ECE 3720 – Microcomputer Interfacing Laboratory

Pre-labs for ECE 3720
Created by Brad Green 8/30/15
Last Updated:07/17/17
Introduction

Objective:
• Gain a better understanding of the functionality of a microcontroller.
• Learn to interface devices with different modules on a microcontroller.

Experiments:
10 experiments involving various modules and a final design project.
Final Design Project

• Will combine either previous labs or new concepts in creative and applicable design
• Worth 30% of final grade
• Weekly Lab Reports for reference when making final design
• Final lab report will be given on final design project
Equipments

• The National Instruments ELVIS II system.

• Microcontroller – chipKIT Cmod (Microchip PIC32MX150F128D).
MPLAB X IDE.

- Text Editor for writing code.
- Cross compiler to generate executables.
- chipKIT PGM to load the executable on the microcontroller.
Prerequisite

• Familiarity with C programming language.
Cmod Pin Layout

• Refer to Reference Manual!
  • The DIP Pin # is the pin on the CMOD device
Laboratory 1 – Introduction to the PIC32MX150F128D
Laboratory 1 Program

• Program will count from 0-15 in binary using LEDs.
• Download the program from our section on Blackboard under the “information” section
• Extract the program to your profile.
• Open MPLAB X IDE and open this folder
  • Code is under source files and called main.c
• Take a few minutes to read the code and figure out what it should do
Laboratory 1 Program

• Plug the Microcontroller USB into the Computer

• Attempt to compile the code (click the Hammer)
  • Any warnings stating libraries are out of date should be ignored.
  • Just look for “Build Successful” in green text.
Laboratory 1 Wiring

V_{cc} (5V) → LED 0 → LED 1 → LED 2 → LED 3 → GND
Lab 1 Compiling/Programming

• When your wiring is complete and you have the MC USB plugged into the computer, turn on the NI-ELVIS II
  • There are two switches, one in the back and one on the top of the board.

• In MPLAB X, click the Make and Program Device Button.
  • This should program the MC, ask if there are problems.
  • This could take a while if there are updates for the Microcontroller.
Lab 1 Modification

• Now that the program is running, modify it to count backwards.
  • Start at 15 and go down to 0 then repeat

• Modify the “delay” to take a longer/shorter amount of time and demonstrate this.
Final Reminders!

• Store each lab as a separate file on your account on the computer you sit at (copying an old lab folder and renaming it for new labs).

• This ensures all necessary files are linked and provides your access to your previous lab code when you begin your final design project.

• Bring USB Flash drive or Save files to Box/Dropbox/etc. to ensure you do not lose data.
Laboratory 2 – Application of a Digital Latch
Creating New Project

• Open MPLAB X
• Select New Project
• Select “Standalone Project” then next
Creating New Project

• Select the “32-bit MCUs” category
Creating New Project

• Select the “PIC32MX150F128D” Device then next
Creating New Project

• Press next on the following screen.
• Then select the XC32 compiler
Creating New Project

• Name and save the project wherever you would like.
• Create a new “C Source File” in sources
  • Right Click Sources
    • Hover on “New”
    • Select C Source File
    • Name it what you like

• The header you need is plib.h
  • #include <plib.h> REQUIRED FOR ALL
Lab 2 Preparation

• This lab combines hardware with software

• Uses 74LS373 D-Latch
  • From Quiz

• Use both the reference manual provided to you (chipkit_cmod_rm.pdf) and the datasheet for the D-Latch you’ve found to accomplish the task
74LS373 – D Latch

• Look up Data Sheet!

• You may use the internet for any and all projects
Data Direction

• To use input and output registers on the microcontroller, must designate them for input or output in code.

