OREGON INSTITUTE OF TECHNOLOGY

**Computer Systems Engineering Technology Department**

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

**Lab 1 – Real-time Executive and Ports**



**Real-time Executive Definition**

All projects will be implemented using a *real-time executive* (RTE). This is a method to have real-time behavior without a real-time operating systems (RTOS).

* RTE is implemented as assembly code in **main.S**. It consists of three (3) sections.



* + {1} Hardware and other configurations are performed. These functions execute one time.
  + A **while(1)** loop.
    - {2} A global flag variable **FLAG\_10MS** will be used to “gate” the 10 ms execution period. This is a *busy wait* until the start of the next 10 ms period. Note that this flag is synchronized with a timer peripheral.



* + - {3} Function calls to “tasks” or “applications” that occur once per 10 ms loop.
* Code Framework:



**main:**

**// stack frame creation**



**// goes here**

**jal config\_one**

**nop**

**{1}**

**jal config\_two**



**nop**

**jal config\_etc**

**nop**

**while\_1:**



**wait\_for\_flag\_10ms:**

**la t1, FLAG\_10MS**

**lw t0, 0(t1)**

**beq t0, zero, wait\_for\_flag\_10ms**



**{2}**

**nop**



**found\_flag\_10ms:**

**la t1, FLAG\_10MS**



**sw zero, 0(t1)**

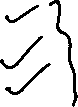
**perform\_tasks:**

**jal task\_one**

**nop**

**{3}**

**jal task\_two**



**nop**

**jal task\_etc**

**nop**

**j while\_1**

**nop**

**// stack frame removal and function return**



**CST 204 Coding Ground Rules**

* + - 1. These rules apply to all labs, not only Lab 1.
      2. The top of every source file will contain comment code with the student’s name.



* + - 1. Assembly functions *other than* **main** will only use the following registers:
         1. **s**-registers; which must be saved and restored via a stack frame
         2. **v0** and **v1**; **v0** must be used for any return value



* + - * 1. **a0** through **a3**; assumed to be C-compatible arguments



* + - * 1. Only the **main** function may use **t0** through **t9**.



* + - 1. A function’s purpose will determine its code implementation
         1. Functions that directly access PIC32 SFRs *will be implemented in assembly language*. You can think of these functions as “drivers” or hardware abstractions.
         2. Functions that do not directly access PIC32 SFRs *will be implemented in C*.
         3. The **main** function will be implemented in assembly language.



**main** will implement a stack frame that handles nested calls.



Stack frame will contain locations for **ra**, **fp**, **a3-a0**.

At **main**’s end, the stack frame is removed and the return is coded. (Note that normal operation of the real-time executive prevents **main**’s return.)

* + - * 1. An assembly function that does not nest is *not required* to implement a stack frame. This is against C convention but will suffice in this class.

However, if **s**-registers are used, it is required to save and restore values in a stack frame.



TIP: You can avoid using a stack frame within a function by only using **v0-v1**, **a3-a0** as temporary registers.

**Create lab1 Project**

* Create a **z:\CST204\labs** Folder on your z: drive.
* Create a **z:\CST204\labs\lab1** Folder.
* Create a new “**lab1**” project (**lab1.X** folder) in the **z:\CST204\labs\lab1** Folder.
  + **Choose Project – Microchip Embedded – Standalone Project**
  + **Select Device – Device – PIC32MX460F512L**



* + **Select Tool – Hardware Tools – ICD 3**



* + **Select Compiler – Compiler Toolchains – XC32 – XC32 (v n.nn)**



* + **Select Project Name and Folder**

**– Project Name: lab1**



**– Project Location: z:\CST204\labs\lab1**

**– Project Location: z:\CST204\labs\lab1\lab1.X**

**– Check box “Set as main project”**



* Create a “**source**” file folder in the **lab1.X** project folder.
* Create a “**library**” file folder in the **lab1.X** project folder.



* Download and save the **lab1\_libs.X.a** file in the new **library** folder (**z:\CST204\labs\lab1\lab1.X\library\lab1\_libs.X.a**).

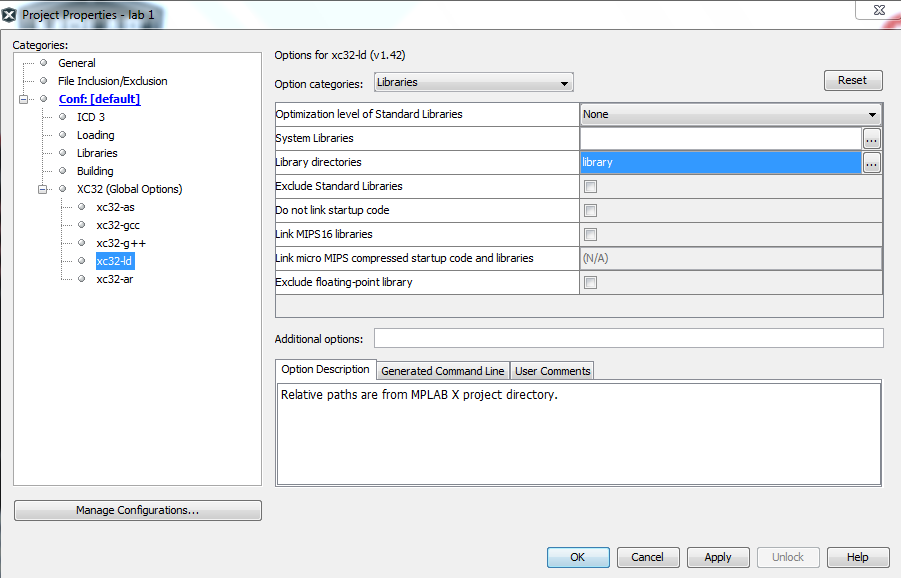


* In **lab1 Projects Window**, right-click “**Libraries**” Project Folder and add the **z:\CST204\labs\lab1\lab1.X\library\lab1\_libs.X.a** Library File to the project.





