

Microcontroller Interfacing Lab

ECE 3720

Syllabus

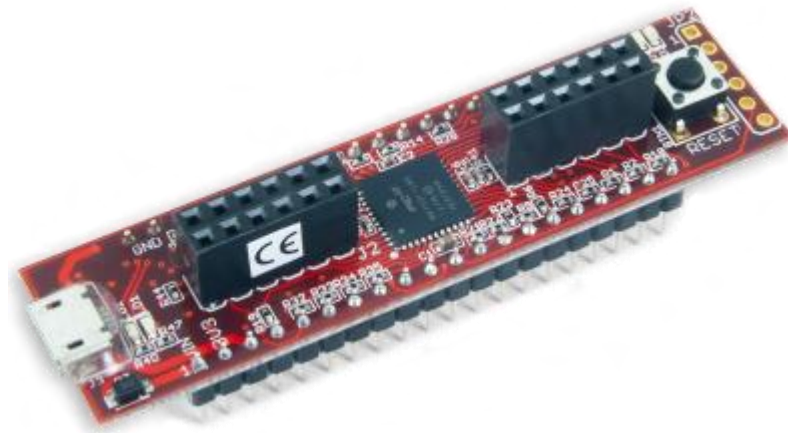
- **Course objective:** Learn about the functionality and modules of a microcontroller, and how to use it to interface with various devices.
- **Pre-Labs:** wiring diagram for upcoming lab
- **Post-Labs:** follow outline provided on Canvas
- **Quizzes:** cover material from preceding lab and upcoming lab
 - Of the three weekly assignments, this should be done last
- **Final design project:** come up with a design that incorporates elements of multiple previous labs
 - More details will be provided closer to the end of the semester



These three assignments will be due before each lab.

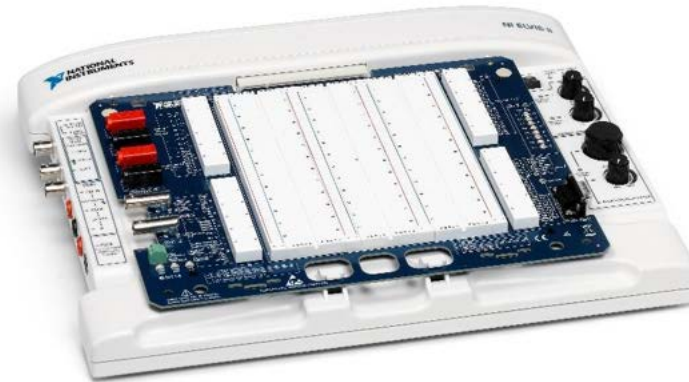
Grade Distribution	
Post-Lab Reports	40%
Pre-Labs	5%
Lab Completion	15%
Quizzes	15%
Design Project	25%

Equipment



ChipKIT Cmod

- Contains a PIC32MX150F128D microcontroller
- Allows easy access to microcontroller's pins



NI-Elvis II or Analog Discovery 2 (AD2)

- Multi-function instruments
- Will be used to supply power, inputs, and outputs for the MC



ChipKIT PGM

- Allows programming of the MC with MPLAB X IDE

Software Development Environment



MPLAB X will be used to write and compile code, and load the executable onto the microcontroller

What is MPLAB X IDE?

MPLAB® X IDE IS A SOFTWARE PROGRAM THAT IS USED TO DEVELOP APPLICATIONS FOR MICROCHIP MICROCONTROLLERS AND DIGITAL SIGNAL CONTROLLERS.

This development tool is called an Integrated Development Environment, or IDE, because it provides a single integrated “environment” to develop code for embedded microcontrollers. MPLAB X IDE incorporates powerful tools to help you discover, configure, develop, debug and qualify your embedded designs. MPLAB X IDE works seamlessly with the MPLAB development ecosystem of software and tools, many of which are completely free.

MPLAB X user's guide, pg. 7

Documentation



MICROCHIP **PIC32MX1XX/2XX**

32-bit Microcontrollers (up to 256 KB Flash and 64 KB SRAM) with Audio and Graphics Interfaces, USB, and Advanced Analog

Operating Conditions

- 2.3V to 3.6V, -40°C to +105°C, DC to 40 MHz
- 2.3V to 3.6V, -40°C to +85°C, DC to 50 MHz

Core: 60 MHz/83 DMIPS MIPS32® M4K®

- MIPS16e® mode for up to 40% smaller code size
- Code-efficient (C and Assembly) architecture
- Single-cycle (MAC) 32x16 and two-cycle 32x32 multiply

Clock Management

- 0.9% internal oscillator
- Programmable PLLs and oscillator clock sources
- Fail-Safe Clock Monitor (FSCM)
- Independent Watchdog Timer
- Fast wake-up and start-up

Power Management

- Low-power management modes (Sleep and Idle)
- Integrated Power-on Reset and Brown-out Reset
- 0.5 mA/MHz dynamic current (typical)
- 20 µA I/O current (typical)

Audio Interface Features

- Data communication: I²S, I²C, and DSP modes
- Control interface: SPI and I²C™
- Master clock:
 - Generation of fractional clock frequencies
 - Can be synchronized with USB clock
 - Can be tuned in run-time

Advanced Analog Features

- ADC Module:
 - 10-bit 1.1 Msps rate with one S&H
 - Up to 10 analog inputs on 28-pin devices and 13 analog inputs on 44-pin devices
- Flexible and independent ADC trigger sources
- Charge Time Measurement Unit (CTMU):
 - Supports mTouch™ capacitive touch sensing
 - Provides high-resolution time measurement (1 ns)
 - On-chip temperature measurement capability

Timers/Output Compare/Input Capture

- Five General Purpose Timers:
 - Five 16-bit and up to two 32-bit Timers/Counters
- Five Output Compare (OC) modules
- Five Input Capture (IC) modules
- Peripheral Pin Select (PPS) to allow function remap
- Real-Time Clock and Calendar (RTCC) module

Communication Interfaces

- USB 2.0-compliant Full-speed DTG controller
- Two UART modules (12.5 Mbps):
 - Supports LIN 2.0 protocols and IrDA® support
- Two 4-wire SPI modules (25 Mbps)
- Two I²C modules (up to 1 Mbaud) with SMBus support
- PPS to allow function remap
- Parallel Master Port (PMP)

Direct Memory Access (DMA)

- Four channels of hardware DMA with automatic data size detection
- Two additional channels dedicated for USB
- Programmable Cyclic Redundancy Check (CRC)

Input/Output

- 10 mA source/sink on all I/O pins and up to 14 mA on non-standard V_{OH}
- 5V-tolerant pins
- Selectable open drain, pull-ups, and pull-downs
- External interrupts on all I/O pins

Qualification and Class B Support


- AEC-Q100 REVG (Grade 2 -40°C to +105°C) planned
- Class B Safety Library, IEC 60730

Debugger Development Support

- In-circuit and in-application programming
- 4-wire MIPS® Enhanced JTAG interface
- Unlimited program and six complex data breakpoints
- IEEE 1149.2-compatible (JTAG) boundary scan

PIC32 Datasheet

For information about the modules and registers of the microcontrollers



DIGILENT

chipKIT™ Cmod™ Reference Manual

Revised November 6, 2013
This manual applies to the chipKIT Cmod rev. E

Overview

The chipKIT Cmod is a chipKIT/MPIDE compatible board from Digilent. It combines a Microchip® PIC32MX150F128D microcontroller with a convenient 600-mil, 40-pin DIP package and two Digilent Pmod connectors. Digilent's Cmod boards are ideally suited for breadboards or other prototype circuit designs where the use of small surface mount packages is impractical.

The chipKIT Cmod takes advantage of the powerful PIC32MX150F128D microcontroller. This microcontroller features a 32-bit MIPS processor core running at 40MHz, 128K of flash memory, and 32K of SRAM data memory.

The chipKIT Cmod can be programmed using the Multi-Platform Integrated Development Environment, MPIDE, an environment based on the open source Arduino® IDE modified to support the PIC32 microcontroller. The board provides everything needed to start developing embedded applications using the MPIDE.

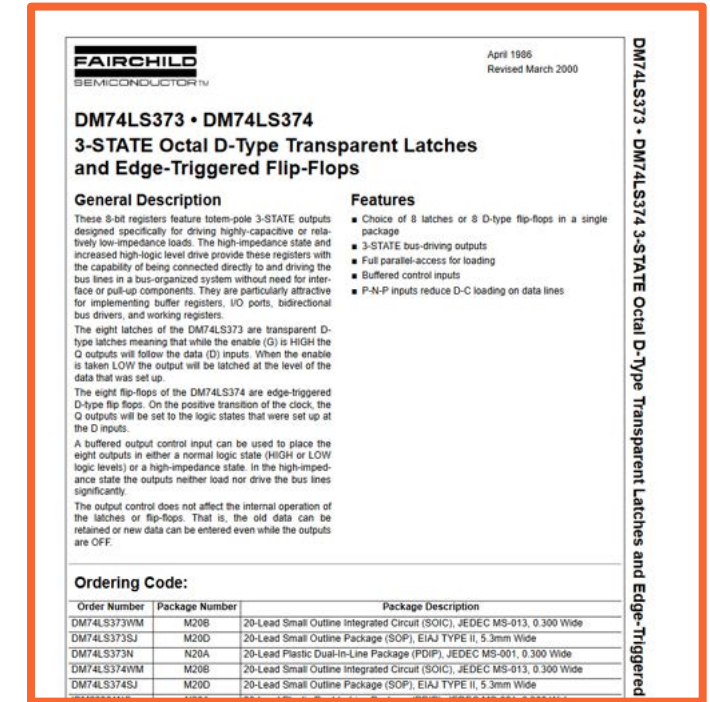
The chipKIT Cmod is also fully compatible with the advanced Microchip MPLAB® IDE. To develop embedded applications using MPLAB®, a separate device programmer/debugger, such as the Digilent chipKIT PGM or the Microchip PICKIT™3 is required.