• TRISx—Tri-State Register
  • Used to select the Data Direction for a specific port
  • 0 for output, 1 for input, set bitwise

• Unless the pins is a power source or ground source pin, if you use a pin you must specify the direction of that pin
Xbits Functionality

• Neat Functionality of MPLAB X for this MC
  • Typing “bits” after ANY register name (so far) references a structure and then typing a period brings up the menu shown below

  • From here, you can select singular pins by name.
Reading/Writing Pins

• LATx is used to write
  • This signifies a “latch” used as a buffer before writing to the output pins.
  • LATB = 0x01; //Or LATBbits.LATB0 = 1;

• PORTx is used to read
  • Reads the input pins value
  • A read to an output pin returns the value stored in the “latch” from above
  • y = PORTBbits.RB1;
Push Buttons

• Push Buttons are Low Active!
  • This is mostly because of how you will wire them
  • THE RESISTOR IS IMPORTANT!
    • Don’t break your board by forgetting it

![Circuit Diagram]

Button “Output”
DIO

• DIO is the digital I/O on the NI-ELVIS II board
• We will use this as switch toggles
• You can open the DIO by opening the NI Instrument Launcher and selecting “Digital Writer”
• This outputs a 5V signal!!!
Note about 5V Tolerance

• MOST PINS ARE ONLY 3.3V TOLERANT!!!
  • Keep this in mind when wiring and use reference manual for 5V pins!

• If you wire a rail to the 3.3 V output on the PIC32 (2nd pin), you can use 3.3V to power devices and as inputs to the MC on Push Buttons, ICs, and other equipment

• BE VERY CAREFUL USING THIS PIN!! SHORTING IT OR APPLYING ANY VOLTAGE CAN DESTROY THE PIC32 POWER SUPPLY, RENDERING THE ENTIRE MICROCONTROLLER USELESS
Lab Goals

• Design a circuit that sends 2 inputs to the microcontroller from DIO

• The MC will read these inputs and send them to a D-Latch and to LEDs

• The Latch should send corresponding outputs to 2 more LEDs

• Use a Push Button to send a signal to the MC
  • Output a High signal when the button is pressed

• **BE CAREFUL WITH 5V TOLERANT PINS**
  • Check the reference manual
  • CHECK WITH YOUR TA BEFORE POWERING YOUR CIRCUIT
Crude Block Diagram (Expect this as the norm)

DO NOT FORGET GROUND, POWER, AND RESISTORS! THESE ARE NOT SHOWN HERE
Result

• You should see 2 LEDs mirror the input values

• You should see the other 2 LEDs from the latch mirror the input LEDs only when the button is pressed.
Preparation for Next Week

• Look up the datasheet for a 10k trimpot (trimming potentiometer)
  • Need to familiarize yourself with how it works and how to use it
  • Will be on next week’s quiz

• It’s the piece in your kit that looks like this!
Lab 3 – Comparator
Analog Select

• There are some “Analog Capable” pins on your MC, and these pins will operate as ANALOG INPUTS by default
  • This means they do not know what a “Logic High” is

• You can choose whether a pin is digital or analog with ANSELx
  • This is the Analog Select register
  • 0 means digital, 1 means analog
  • Pins are Analog by default
Determining Analog Pins
Lab 3 Prep

• Uses 10k Potentiometer
  • Hardware

• Uses comparator
  • Software
Comparator Control Register

- Refer to page 211 in the Datasheet
  - Wiring schematic for the internal comparators of the PIC32

- Page 212 Shows how to use the CMXCON register
  - Used to control the behavior of the above mentioned comparators

- We do not have access to the comparator outputs by default so we need a different way to see the results.
Comparator Status Register

- CMSTAT – Only 1 for all 3 comparators
- Holds status for all comparators!

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Bit 31/23/15/7</th>
<th>Bit 30/22/14/6</th>
<th>Bit 29/21/13/5</th>
<th>Bit 28/20/12/4</th>
<th>Bit 27/19/11/3</th>
<th>Bit 26/18/10/2</th>
<th>Bit 25/17/9/1</th>
<th>Bit 24/16/8/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>23:16</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>15:8</td>
<td>U-0</td>
<td>U-0</td>
<td>R/W-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td>7:0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>R-0</td>
<td>R-0</td>
<td>R-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparator Control Register (details)

• We will use the comparator 3 different ways
  1. $I_{V_{\text{ref}}}$ as the inverting input and $C1\text{INA}$ as the non-inverting input
  2. $C2\text{IND}$ as the inverting input and $C_{\text{V}_{\text{ref}}}$ as the non-inverting input
  3. $C3\text{IND}$ as the inverting input and $C3\text{INA}$ as the non-inverting input

• How do these modes differ?

