SFR)

3) Establish digital output configuration (**TRISxCLR** SFR)

Remember, use the three steps to writing a SFR: load address of SFR, load or move data to a register, store the register value to the SFR (**la,li,sw**).

* + - * **RB15** – Used as LCD **RS** signal
      * **RD5** – Used as LCD **RW’** signal



* + - * **RD4** – Used as LCD **E** signal
    - **instr\_busy\_wait** – Write this *new function* to loop four (4) instructions including the nop in the delay slot. **a0** will contain the number of loops. Since the instruction time is 50 ns (20 MHz CPU clock), each loop is assumed to wait 200 ns. **a0** will be passed in appropriately.
    - **lcd\_config** – Write this *new function* to configure the LCD module. It calls a sequence of *new functions* that are defined below in a new file **lcd.S**. This call will be added to the **main.S** real-time executive framework configurations. Since this function calls other functions, it must implement a C-compatible stack frame. The call sequence is as follows:
      * Call **instr\_busy\_wait** for 40 ms. Set **a0** appropriately.
      * Call **lcd\_write** to send the command **0x38** (**Function set – 8-bit interface, 2-line display, 5x8 font**). Set **a0 = 0** (indicating a COMMAND write) and **a1 = 0x38** (command value for Function set).
      * Call **lcd\_wait\_for\_not\_busy** to wait for previous command to complete. This is a busy wait on the **BF** so no arguments are needed.
      * Call **lcd\_write** to send the command **0x06** (**Entry mode set – Increment address, No shift**).
      * Call **lcd\_wait\_for\_not\_busy**.
      * Call **lcd\_write** to send the command **0x0f** (**Display mode set – On, Cursor on, Cursor blink**).
      * Call **lcd\_wait\_for\_not\_busy**.
      * Call **lcd\_write** to send the command **0x80** (**Set address = 0**).
      * Call **lcd\_wait\_for\_not\_busy**.
      * Call **lcd\_write** to send the command **0x01** (**Clear display**).
* Create a new file **lcd.S** within the **cst204:\labs\lab2\lab2.X\source** folder and add it to the **Source Files** Project Folder.
  + **lcd.S** will contain the assembly code to *bit bash* the LCD timing control signals. There will be four (4) assembly functions to implement the LCD “driver”. Each is specified by a required name and operation.
    - **lcd\_read** – Read either the BUSY FLAG/ADDRESS COUNTER value or DATA value at current ADDRESS COUNTER value.

**a0** = 0 (BF/ADDR) or 1 (DDRAM Data)

**v0** = BF/ADDR or Data value (return value)

1. Set RS (**PORTB**<15>)

If **a0** = 0, then **PORTBCLR**<15>, else, **PORTBSET**<15>

1. RW’ = 1 (**PORTDSET**<5>)
2. E = 1 (**PORTDSET**<4>)
3. Delay 9 **nop** instructions (E pulse width timing requirement)

[20 MHz clock = 50 ns; 50 \* 9 = 450 ns pulse width for E]

1. Read DB7-DB0 into v0 (**PORTE**<7-0>)



Mask **v0** to a byte (0xff) for return

1. E = 0 (**PORTDCLR**<4>)
   * + **lcd\_write** – Write a value to either the INSTRUCTION REGISTER or DATA REGISTER.

**a0** = 0 (Command) or 1 (Data)

**a1** = Command or Data value

1. Set the **RE7-RE0** latch value to the **a1** input value (**PORTE**<7-0>)

Must first **PORTECLR**<7-0>, THEN **PORTESET** the **a1** input value.

1. Set RS (**PORTB**<15>)

If **a0** = 0, then **PORTBCLR**<15>, else, **PORTBSET**<15>

1. RW’ = 0 (**PORTDCLR**<5>)
2. E = 1 (**PORTDSET**<4>)
3. Drive RE7-RE0 port pins to OUTPUT (**TRISECLR**<7-0>)
4. Delay 9 **nop** instructions (E pulse width timing requirement)
5. E = 0 (**PORTDCLR**<4>)
6. Float RE7-RE0 port pins to INPUT (**TRISESET**<7-0>)
   * + **lcd\_wait\_for\_not\_busy** – Polls the BUSY FLAG (**BF**) using **lcd\_read** until **BF** clears.

Poll the BUSY FLAG calling the **lcd\_read** function until BF = 0. BF is Bit 7 and should be masked in order to check.

* + - **lcd\_write\_string** – Uses **lcd\_write** to send characters in a string to the LCD until the null terminator is read.

**a0** = Address of string

Read through a string sending characters to the LCD by calling **lcd\_write** until the null terminator is read. BF must be checked by calling **lcd\_wait\_for\_not\_busy** before each call to **lcd\_write**.

Completion of Part 1.