Features include:

- Microchip® PIC32MX150F128D microcontroller (40/50 Mhz 32-bit MIPS, 128K Flash, 32K SRAM)
- Convenient 600-mil, 2x20-pin DIP package
- 5V – 12V recommended operating voltage
- 33 available I/O pins
- Two user LEDs
- PC connection uses a USB A to micro B cable (not included)
- 13 analog inputs
- 3.3V operating voltage

Cmod Reference Manual

For pinout diagram, power info, and other details about the board



FAIRCHILD SEMICONDUCTOR™

April 1985
Revised March 2000

DM74LS373 • DM74LS374
3-STATE Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops

General Description

These 8-bit registers feature totem-pole 3-STATE outputs designed specifically for driving highly-capacitive or relatively low-impedance loads. The high-impedance state and increased high-logic level drive provide these registers with the capability of being connected directly to and driving the bus lines in a bus-organized system without need for interface or pull-up components. They are particularly attractive for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers.

The eight latches of the DM74LS373 are transparent D-type latches meaning that while the enable (G) is HIGH the Q outputs will follow the data (D) inputs. When the enable is taken LOW the output will be latched at the level of the data that was set up.

The eight flip-flops of the DM74LS374 are edge-triggered D-type flip-flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D inputs.

A buffered output control input can be used to place the eight outputs in either a normal logic state (HIGH or LOW logic levels) or a high-impedance state. In the high-impedance state the outputs neither load nor drive the bus lines significantly.

The output control does not affect the internal operation of the latches or flip-flops. That is, the old data can be retained or new data can be entered even while the outputs are OFF.

Features

- Choice of 8 latches or 8 D-type flip-flops in a single package
- 3-STATE bus-driving outputs
- Full buffered-access for loading
- Buffered control inputs
- P-N-P inputs reduce D-C loading on data lines

Ordering Code:

Order Number	Package Number	Package Description
DM74LS373WM	M20B	20-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300 Wide
DM74LS373SJ	M20D	20-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
DM74LS373N	N20A	20-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300 Wide
DM74LS374WM	M20B	20-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300 Wide
DM74LS374SJ	M20D	20-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide

DM74LS373 • DM74LS374 3-STATE Octal D-Type Transparent Latches and Edge-Triggered

Other Datasheets

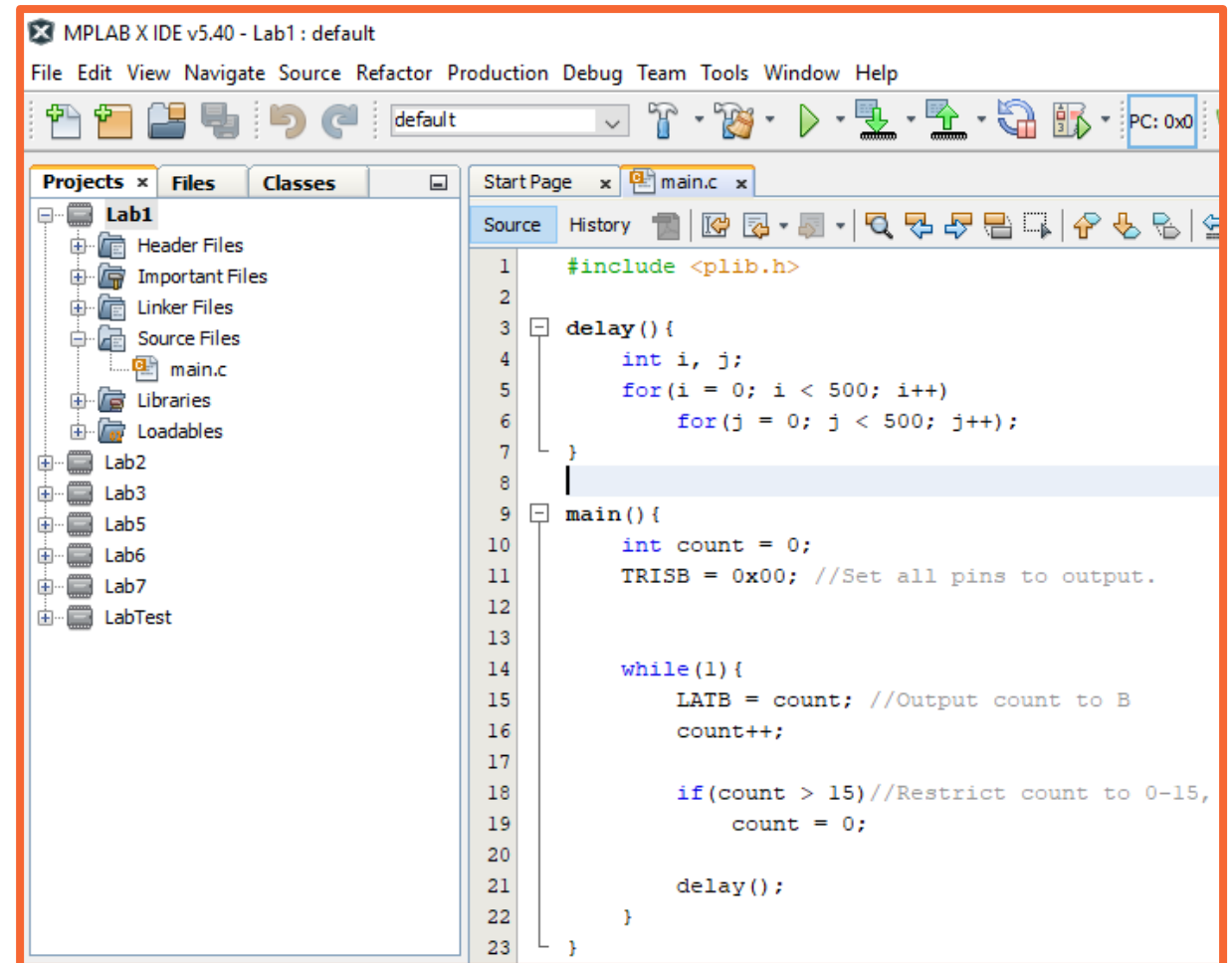
Describe pinout and behavior of the devices used in each lab

Lab 1: Intro

ECE 3720

Lab 1 Program



- Navigate to the *Lab 1* module in Canvas and download *Lab1.X.zip*
- Extract the *Lab1.X* project folder to your profile.
- Open MPLAB X, then click *File > Open Project...* and select and open the project.
- Under the *Projects* tab on the left side of the screen, expand *Lab1* and *Source Files*. You should see *main.c*. Double-click on it to open it.
- Observe how this program counts from 0 to 15, outputting the value on Port B
 - Lab 2 will cover the details of how this works.



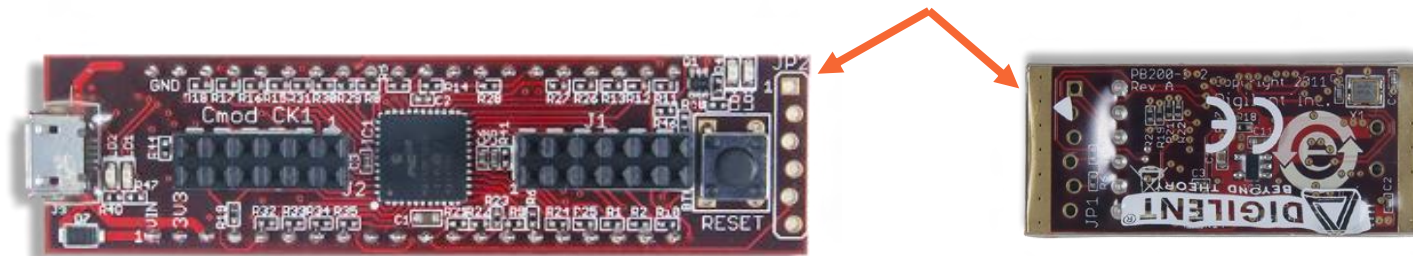
The screenshot shows the MPLAB X IDE interface. The title bar reads "MPLAB X IDE v5.40 - Lab1 : default". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Production, Debug, Team, Tools, Window, and Help. The toolbar contains various icons for file operations and execution. The left pane shows the "Projects" tab with a tree view for "Lab1" containing subfolders for Header Files, Important Files, Linker Files, Source Files (containing main.c), Libraries, and Loadables. Below this are other lab projects: Lab2, Lab3, Lab5, Lab6, Lab7, and LabTest. The right pane shows the "Source" view of the file "main.c". The code is as follows:

```
1  #include <plib.h>
2
3  delay(){
4      int i, j;
5      for(i = 0; i < 500; i++)
6          for(j = 0; j < 500; j++);
7  }
8
9  main(){
10     int count = 0;
11     TRISB = 0x00; //Set all pins to output.
12
13
14     while(1){
15         LATB = count; //Output count to B
16         count++;
17
18         if(count > 15)//Restrict count to 0-15,
19             count = 0;
20
21         delay();
22     }
23 }
```