Note 1: Internally connected. See Section 23.0 “Comparator Voltage Reference ($C_{\text{VREF}}$)” for more information.

2: Internal precision voltage reference (1.2V).
Using Comparator Voltage Reference

• Refer to Page 216 for table and formulae

• CVRCON is the register of interest
  • Need to correctly set the following bits to achieve a $CV_{ref}$ value of $0.75*V_{ref}$
    • CVRSS
    • CVRR
    • CVR
  • We want to achieve a value of $0.75*V_{ref}$ using this register
Goal

• To have 3 separate comparators turn on 3 LEDs sequentially with a single potentiometer
  • 1 comparator turns on at $I_{V_{\text{ref}}}$
  • The next turns on at $C_{V_{\text{ref}}}$
  • The last turns on at $C_{3\text{INA}}$

• As you rotate the potentiometer CW, you should see the LEDs light up in order and stay on until you rotate CCW and lower the voltage.
Crude Block Diagram (Not Drawn Accurately)

DO NOT FORGET GROUND, POWER, AND RESISTORS!
THESE ARE NOT SHOWN HERE
Lab 4

Keypad Interfacing
Understanding a Keypad

Keypad Figure 5.5

Use as Outputs

Use as Inputs
Pulling Resistors

• Used here on the input pins of the keypad

• Maybe internal to the PIC32 or external with addition of resistors to the hardware
Pulling Resistors

• What would you use as $V_{in}$?

• Pull up is shown to the right, what changes for a pull down?
Pulling Registers

- Change Notification (CNENX)
  - Can be used for external interrupts, used here for weak pulling registers

- CNPUx
  - Pull Up Enable

- CNPDx
  - Pull Down Enable

- Both are useful, which one is required today?
Wiring Guide

• You must wire your keypad as listed below, else it **WILL NOT WORK** with the masks we give you

Keypad 0 → Port C 1
Keypad 1 → Port C 2
Keypad 2 → Port C 4
Keypad 3 → Port C 3
Keypad 4 → Port C 5
Keypad 5 → Port C 6
Keypad 6 → Port C 7
Keypad 7 → Port C 0

• All wiring should be done with the brass facing you, and pin 0 will be the far right pin.
Masks

• Using the below masks will make your work much simpler in the code
• unsigned char mask[16]={0xEE,0xDE,0xBE,0x7E,
  0xED,0xDD,0xBD,0x7D,
  0xEB,0xDB,0xBB,0x7B,
  0xE7,0xD7,0xB7,0x77};
• unsigned char key[16]={1, 2, 3, 0xA,
  4, 5, 6, 0xB,
  7, 8, 9, 0xC,
  14, 0, 15, 0xD};
Today’s Goal

• When you push a button on the keypad, have the MC light up the corresponding number in binary on the LEDs
  • Assume 0-15, so when pushing the “0th” button, no lights will come on

• Make sure to use the masks!

• As a hint, you will need to loop over the mask when “scanning”
Crude Block Diagram (Not Drawn Accurately)

DO NOT FORGET GROUND, POWER, AND RESISTORS!
THESE ARE NOT SHOWN HERE
What you should have learned...

• How Pull Up resistors work.
• How to use the CNPUEX register correctly
• How to interface with a keypad
Lab 5

Interrupts
What is an Interrupt??

• Think about what “interrupting” means in everyday life

• Interrupts on an MC are not much different
  • Halt main work (call this work ‘a’)
  • Complete work ‘b’
  • Go back to work ‘a’ as if nothing happened
Understanding Interrupt Implementation

• Interrupts are handled automatically by the microcontroller
  • THIS MEANS YOU WILL NOT EXPLICITLY TELL YOUR MC TO RUN INTERRUPT CODE

• To further clarify, you WILL NOT write any code that checks the value of a pin

• Instead, you tell the MC:
  • That interrupts can happen
  • And how to handle an interrupt when it does occur
Setting Up Interrupts

• Referring to the Reference Manual,
  • “Only INT0 is hard-mapped on the microcontroller… The other external interrupts are accessible via PPS…”

• So by default, you only have 1 single interrupt. But it is simple to map others (covered in lab 6)
A Few Registers...