* Download and add the **app\_test\_lcd.c** to the **cst204:\labs\lab2\lab2.X\source** folder and add it to the **Source Files** Project Folder. This will alternate two lines of text and clear screens every second. Use this as a model to create a state machine in C.
* Modify **data.S** to declare and initialize the global variables **lcd\_test\_state = 0** and **lcd\_test\_count = 0**. Also declare the global **constant** strings [in **.text** segment] **LCD\_TEST\_LINE1[20] = ”This is line 1”** and **LCD\_TEST\_LINE2[20] = ”This is line 2”**.
* Modify **main.S** to include the **lcd\_config** and **app\_test\_lcd** calls to the real-time executive. The call to **app\_heartbeat** remains.
* Demonstrate operation to the instructor.
* Zip the **cst204:\labs\lab2** folder and rename it **cst204:\labs\lab2part1.zip** and upload.

Description of Part 2.

* Implement a digital clock function to be **displayed on Line 1 of the LCD**. The display should indicate:

**Time: 10:10:45**



Address 13

Address 10



Address 07



Address 00



NOTE: Use this format exactly. If your function appears to vary during demonstration, you will be instructed to change it.

* Create a new file **app\_clock.c** within the **cst204:\labs\lab2\lab2.X\source** folder and add it to the **Source Files** Project Folder.
  + **void app\_clock(void)** will implement a state machine to keep track of global variables **clock\_second**, **clock\_minute**, and **clock\_hour**. Define these global variables in the **data.S** file. At the appropriate time, it will write the individual TWO CHARACTER fields ONLY for the **clock\_second**, **clock\_minute**, and **clock\_hour** values. It will NOT write the entire line of text. The hour 3 should be output into the hour field as “ **3**” (with a space) and NOT “**03**”. Use the **app\_test\_lcd** app implementation as an example state machine model.

**Update Clock Seconds LCD field**

(a)

0

1

2

3

**clock\_count ≠ 100**

**clock\_count = 100 &**

**clock\_second < 59**

**clock\_second++**

**clock\_state = 1**

(a)

(b)

**Update Clock Minutes LCD field**

**clock\_count = 100 &**

**clock\_second ≥ 59 &**

**clock\_minute < 59**

**clock\_second = 0**

**clock\_minute++**

**clock\_state = 2**

(b)

(c)

(d)

**clock\_count = 100 &**

**clock\_second ≥ 59 &**

**clock\_minute ≥ 59 &**

**clock\_hour ≥ 12**

**clock\_second = 0**

**clock\_minute = 0**

**clock\_hour = 1**

**clock\_state = 3**

(d)

**clock\_count = 100 &**

**clock\_second ≥ 59 &**

**clock\_minute ≥ 59 &**

**clock\_hour < 12**

**clock\_second = 0**

**clock\_minute = 0**

**clock\_hour++**

**clock\_state = 3**

(c)

**Update Clock Hours LCD field**

Note that 0 ≤ Seconds ≤ 59, 0 ≤ Minutes ≤ 59, and 1 ≤ Hours ≤ 12. Therefore, the increment process will reset the value to the minimum once the maximum is reached. Also, only ONE arc (state) transition is accomplished per 10 ms execution loop. Again, refer to the **app\_test\_lcd.c** source file as a model for the state machine.

Modify the **data.S** file declare and initialize the **clock\_count** and **clock\_state** global variables.

* + **void lcd\_time\_init(void)** will write out the time template using the initial value of the **clock\_second**, **clock\_minute**, and **clock\_hour** global variables. There will be a single **lcd\_time\_init** call from the configuration section in main.S. In this function, use a global CONSTANT string **TIME\_INIT[]** and place the **clock\_second**, **clock\_minute**, and **clock\_hour** that matches the Time display noted above. Use the initial clock variable values for the time. This constant string is defined in the **data.S** file.
  + **void clockfield\_to\_lcdstr(int clockfield, int field\_type)** is a helper function that will convert a binary number between 0 and 59 (***clockfield***) and convert it to a two ASCII characters and place them in global string buffer **TIME\_INIT[]** at locations ***field\_type*** and ***field\_type + 1***. Field\_type is the address of hour, minute or second. (7, 10 or 13). [Also note that, if field\_type is 7 (***hour***) and if the upper digit is 0, then store BLANK character at location field\_type.] The buffer is defined in the **data.S** file. The string must be null terminated.

All functions and variables defined and implemented elsewhere must be declared and prototyped “**extern**”.

If you find it helpful, you may use standard C library calls for string manipulation.

Completion of Part 2.

* Modify **main.S** to include the **lcd\_time\_init** call in the configuration section and **app\_clock** call in the task section of the real-time executive. You will need to remove the executive call to **app\_test\_lcd.**
* Demonstrate operation to the instructor.
* Zip the **cst204:\labs\lab2** folder and rename it **cst204:\labs\lab2part2.zip** and upload.