Programming the Microcontroller

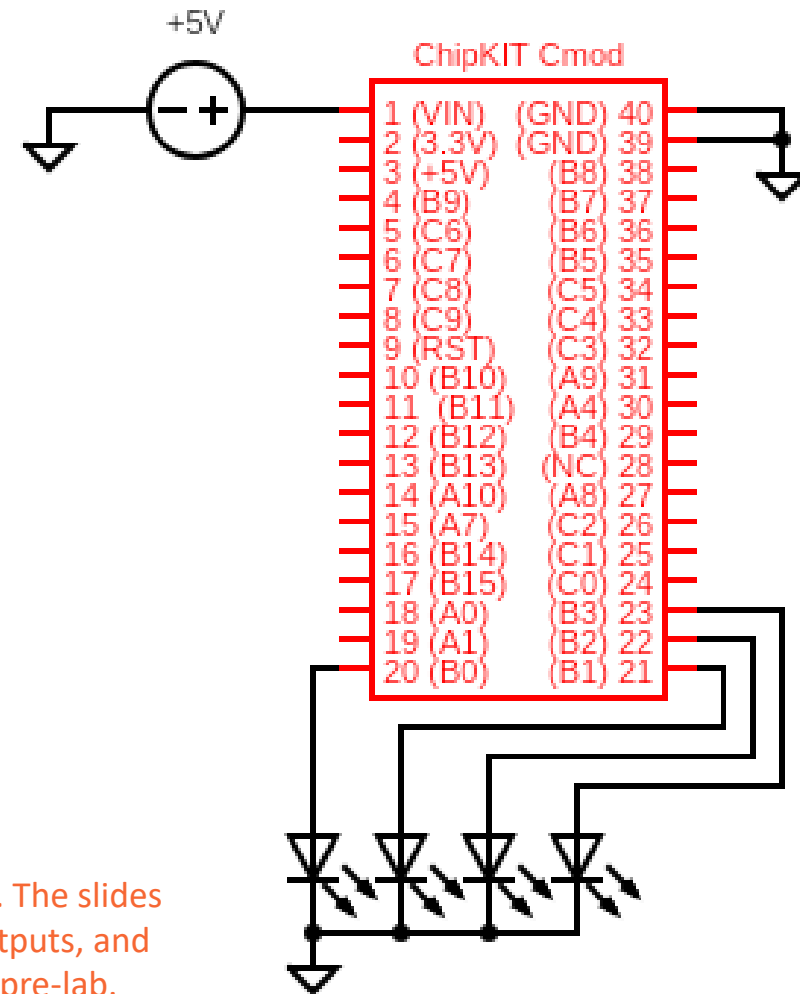
- Click the hammer icon () to build the program.
 - You will see warnings in the output window about outdated libraries. You can ignore these.
 - Look for the green *Build Successful*.
- Connect the Cmod to the PGM (see below) and plug the PGM into your computer's USB port.
- To load the program onto the microcontroller, click *Make and Program Device* ().
 - If asked to choose a device, look for the chipKIT device at the bottom of the list.
 - Watch the output window to see when the process is complete (it may take a while the first time).
 - Notice that the program is built as part of this process.

The arrow on the PGM should be closer to the side of the Cmod with this 1 label.



Lab 1 Wiring

- Power (5-12V) should be supplied to pin 1 of the Cmod, labeled *VIN*
- Pins 39 and 40 should always be grounded.
- Pins 20-23 should be connected to LEDs to display the output.
 - These pins correspond to the 4 least-significant bits of Port B



Note that you will not be given the wiring for future labs. The slides will provide a simple diagram showing the inputs and outputs, and you will have to submit a more detailed diagram as your pre-lab.

Lab 1 Modification

Once you have the program running on your microcontroller, attempt the following:

- Modify the code to count backwards, looping from 0 back up to 15.
- Modify *delay* to take a longer or shorter period of time.

Lab 2: Application of a Digital Latch

ECE 3720

Preview

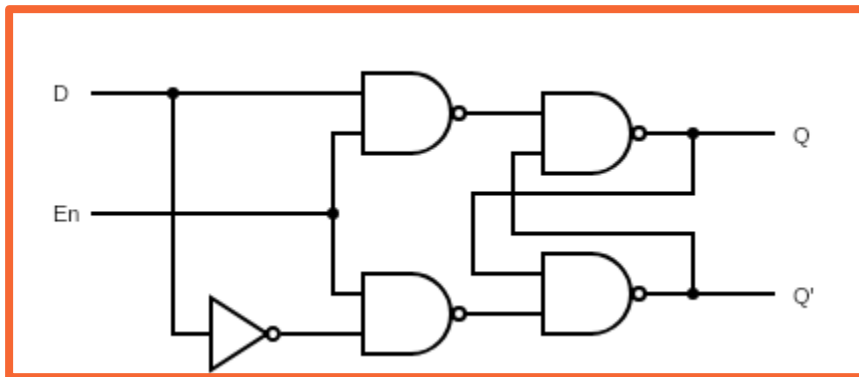
The microcontroller will read two digital inputs and output those same values to a latch. When a button is pressed, the latch outputs will be updated to mirror the inputs. The states of the latch's inputs and outputs will be indicated by LEDs.

Topic	Slide
Latches vs Flip-Flops	3
SN74LS373	4
I/O Registers (TRIS, LAT, PORT, ANSEL)	5
Push Buttons	9
Pull-Up and Pull-Down Resistors	10
Active-Low vs Active-High	11
DIO	12
Voltages	13
Creating a new MPLAB project	14
Lab Goals	18

Latches vs Flip-Flops

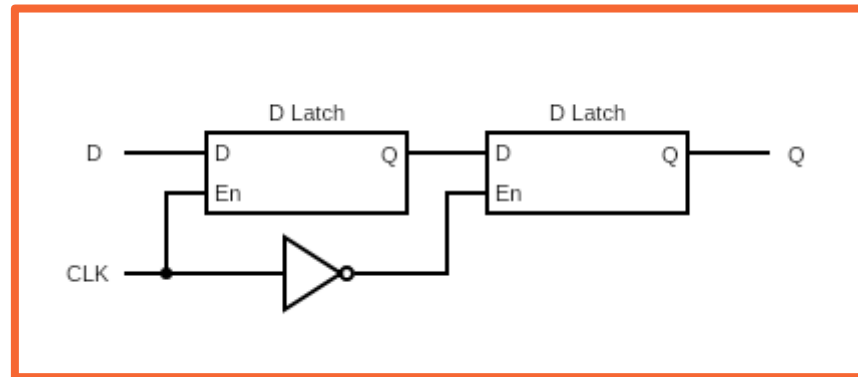
- Both latches and flip-flops are storage elements for holding a binary state.
- Latches are *level-sensitive*, while flip-flops are *edge-sensitive*.

D Latch



- While En is HIGH, Q mirrors D.
- When En is LOW, Q remains constant.
- **This is what will be used in this lab.**

D Flip-Flop



- Q updates to match D on rising edge of CLK.
- Consists of two D latches
- Useful for sequential circuits, since it will only update at the times determined by the clock.

SN74LS373

- Datasheet available [here](#)
- Notice that this datasheet covers multiple devices, including both latches and flip-flops.
- Use the datasheet to learn the behavior and pinout of the SN74LS373
 - Notice that the pins are arranged in a Q-D-D-Q-Q... pattern

The eight latches of the 'LS373 and 'S373 are transparent D-type latches, meaning that while the enable (C or CLK) input is high, the Q outputs follow the data (D) inputs. When C or CLK is taken low, the output is latched at the level of the data that was set up.

SN74LS373 datasheet, pg. 1

We use this

Output updates as long as C is HIGH

Function Tables

'LS373, 'S373
(each latch)

INPUTS			OUTPUT
\overline{OC}	C	D	Q
L	H	H	H
L	H	L	L
L	L	X	Q ₀
H	X	X	Z

Not this

Output updates on rising edge of CLK

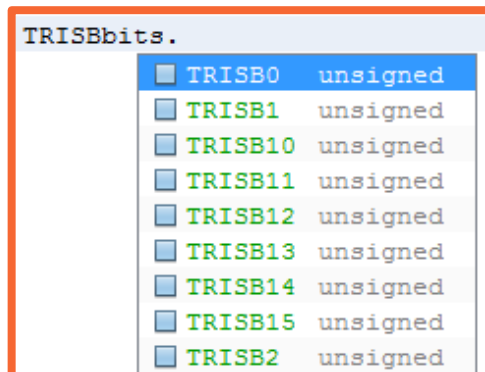
'LS374, 'S374
(each latch)

INPUTS			OUTPUT
\overline{OC}	CLK	D	Q
L	↑	H	H
L	↑	L	L
L	L	X	Q ₀
H	X	X	Z

SN74LS373 datasheet, pg. 3

Registers

- Every lab in this class will involve setting the values of the PIC32's registers to achieve the desired behavior.
 - In particular, the I/O registers detailed in the following slides will be some of the first things to consider for each lab.
 - Register names refer to memory locations, defined through plib.h.
- Xbits functionality
 - Type "bits." after a register name in MPLAB to see a list of individual bits from that register.
 - This is often preferable, since it improves readability and lets you address individual bits, leaving the others unchanged.



11.1 Parallel I/O (PIO) Ports

All port pins have 10 registers directly associated with their operation as digital I/O. The data direction register (TRISx) determines whether the pin is an input or an output. If the data direction bit is a '1', then the pin is an input. All port pins are defined as inputs after a Reset. Reads from the latch (LATx) read the latch. Writes to the latch write the latch. Reads from the port (PORTx) read the port pins, while writes to the port pins write the latch.

PIC32 datasheet, pg. 144

Data Direction

- Use TRISx registers to designate pins as inputs or outputs.
 - where x is A, B, or C
 - TRIS is short for “Tri-State”
- Set bitwise, 1 for input, 0 for output
 - (“1input or Output”)

```
TRISB = 0b00001111;    // Set B0-B3 as inputs and B4-B7 as outputs
TRISB = 0x0F;          // Equivalent
TRISB = 15;            // Equivalent

TRISBbits.TRISB2 = 1;  // Set just B2 as an input
TRISBbits.TRISB2 = 0; // Set just B2 as an output
```

11.1 Parallel I/O (PIO) Ports

All port pins have 10 registers directly associated with their operation as digital I/O. The data direction register (TRISx) determines whether the pin is an input or an output. If the data direction bit is a '1', then the pin is an input. All port pins are defined as inputs after a Reset. Reads from the latch (LATx) read the latch. Writes to the latch write the latch. Reads from the port (PORTx) read the port pins, while writes to the port pins write the latch.