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Bit 31/23/15/7</th>
<th>Bit 30/22/14/6</th>
<th>Bit 29/21/13/5</th>
<th>Bit 28/20/12/4</th>
<th>Bit 27/19/11/3</th>
<th>Bit 26/18/10/2</th>
<th>Bit 25/17/9/1</th>
<th>Bit 24/16/8/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23:16</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:8</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>R/W-0</td>
<td>U-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MVEC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TPC&lt;2:0&gt;</td>
<td></td>
</tr>
<tr>
<td>7:0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Interested in bits 4-0, why?
  - What does this do?

- Page 90
A Few Registers...

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Bit 31/23/15/7</th>
<th>Bit 30/22/14/6</th>
<th>Bit 29/21/13/5</th>
<th>Bit 28/20/12/4</th>
<th>Bit 27/19/11/3</th>
<th>Bit 26/18/10/2</th>
<th>Bit 25/17/9/1</th>
<th>Bit 24/16/8/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IEC31</td>
<td>IEC30</td>
<td>IEC29</td>
<td>IEC28</td>
<td>IEC27</td>
<td>IEC26</td>
<td>IEC25</td>
<td>IEC24</td>
<td></td>
</tr>
<tr>
<td>23:16</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IEC23</td>
<td>IEC22</td>
<td>IEC21</td>
<td>IEC20</td>
<td>IEC19</td>
<td>IEC18</td>
<td>IEC17</td>
<td>IEC16</td>
<td></td>
</tr>
<tr>
<td>15:8</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IEC15</td>
<td>IEC14</td>
<td>IEC13</td>
<td>IEC12</td>
<td>IEC11</td>
<td>IEC10</td>
<td>IEC09</td>
<td>IEC08</td>
<td></td>
</tr>
<tr>
<td>7:0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IEC07</td>
<td>IEC06</td>
<td>IEC05</td>
<td>IEC04</td>
<td>IEC03</td>
<td>IEC02</td>
<td>IEC01</td>
<td>IEC00</td>
<td></td>
</tr>
</tbody>
</table>

- This register is unreadable, but necessary
  - What is it used for?

- Page 92
A Few Registers...

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Bit 31/23/15/7</th>
<th>Bit 30/22/14/6</th>
<th>Bit 29/21/13/5</th>
<th>Bit 28/20/12/4</th>
<th>Bit 27/19/11/3</th>
<th>Bit 26/18/10/2</th>
<th>Bit 25/17/9/1</th>
<th>Bit 24/16/8/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP03&lt;2:0&gt;</td>
<td></td>
<td>IS03&lt;1:0&gt;</td>
<td></td>
</tr>
<tr>
<td>23:16</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP02&lt;2:0&gt;</td>
<td></td>
<td>IS02&lt;1:0&gt;</td>
<td></td>
</tr>
<tr>
<td>15:8</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP01&lt;2:0&gt;</td>
<td></td>
<td>IS01&lt;1:0&gt;</td>
<td></td>
</tr>
<tr>
<td>7:0</td>
<td>U-0</td>
<td>U-0</td>
<td>U-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
<td>RW-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP00&lt;2:0&gt;</td>
<td></td>
<td>IS00&lt;1:0&gt;</td>
<td></td>
</tr>
</tbody>
</table>

• Another tough one...
  • What is the purpose here?