* In **Project Properties**, Browse and add the **z:\CST204\labs\lab1\lab1.X\library** path in the **xc32-ld – Library directories** option box





Note: The following **Configuration functions** provided to you in the ***library file*** **lab1\_libs.X.a** are:



**intsys\_config**

Configure the CPU interrupt system. Timer 1 interrupt is used.



**timer1\_config**

Configure Timer 1. This is provided for you right now.

These functions are provided to you right now. However, you will eventually implement them yourself when the interrupts are covered.

These configuration functions will be called by **main.S**.

**Create Source Files**

**data.S** assembly source file – This file will define the required global variables. It does not contain any executable code and is added to the Project Source Files folder. The global variables needed for Lab 1 are:

**FLAG\_10MS: .word 0**

**heartbeat\_state: .word 0**

**heartbeat\_count: .word 0**

Note: This file for Lab 1 is provided in the Canvas Lab Assignment.

* It is to be used as a template for all remaining lab projects.
* **FLAG\_10MS** is used by the real-time executive in **main.S**.
* **heartbeat\_state** and **heartbeat\_count** are used by the “**app\_heartbeat**” application defined below.

**config\_bits.S** assembly source file – The Configuration Bits in registers **DEVCFG1** and **DEVCFG2** need to be configured to *setup the clock frequency* and also *disable the Watchdog Timer*. It does not contain any executable code and is added to the Project Source Files folder.

These configuration word values can be obtained from the **Window – Target Memory Views – Configuration Bits** debug window. In this window, you select the needed parameter values and it will indicate the Device Configuration Bits Register word value. Note that the *Physical Address* is given in the **Configuration Bits** debug window but the code needs to refer to the *Virtual Address*, i.e. **0x1FC02FF4** versus **0xBFC02FF4**, for register **DEVCFG2**.



Using the **Configuration Bits** debug window, determine the configuration word values and enter the code as shown below, substituting the **0xNNNNNNNN** values with the configuration words.

**// For DEVCFG2**

**// FPLLIDIV = DIV\_2, FPLLMUL = MUL\_20, and FPLLODIV = DIV\_4**

**.section .config\_BFC02FF4, code**

**.word 0xNNNNNNNN**

**// For DEVCFG1**

**// Set FNOSC = PRIPLL, POSCMOD = XT, and FWDTEN = OFF**

**.section .config\_BFC02FF8, code**

**.word 0xNNNNNNNN**

**hardware.S** assembly source file – This file will call various functions that perform hardware and system configurations. These configuration functions will then be called by **main.S**. The individual functions will use the SFR labels with the “read-modify-write” modifiers, e.g. **PORTACLR**, **PORTASET**, **ODCACLR**, etc. The file is added to the Project Source Files folder.

Assembly configuration functions to write:

**waitstates\_config**

Description: Configure SYSCLK with 0 wait states. This done by writing the **CHECON** register. You will need to look at the PIC32MX Reference Manual (Section 4) and the Datasheet (Section 9) for the **CHECON** definition and configure the appropriate bits.

**gpio\_config**

Configure **RA7** (Port A, Bit 7) to default value 0; non-open drain; digital output. This is done by writing Bit 7 of the **PORTA**, **ODCA**, and **TRISA** registers. Again, the PIC32MX Datasheet and Reference Manual (Section 12) define these registers.

**app\_heartbeat.S** assembly source file – This will blink the LED on the Explorer 16 Development Target Board. Assembly “task” or “application” functions to write:

**app\_heartbeat**

Blink the LED connected to Port A, bit 7 (RA7), every second with a specific cadence. This will be implemented with a state machine that transitions every 10 ms loop.

Basic algorithm:

A) Modify input conditions

1. Increase variable **heartbeat\_count**. This count cycles between 0 and 99, yielding 100 periods. Those 100 periods at 10 ms per period yields 1 second.

B) Perform state transition

1. Check current state using the **heartbeat\_state** global variable. Process the actions according to the state diagram below. You can do this in a “switch” manner.
2. Update the LED at RA7 at the appropriate transitions.
   1. Writing to SFRs
   2. Use **PORTACLR** or **PORTASET** with the BIT YOU NEED TO CONTROL set to 1 or 0; 1 means either CLEAR the bit or SET the bit. 0 means do not modify the bit.

**heartbeat \_count = 25 /**

**RA7 = 0**

**heartbeat \_state = 0**

**heartbeat\_count = 0 /**

**RA7 = 1**

**heartbeat \_state = 1**

0

1

**heartbeat\_count != 0**

**heartbeat\_count != 25**

C) Exit

1. App or task will always exit once state processing is performed. Only one state transition is performed per RTE 10 ms loop.

**main.S** assembly source file – This will implement the real-time executive as defined at the beginning of this Lab Description.

Configurations to be called:

* **waitstates\_config**
* **intsys\_config**
* **timer1\_config**
* **gpio\_config**

Tasks or applications to be called:

* **app\_heartbeat**

Completion of Lab 1.

* Demonstrate operation to the instructor.
* Zip the **cst204:\labs\lab1** folder and rename it **cst204:\labs\lab1.zip** and upload.