PIC32 datasheet, pg. 144

Read / Write

- Read from PORTx
 - When a pin is designated as an input, reading the corresponding bit of PORTx will return the value on that pin.

```
int x;  
x = PORTAbits.RA0;    // Read the value of A0 into variable x
```

- Write to LATx
 - When a pin is designated as an output, writing to the corresponding bit of LATx will output that value on the pin.

```
LATCbits.LATC0 = 1;    //Write a 1 (HIGH)to C0  
  
int y = 12;  
LATB = y;              // Write the value of variable y to Latch B  
                      //(consider how many bits/pins this requires)  
  
LATCbits.LATC0 = PORTAbits.RA0; // What does this do?
```

11.1 Parallel I/O (PIO) Ports

All port pins have 10 registers directly associated with their operation as digital I/O. The data direction register (TRISx) determines whether the pin is an input or an output. If the data direction bit is a '1', then the pin is an input. All port pins are defined as inputs after a Reset. Reads from the latch (LATx) read the latch. Writes to the latch write the latch. Reads from the port (PORTx) read the port pins, while writes to the port pins write the latch.

PIC32 datasheet, pg. 144

Analog Select

- Use ANSELx to set pins to analog or digital mode.
- 0 for digital, 1 for analog
 - We will typically use digital.
- Use reference manual to determine which pins are analog capable.

```
ANSELB = 0;           // Set all B pins to digital
ANSELBbits.ANSB0 = 1; // Set B0 to analog mode
```

11.1.2 CONFIGURING ANALOG AND DIGITAL PORT PINS

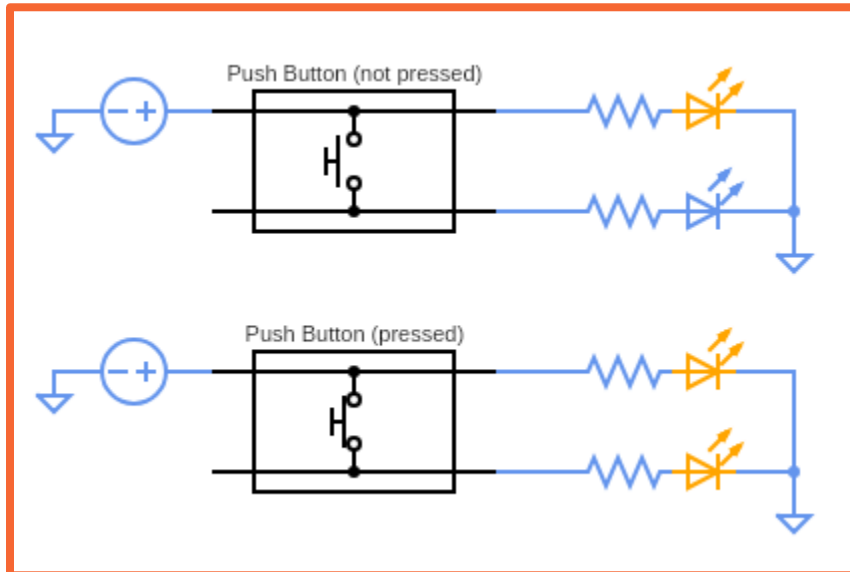
The ANSELx register controls the operation of the analog port pins. The port pins that are to function as analog inputs must have their corresponding ANSEL and TRIS bits set. In order to use port pins for I/O functionality with digital modules, such as Timers, UARTs, etc., the corresponding ANSELx bit must be cleared.

The ANSELx register has a default value of 0xFFFF; therefore, all pins that share analog functions are analog (not digital) by default.

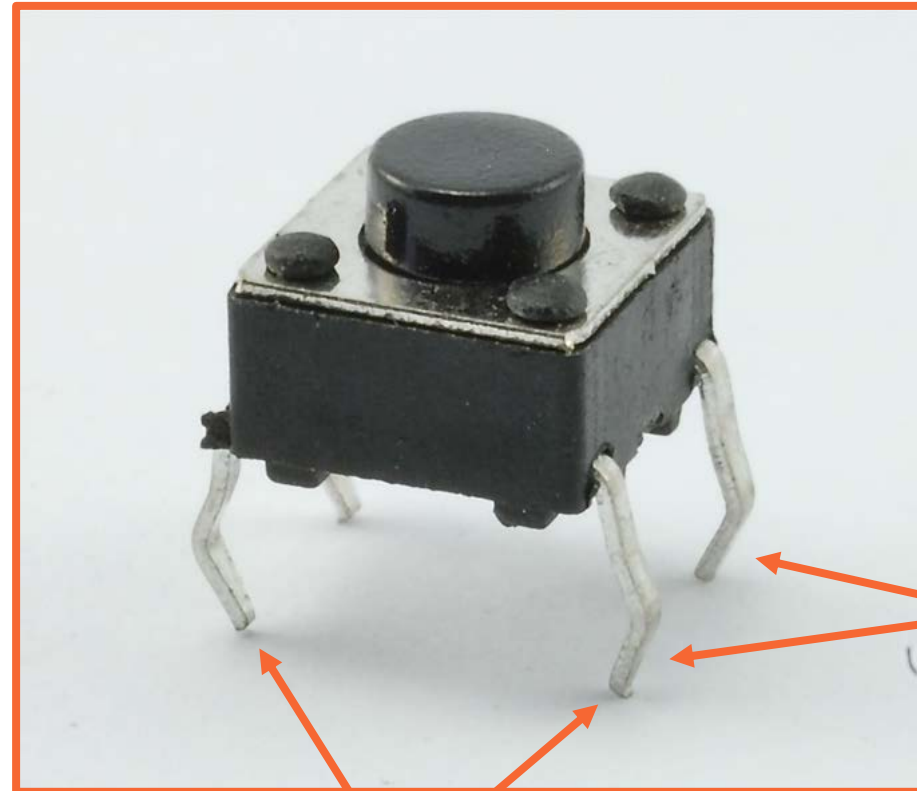
PIC32 datasheet, pg. 144

Push Buttons

Illustration of Internal Connection



Notice that the top LED in this setup is always connected to the voltage source, but the bottom is only connected when the button is pressed.



Connected when button is pressed

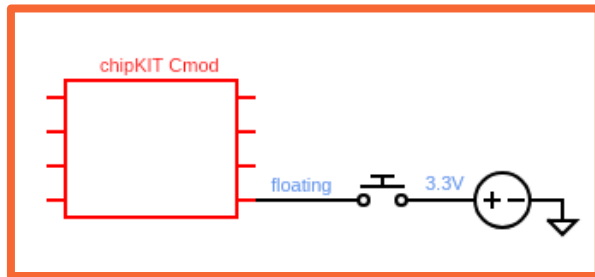
Always connected

Pull-Up and Pull-Down Resistors

- If a microcontroller's input pin is left floating, it may not read the correct digital value.
- Pull-up and pull-down resistors should be used to ensure consistent behavior.

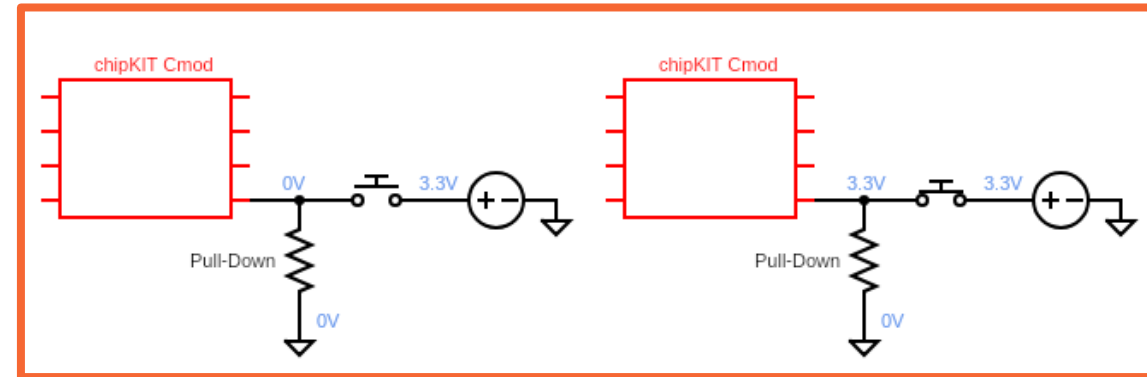
Floating (incorrect)