• Page 93
### REGISTER 7-4: IFSx: INTERRUPT FLAG STATUS REGISTER

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Bit 31/23/15/7</th>
<th>Bit 30/22/14/6</th>
<th>Bit 29/21/13/5</th>
<th>Bit 28/20/12/4</th>
<th>Bit 27/19/11/3</th>
<th>Bit 26/18/10/2</th>
<th>Bit 25/17/9/1</th>
<th>Bit 24/16/8/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IFS31</td>
<td>IFS30</td>
<td>IFS29</td>
<td>IFS28</td>
<td>IFS27</td>
<td>IFS26</td>
<td>IFS25</td>
<td>IFS24</td>
<td></td>
</tr>
<tr>
<td>23:16</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IFS23</td>
<td>IFS22</td>
<td>IFS21</td>
<td>IFS20</td>
<td>IFS19</td>
<td>IFS18</td>
<td>IFS17</td>
<td>IFS16</td>
<td></td>
</tr>
<tr>
<td>15:8</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IFS15</td>
<td>IFS14</td>
<td>IFS13</td>
<td>IFS12</td>
<td>IFS11</td>
<td>IFS10</td>
<td>IFS09</td>
<td>IFS08</td>
<td></td>
</tr>
<tr>
<td>7:0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
<tr>
<td>IFS07</td>
<td>IFS06</td>
<td>IFS05</td>
<td>IFS04</td>
<td>IFS03</td>
<td>IFS02</td>
<td>IFS01</td>
<td>IFS00</td>
<td></td>
</tr>
</tbody>
</table>

• Another tough one...
  • What is the purpose here?

• Page 92
**TABLE 7-1: EXTREMELY IMPORTANT!!!**

- Make sure to understand this table.
  - What is the vector for INT2?

- This gives you all you need to know to understand the registers on previous slides...

![Table 7-1: Interrupt IRQ, Vector and Bit Location](image)

<table>
<thead>
<tr>
<th>Interrupt Source$^{(1)}$</th>
<th>IRQ #</th>
<th>Vector #</th>
<th>Interrupt Bit Location</th>
<th>Persistent Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Priority</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sub-priority</td>
<td></td>
</tr>
</tbody>
</table>

Gives Vector #
Gives Flag location
Gives Enable bit location
Gives Priority/Sub-priority location
Questions!

• Which register holds INT0’s Flag?

• What about INT1’s Flag?

• Which register holds INT2’s Priority?

• What about INT3’s Priority?

• Where do you find INT4’s Enable?
Register Summary

- **INTCON**
  - Used for us to set interrupt edge trigger direction

- **IECx**
  - Enable Register

- **IFSx**
  - High values mean interrupt has occurred

- **IPCx**
  - Priority Control registers
  - Here, pick something from 1(highest) to 7(lowest) priority

- Using INTCONbits, IEC0bits, IFS0bits, and IPCxbits will help TONS
Using Interrupts

• Make sure to enable interrupts globally!
  • INTEnableSystemMultiVectoredInt(); //Need this line exactly as written!!!

• Then use earlier registers to enable, set edge, and set priority.
  • Is good practice to clear the flags as well

• We need one more thing...
Interrupt Service Routine!

• ISRs tell the microcontroller what to do when interrupts occur

• Interrupts usually have the same flow...
  • DO INTERRUPT WORK
  • CLEAR CORRESPONDING FLAG
  • RETURN

• We will write our ISR using a handy macro
__ISR(vector, ipl)

• Note, there are 2 underscores!!!

• Guess what vector is?

• `ipl` refers to the interrupt priority level, but we can actually omit this and let the MC automatically assign a level (works for our purposes)

• What Interrupt does the following ISR correspond to?
  • __ISR(11)
Writing ISR Function

• Functions are of the following prototype
  • \texttt{RETURN\textsc{type} function\textsc{Name}(\textsc{parameters})}\{  
    //Code to execute
  \}

• To write an ISR, you will use the macro on the previous page
  • \texttt{RETURN\textsc{type} \_\_ISR(vector) function\textsc{Name}(\textsc{parameters})}\{  
    //Code to execute
  \}

• Does an interrupt return data? Does it have any parameters?
ISR Flow

• Main runs like normal – should have infinite loop executing over and over

• Interrupt is Triggered (input to MC changed)
  • This sets a flag
  • Which in turn alerts the MC that an interrupt occurred
  • MC saves main state
  • MC looks up what to do when this specific interrupt occurs
  • MC executes what it found
  • If flag is clear, interrupt restores main state and continues operation
Project Goal

• To implement an ISR using INT0 triggered by a switch

• Find Lab1’s code (count to 15 in binary on LEDs)

• Your ISR will Display a set value on the LEDs (pick anything, I personally like turning them ALL ON), wait a few seconds, then exit the interrupt and return normal functionality as if nothing happened!
Switch Problems

• Switches have a tendency to rapidly bounce between 1/0 when you first toggle them. To remedy this, you can use a debouncer, shown below!