Pin has indeterminate value until button is pressed



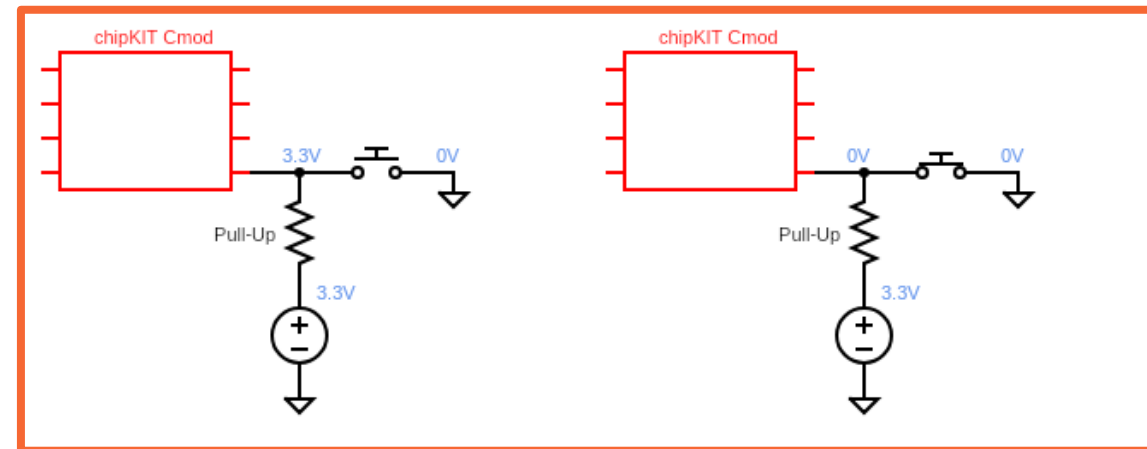
Pull-down

Pin is pulled LOW until button is pressed



Pull-up

Pin is pulled HIGH until button is pressed

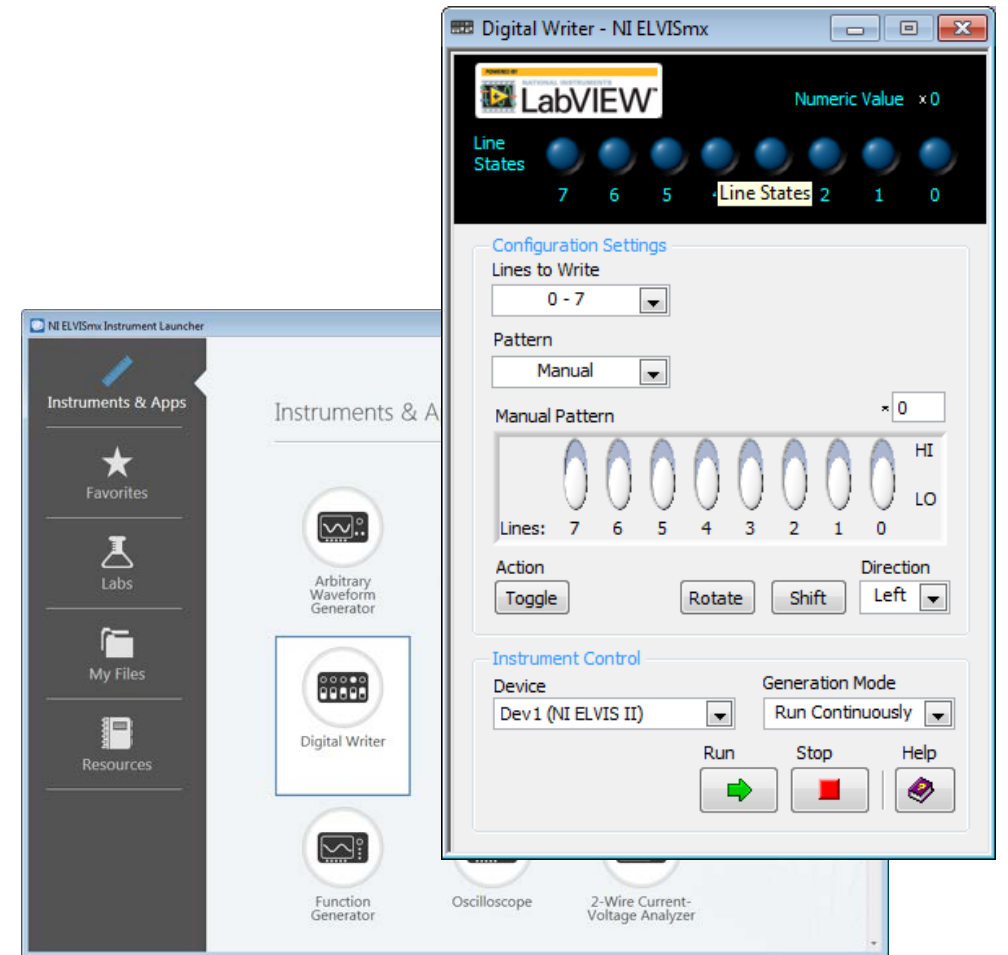
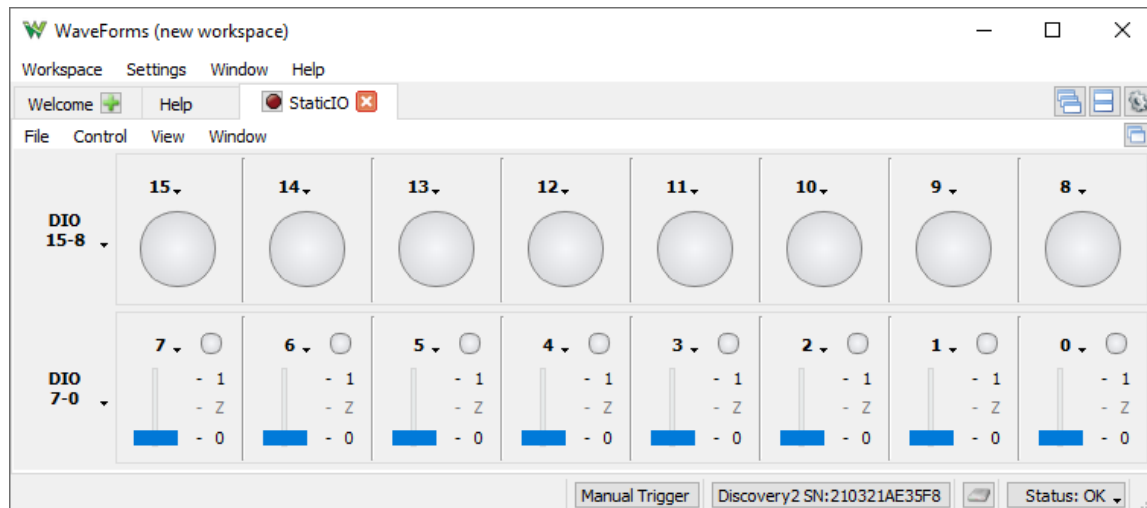


Active-Low vs Active-High

- An active-high signal is “on” when its voltage is HIGH.
- An active-low signal is “on” when its voltage is LOW.
 - e.g., the Output Control (/OC) pin of the SN74LS373
- **In this lab, we want the button’s output to be ACTIVE-LOW.**
 - **This means the MC will read LOW when the button is pressed, and HIGH otherwise.**
 - Reference the diagrams in the previous slide to determine how to wire the button to achieve this.
 - Since the button will be used to update the latch, consider what this means for the program you will write.

DIO (Digital Input/Output)

- If using the NI-ELVIS II:
 - Open the *NI Instrument Launcher* application and select *Digital Reader*.
 - Click the switches to toggle the outputs HIGH or LOW.
 - The DIO ports are located on the top-right section of the board.
- If using the Analog Discovery 2:
 - Open the *WaveForms* application and select *StaticIO*.
 - Click the drop-down arrow(s) and select push/pull switch.
 - The DIO pins are labeled 0-15 on the AD2's case.



Notes on Voltages

- The PIC32 runs on 3.3V, but the Elvis board uses 5V logic.
 - The chipKIT Cmod is powered by 5V, but uses a regulator to supply the PIC32 with 3.3V.
- **Only some of the PIC32's pins are 5V tolerant.**
 - Carefully read the pinout diagram in the reference manual to choose an appropriate pin.
- Pin 2 of the Cmod provides a 3.3V output.
 - This is the output of the regulator that powers the PIC32.
 - **Do not short this pin to ground or apply a voltage to it.**

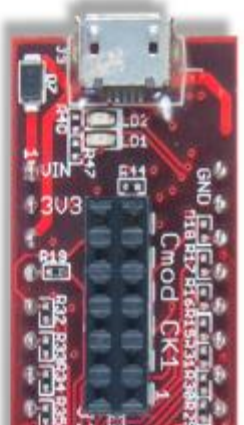
See sections 3 and 4 in the Cmod reference manual for details on the power supply and 5V compatibility.

chipKIT™ Cmod
Rev E


Flash 128 kB
RAM 32 kB

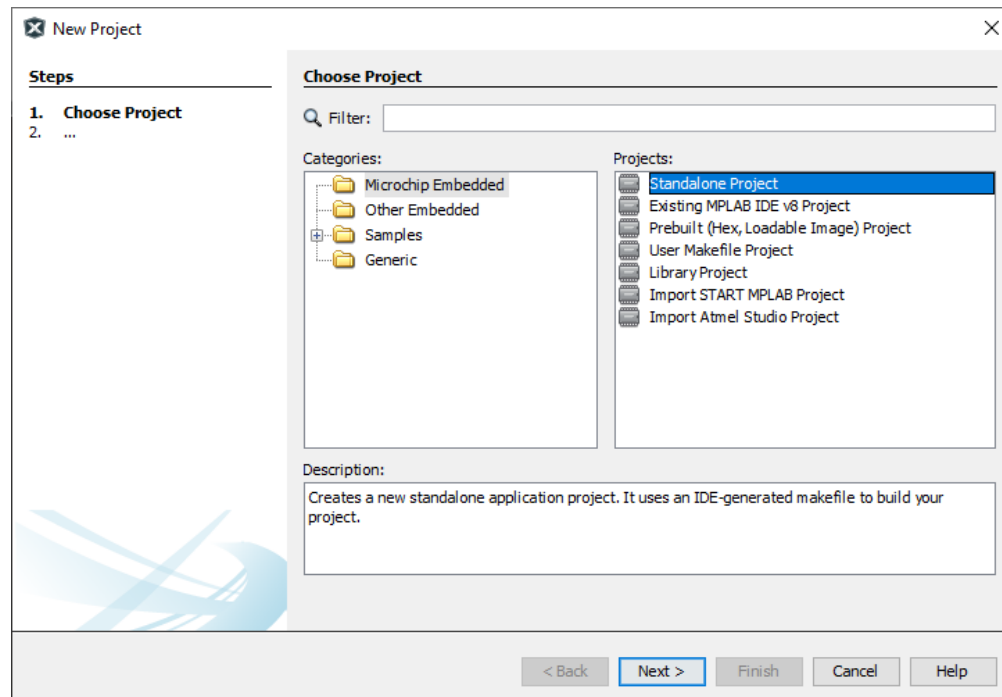
Inputs are 3.3V tolerant.
* indicates 5V tolerant.

Hardware	Pin number	PIC32 Register ID	Pmod Connector
VIN	1		
+3.3V	2		
+5V	3		
SDA(1)	4*	RB09	
RX(1)	5*	RC06	
	6*	J1-08 RC07	
	7*	J1-07 RC08	
	8*	J1-10 RC09	

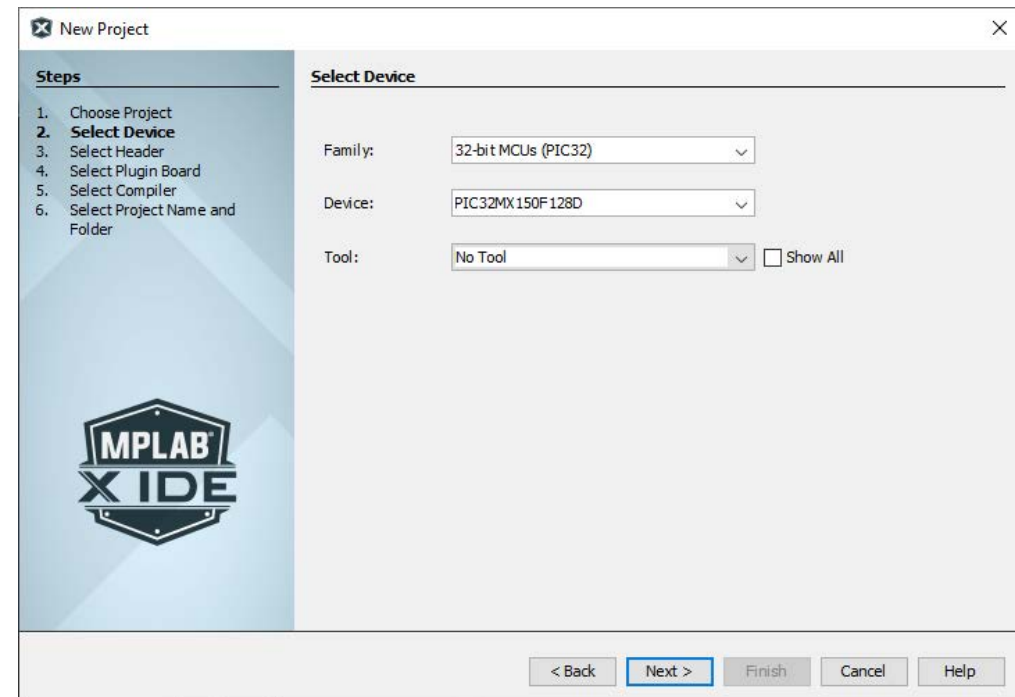


Creating a new project

1. Open MPLAB X
2. Select *File > New Project* (or click )
3. Select *Standalone Project* and click Next

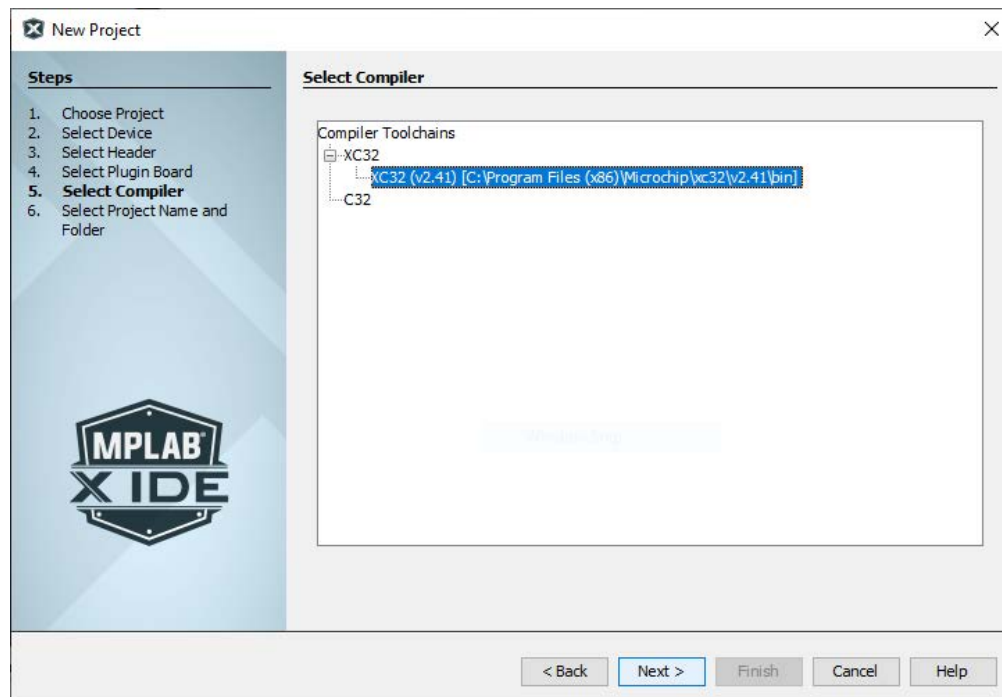


4. Select *32-bit MCUs*
5. Select *PIC32MX150F128D* and click Next



Creating a new project

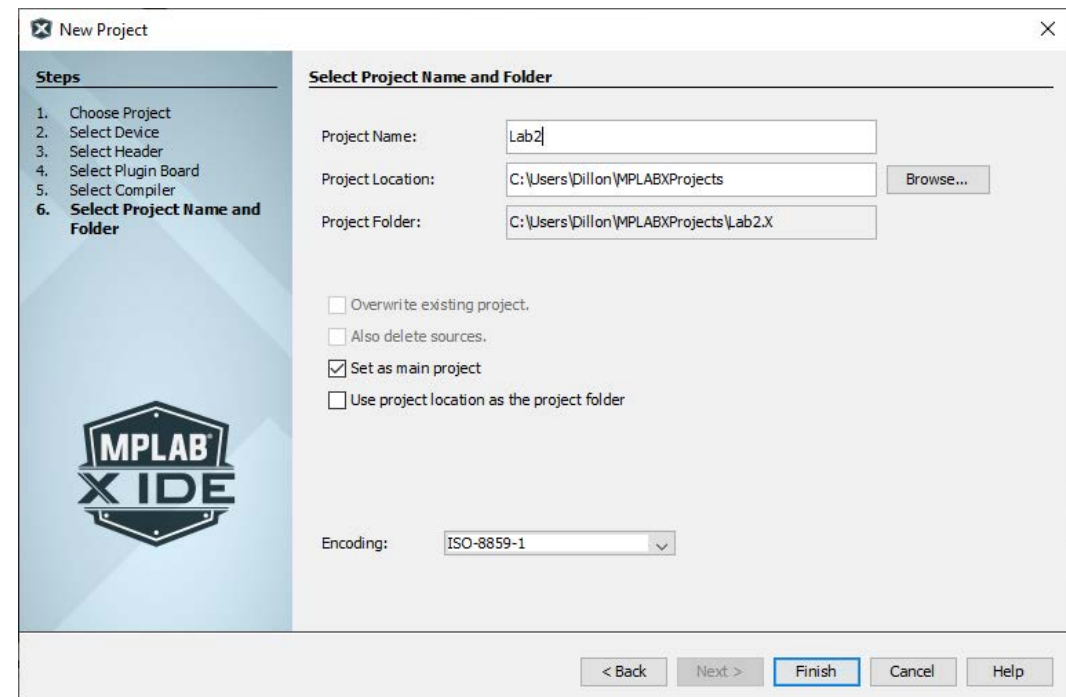
6. Select the XC32 compiler and click Next