• Hook up the switch as shown here
  • RESISTORS REQUIRED!!!!! WHY?

• Wire one output to INT0

• NAND is the 74LS00
Crude Block Diagram (Not Drawn Accurately)

DO NOT FORGET GROUND, POWER, AND RESISTORS!
THESE ARE NOT SHOWN HERE
What you should have learned...

- What an interrupt is
- How to set up interrupts
- How to handle interrupts
Lab 6

Peripheral Pin Select
What is Peripheral Pin Select (PPS)

• Recall from last lab that you only have access to INT0 directly

• We can, however, use more interrupts through PPS!

• This is a mapping system that lets us change what modules of the PIC32 connect to the external pins
Input Mapping

• Maps many pins to a single peripheral

• Multiplexer selects the external pin used

• Pay attention to pin references here
Output Mapping

• Maps many output peripherals to a single pin

• Multiplexer selects which peripheral will output on a given pin
Setting PPS

• There are safeguards protecting against certain PPS operations

• PPS pins may only be set once based on configuration bits (bits that control operational modes of the microcontroller)
  • IOL1WAY

• Some bits even prevent setting PPS at all
  • IOLOCK

• These bits CAN be modified!
Useful Macros!

• PPSInput(grp,fn,pin)

• PPSOutput(grp,pin,fn)

• FN and PIN are easy, what does GROUP apply to?

• Make a guess based on tables 11-1 (pg 146) and 11-2 (pg 148)

• These replace manually setting registers based on tables on 149
  • Each multiplexer can be thought of as a register, and this gets confusing really fast!
Few more macros

• PPSUnlock

• PPSLock

• These LOCK and UNLOCK the IOLOCK bit.

• NOTE: If you unlock, make sure to lock after making your changes!
Let’s Map!

• What to write to map:
  
  • C1 $\rightarrow$ U1RX
  
  • B8 $\rightarrow$ IC3
  
  • OC2 $\rightarrow$ B5
  
  • C1OUT $\rightarrow$ B10
Now you can set up multiple interrupts!

• This will allow you to interface correctly with a Rotary Pulse Generator

• These are commonly used to determine rotational position of motors or other devices
Grayhill 61C Optical Encoder

• Refer to datasheet for wiring schematic
  • We do not care about pins 2 and 3
  • Make sure to pay attention numbering and all components!

• We will use this to determine if we are rotating CW or CCW!
61C Table

- 2 outputs, 2 interrupts!

- With careful logic, you can determine direction you rotate WITHOUT saving previous position!
Project Goal

• You will use PPS to enable a second interrupt!
  • Carefully choose your PPS pin for 5V tolerance!

• You will maintain a GLOBAL count
  • CW rotations increase count
  • CCW rotations decrease count

• Modifications to count are to be done ONLY IN INTERRUPTS
  • This will be done based off of understanding the table for the 61C encoder
  • DO NOT USE PREVIOUS STATE LOGIC

• Display count on LEDs (so count from 0->15 and wrap around)
PPS Note

• The Cmod reference manual gives pin assignments for PPS available pins
  • May be easiest for TA to assist you if you follow these pin assignments!
Crude Block Diagram (Not Drawn Accurately)

DO NOT FORGET GROUND, POWER, AND RESISTORS! THESE ARE NOT SHOWN HERE
What You should have learned...

• How to use PPS

• How to understand a Rotary Pulse Generator
  • Usage and deeper understanding of logic
Lab 7

Timers
Timers!

• Timers are extremely useful and are the basis for many peripherals in any microcontroller

• Timers alone can act as “alarms” or counters
16 bit and 32 bit Timers Available!

• 16 bit timers are more simple, but restrictive
  • Type A
  • ONLY 16 bit

• 32 bit timers allow for larger timers/counters, but more difficult to set up!
  • Type B
  • Made by combining two 16 bit timers of Type B
Type A (16 bit only) timers

• To put it simply, a comparator that sets a flag when timer reaches PRx value!

• Where timers get complicated is selecting source and type of operation.