7. Name the project

8. Select a save location you can always access

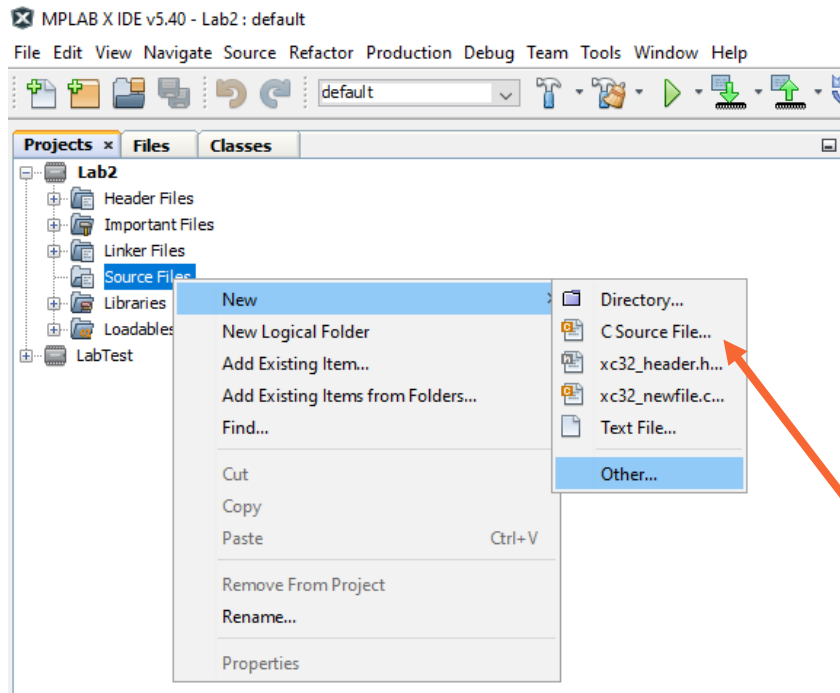
9. Click Finish



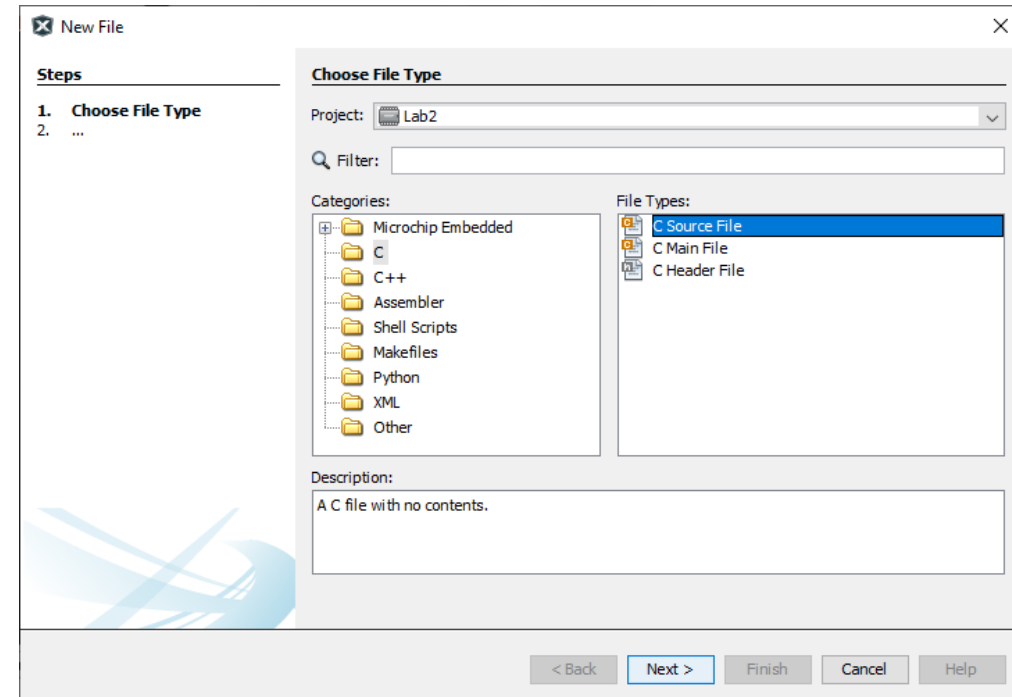
Creating a new project

10. Right-click *Source Files* under project name

11. Select *New > Other...*



12. Select *C* and *C Source File*, then click Next

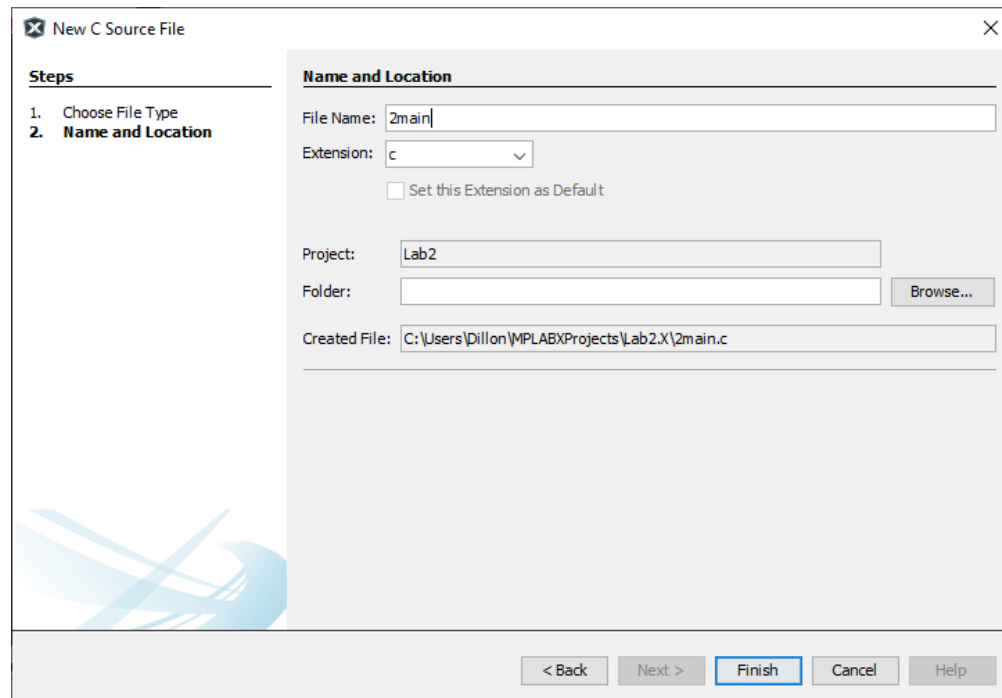


After these steps have been done once,
New > C Source File will become an option.

Creating a new project

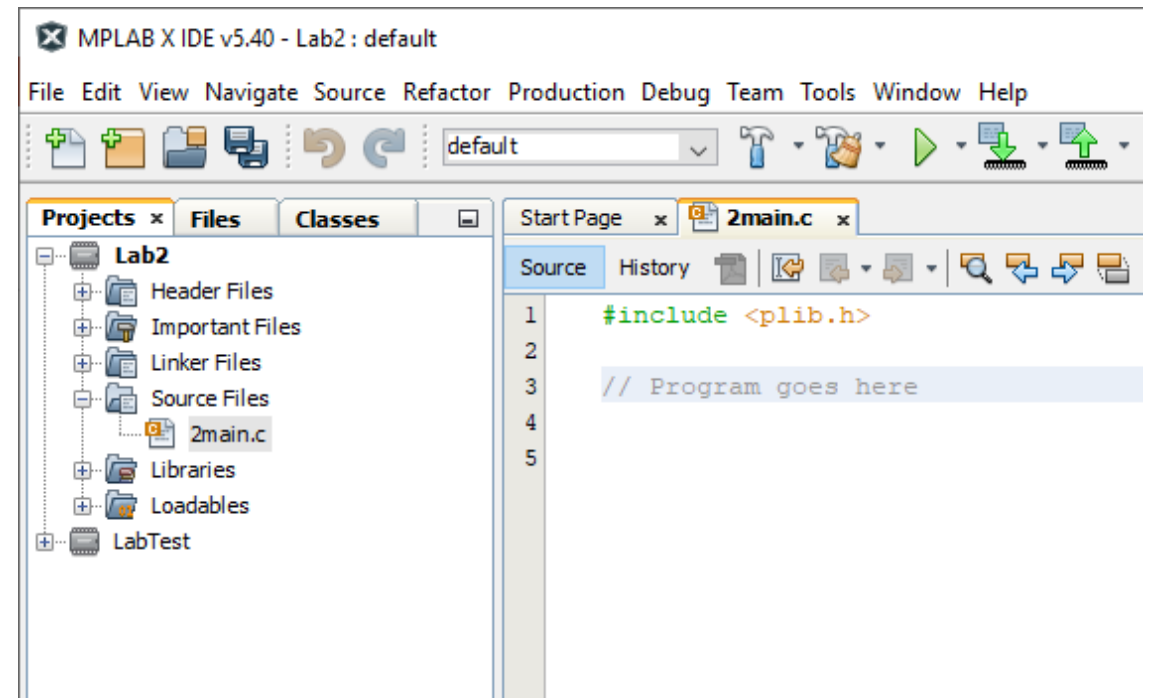
13. Name the source file and click Finish

(It is helpful to give each project's source file a different name.)



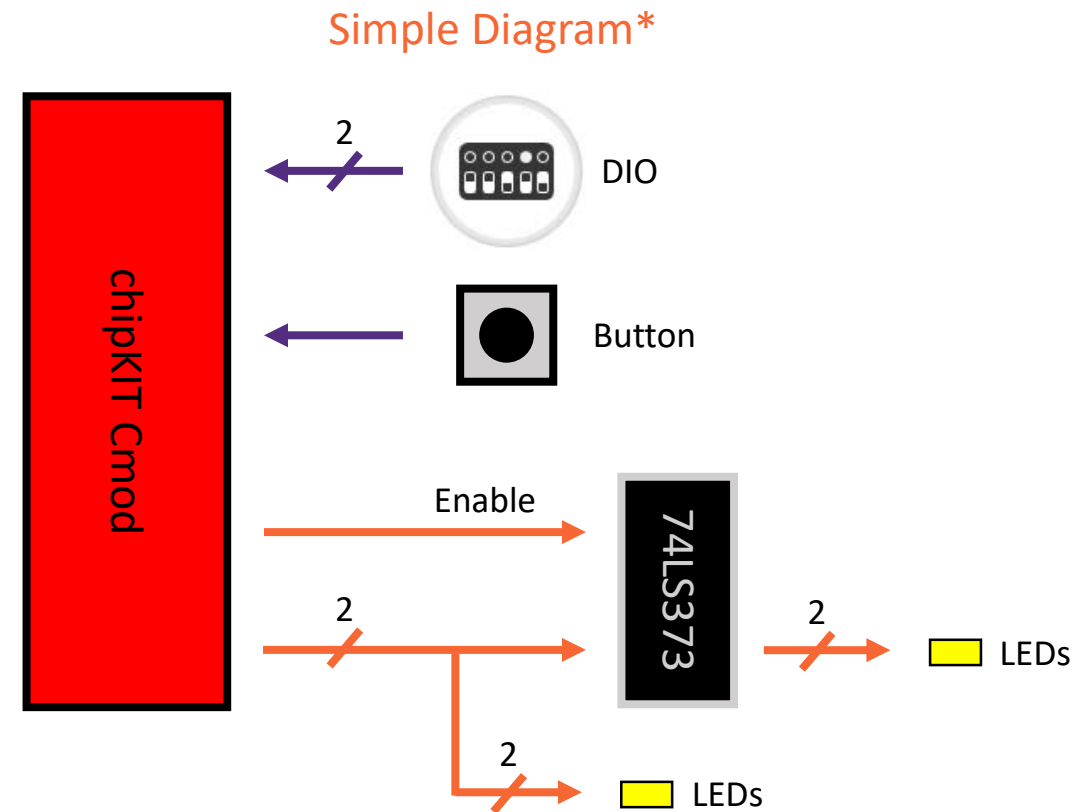
14. Make sure to include *plib.h*

You can now write your program.



Lab Goals

- Read two inputs from DIO with the MC.
- Output those same values from the MC to both the latch and the first pair of LEDs.
- Read an active-low input signal from a button.
- When the button is pressed, output a signal from the MC to enable the latch.
- The second pair of LEDs, connected to the latch outputs, should update to match the inputs when the button is pushed.



*Your pre-lab diagram must include the specific pins to be used, as well as resistors, connections to power and ground, etc.

Lab 3: Comparators

ECE 3720

Preview

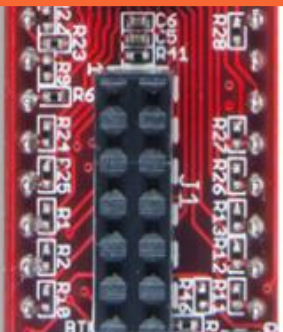
A variable, analog voltage will be produced with a potentiometer and connected to three inputs of the microcontroller. Each of the PIC32's three comparators will be set up to compare this input with a different reference voltage. The states of the comparators' outputs will be indicated by LEDs, such that turning the potentiometer clockwise will light up the LEDs in succession.