• Type A has special block which allows for RTC applications

Note
1: For information on enabling the 32 kHz Secondary Oscillator (Sosc), refer to Section 6. “Oscillators” (DS61112).
2: The default state of the SOSCEN bit (OSCCON<1>) during a device Reset is controlled by the FSOSCEN bit (DEVCFG1<5>).
Type B (16 bit) timers

- Notice this looks generally the same, with a few differences.
- More possible prescale values
- No block for RTC
- Added ADC functionality for Timer3
Type B (32 bit) timers

- Two Type B timers smooshed together

- Overall logic is the same!

- Can only be timers 2/3 or 4/5

- Even timer supplies control logic!

- Odd timer supplies interrupt!

---

**Note:** The TxCK pin is not available in certain PIC32MX family devices. In such cases, the timer must use the peripheral clock as its input clock. Refer to the “Timers” chapters in the specific device data sheet for the I/O pin details.
Registers

• If you can follow the diagrams, you can understand registers!

• TxCON – Control Register
  • Enable
  • Source Select
  • Prescale
  • T32 – Enable 32 bit operation (ONLY ON EVEN NUMBER TIMERS)

• TMRx – Running Counter
  • Reset when matched!

• PRx – Value to “count” to, or value to match
Design Notes

• 16 Bit timers are simple, each uses all of it’s own control and TMR/PR registers

• 32 Bit timers ignore the odd timer control bits, TMR register, and PR register.
  • When using Timers 2/3 as 32 bit
    • Write to TMR2 and PR2 and T2CON to setup timer
    • Set up Timer 3’s interrupts and use it’s values for ISR (refer to Interrupt lab!)
MUSIC!

• What is music?
  • Think physics!

• How could we use timers to play a song?
Frequency

• What is frequency?

• How can you translate this into something a microcontroller can understand?
More Timer Notes

• **CANNOT** Modify timer period while timer is running!
  • Unless in an interrupt
  • What does this mean when writing a song??

• Attempting to edit period register may result in undescribed behavior
  • These notes are taken directly from Reference Manual!
Today’s Objectives

• A framework for today's lab is provided
  • We will be giving the control structure to switch to the next note and how long to play each note
  • You must set up an interrupt for the timer you select that updates the period and plays music as necessary!

• Make a song!
  • Will need to use speaker (AC-1005G-P)
  • How does speaker work??
  • We will neither need an amplifier nor a resistor for our purpose
DO NOT FORGET GROUND, POWER, AND RESISTORS!
THESE ARE NOT SHOWN HERE
What You should have learned...

• How to set up and use timers

• How speakers work and how to make music!
Lab 8

Pulse Width Modulation
PWM

• What is a PWM?

• What do you use them for?
Output Compare

• Some Microcontrollers have dedicated PWM peripherals, but not the PIC32

• Output Compare must be used to generate PWM signals

• What is output compare?
Look Familiar?

Note 1: Where ‘x’ is shown, reference is made to the registers associated with the respective output compare channels, 1 through 5.

2: The OCFA pin controls the OC1-OC4 channels. The OCFB pin controls the OC5 channel.
Very Similar to Timers!

• You can determine some registers required from the block diagram

• Refer to register 15-1
  • What bit(s) are required for operation?
  • How do we select PWM Mode?
PWM Output

• What defines your period?
• What defines your Duty Cycle?
  • How do you mathematically define the Duty Cycle (percentage)?
Design Notes

• In PWM mode, ONLY OCxRS can be written during operation
  • The PWM will AUTOMATICALLY write OCxRS to OCxR when the previous PWM cycle is finished

• This means you can ONLY write DIRECTLY to OCxR when the OC module is off!

• You still need to set up the timer source you select!
Today’s Objectives

• Set up an Output Compare Module in PWM mode to drive a motor using the Motor Driver IC (L293DNE) in your kit

• Use an external interrupt to modify the Duty Cycle
  • Cycle through duties of 0%, 25%, 50%, 75%, and 100%
  • So a push of the button will increase the duty cycle by 25% and will wrap around automatically

• EXTRA CREDIT!
  • If you can correctly utilize 2 PWMs to drive the motor in both directions using the H-Bridge capabilities of the motor driver chip
Crude Block Diagram (Not Drawn Accurately)

DO NOT FORGET GROUND, POWER, RESISTORS, AND DIODES!!
THESE ARE NOT SHOWN HERE
What You should have learned...

• How to set up Output Compare in PWM mode

• How to use the motor driver chip

• Extra Credit: How to properly use the H-Bridge to drive the motor in both directions
Lab 9
Serial Peripheral Interface
SPI

• What is SPI?

• What do you use it for?
Block Diagram

• What’s Important here?

• What does the “Note:” mean?

Note: Access SPIxTXB and SPIxRXB FIFOs via SPIxBUF register.
Questions (Refer to Registers!)

• What data widths are available?

• Which bit enables the SPI Output?

• Is SSx used by default?

• What is the purpose of MSSEN?
Master Mode

• Note arrow direction!

• Care to guess how this works?
Slave Mode

- What’s different?
- How does this work?

Note 1: In Normal mode, the usage of the Slave Select pin (SSx) is optional.
2: The control of the SDOx pin can be disabled for Receive-Only modes.
SPI Output

• Refer to Table 23-9 in the SPI reference

• Why is SCKx repeated?
Design Notes

• Read the configuration registers carefully!
  • Many more bits will be used in this lab than with previous peripherals!

• Check 23.3.3.1 for Master Mode Operation Setup
  • THIS IS NOT ALL INCLUSIVE! – Meaning there is more to do than the 10 steps listed!!
Today’s Objectives

• Set up SPI in master mode to output to the **SN74HC595** shift register
  • What data width will you use?
  • SPI will handle ALL of the logic for the Shift Register
    • So make sure to set SPI up correctly in correspondence with the SN74HC595 datasheet!!!

• Use an interrupt to send data via SPI
  • Using an Array of Known values is suggested, like
  • `char spiChars[18] = {0, 1, 2, 4, 8, 16, 32, 64, 128, 255, 254, 253, 251, 247, 239, 223, 191, 127};`
Crude Block Diagram (Not Drawn Exactly)

DO NOT FORGET GROUND, POWER, RESISTORS, AND DIODES!!
THESE ARE NOT SHOWN HERE
What You should have learned...

- How to use SPI and how it works
- How to use a shift register
Lab 10

Analog to Digital Converter
ADC

• What is ADC?

• What do you use it for?
ADC Process

• Two main steps
  • Sampling

• Conversion
$T_{AD}$

- $T_{AD}$ is a measure of time for 1 ADC cycle

- This will most likely not equal your reference clock!!

- Sampling generally wants to last a few $T_{AD}$s to make sure the value on the input samples correctly

- Conversion takes $1 \ T_{AD}/\text{bit} + 2 \ T_{AD} = 12 \ T_{AD}$

- You’ll define $T_{AD}$ timings, and then you will define all ADC functions based on $T_{AD}$s
Block Diagram

• What’s Important here?

• It’s really less scary that it seems!
Questions (Refer to Registers!)

• How wide is our ADC?

• How many ways can we tell the ADC to format output data?

• How do we select the MC Voltage/Ground as our reference voltage?

• How do we select AN4 as our input voltage?

• How do we set up Auto-Conversion and what does this do?

• What about Auto-Sampling?
ADC Interrupt

• How do we set this up?

• When is the interrupt generated?
Design Notes

• Read the configuration registers carefully!
  • Similar to SPI, there are a TON of configuration bits required!

• Check 17.4 for ADC Setup
  • THIS IS NOT ALL INCLUSIVE! – Meaning there is more to do than the 15 steps listed!!

• Remember ANSELx? We will use this with the Reference Manual when setting up Analog Pins!
  • We do not have AD1PCFG!
Today’s Objectives

• You will convert the output of an FSR circuit to a digital value and display 8 bits on LEDs.
  • Should you show the most significant or least significant 8 bits?
  • Handle LED display with the ADC interrupt
  • How should we sample and convert?

• So you should see the LEDs start to light up as you press the FSR, but none will be on when you release it.

• Pay close attention to the FSR wiring diagram!
DO NOT FORGET GROUND, POWER, RESISTORS, AND DIODES!!
THESE ARE NOT SHOWN HERE
What You should have learned...

• How to use ADC

• How to use an FSR