Topic	Slide
Analog Capable Pins	<u>3</u>
Potentiometer	<u>4</u>
Comparators	<u>5</u>
Lab Goals	<u>8</u>
Notes	<u>9</u>

Analog Capable Pins

- This is the first lab in which we will use the analog capabilities of the PIC32.
- Only some pins of the PIC32 can read analog voltages.
 - Refer to the pinout diagram in the Cmod reference manual to identify these pins.
 - Note that none of these are 5V tolerant.
- Recall that the ANSELx registers are used to set pins to analog or digital mode.

analogRead()		RB13	A11	13
digitalRead() and digitalWrite()		RA10		14*
digitalRead(), digitalWrite() and analogWrite()		RA07		15*
	SCK(1) J1-04	RB14	A10	16
	SCK(2) J2-10	RB15	A9	17
	AREF	RA00	A0	18
		RA01	A1	19
	T1(2) J2-02	RB00	A2	20



Rei Vilo, 2012-2015

Blue boxes indicate analog capable pins

11.1.2 CONFIGURING ANALOG AND DIGITAL PORT PINS

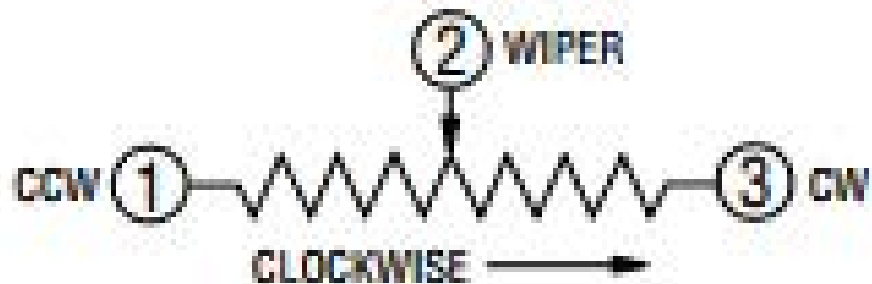
The ANSELx register controls the operation of the analog port pins. The port pins that are to function as analog inputs must have their corresponding ANSEL and TRIS bits set. In order to use port pins for I/O functionality with digital modules, such as Timers, UARTs, etc., the corresponding ANSELx bit must be cleared.

The ANSELx register has a default value of 0xFFFF; therefore, all pins that share analog functions are analog (not digital) by default.

PIC32 Datasheet, pg. 144

Potentiometer

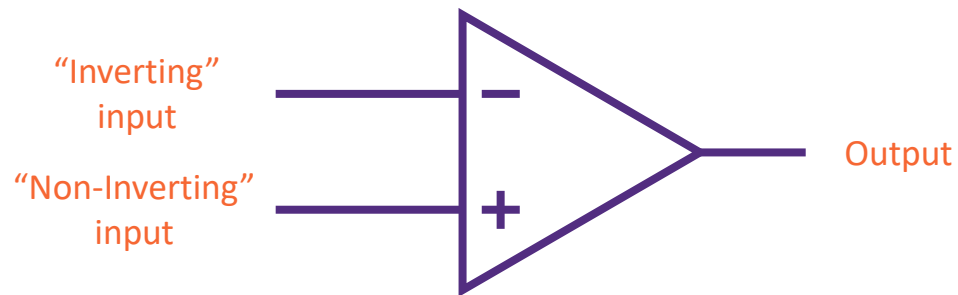
- Variable resistor
 - Ours is a 10 k Ω trimming potentiometer (“trimpot”).
- Three pins
 1. Min voltage (GND)
 2. Output
 3. Max voltage (3.3V)
- Turning the knob clockwise increases resistance between pins 1 and 2.



Notice that pins 1 and 3 are labeled on the top face

Comparators

- Compare two input voltages, outputting logical HIGH or LOW depending on which input is greater
- The PIC32 contains 3 comparators
 - We will use all 3 in this lab, each set up to compare the input from the potentiometer to a different voltage.
 - This will be accomplished by using the registers detailed in the next two slides.



Inputs	Output
Non-Inverting > Inverting	1
Non-Inverting < Inverting	0

Comparator Registers

The following are the registers of interest for this lab. You will need to reference their sections in the datasheet.

- **CMxCON** (datasheet pg. 212)
 - Control registers for each comparator (x is 1, 2, or 3)
 - Used to control the values in and out of the comparators
- **CMSTAT** (pg. 213)
 - Status register that holds the output values from all 3 comparators
- **CVRCON** (pg. 216)
 - Control register used to set the comparator reference voltage (CV_{REF})
 - You will set the values of **CVRSS**, **CVRR**, and **CVR**
 - **We want $CV_{REF} = 0.75 * CV_{RSRC}$**
 - CV_{RSRC} will be the difference between V_{REF+} and V_{REF-} , which you will supply externally.
 - Note that you cannot achieve *exactly* $0.75 * CV_{RSRC}$, but you can get close.

```

CM1CONbits.ON = 1;
CM1CONbits.


|                                     |        |            |
|-------------------------------------|--------|------------|
| <input type="checkbox"/>            | CCH    | __uint32_t |
| <input type="checkbox"/>            | CCH0   | __uint32_t |
| <input type="checkbox"/>            | CCH1   | __uint32_t |
| <input type="checkbox"/>            | COE    | __uint32_t |
| <input type="checkbox"/>            | COUT   | __uint32_t |
| <input type="checkbox"/>            | CPOL   | __uint32_t |
| <input checked="" type="checkbox"/> | CREF   | __uint32_t |
| <input type="checkbox"/>            | EVPOL  | __uint32_t |
| <input type="checkbox"/>            | EVPOL0 | __uint32_t |
| <input type="checkbox"/>            | EVPOL1 | __uint32_t |
| <input type="checkbox"/>            | ON     | __uint32_t |
| <input type="checkbox"/>            | w      | __uint32_t |


```

Comparator Setup

The following are the comparator input combinations used in this lab:

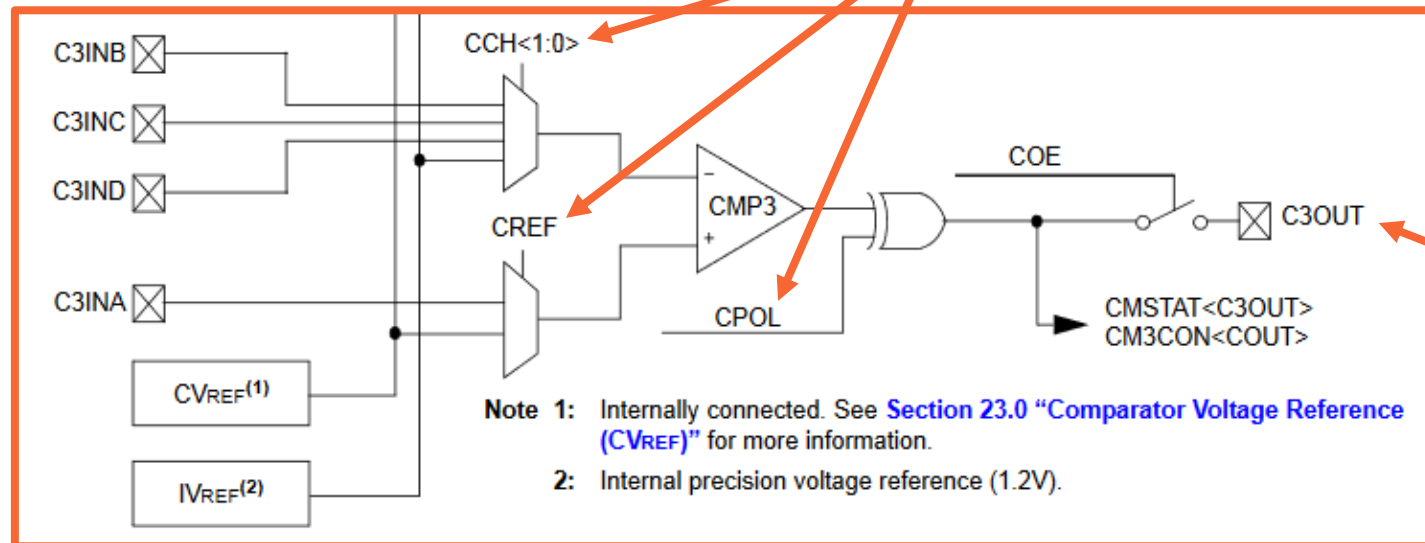
CMP1: IV_{REF} as inverting input and C1INA as non-inverting

CMP2: C2IND as inverting input and VC_{REF} as non-inverting

CMP3: C3IND as inverting input and C3INA as non-inverting

Notice that the values in CMXCON control the multiplexers and the output polarity

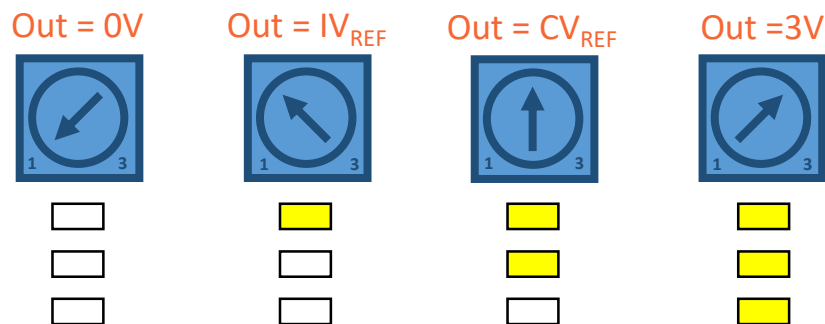
This is only 1/3 of the full diagram in the datasheet.



We can't access this right now, so where else can we find the comparator output?

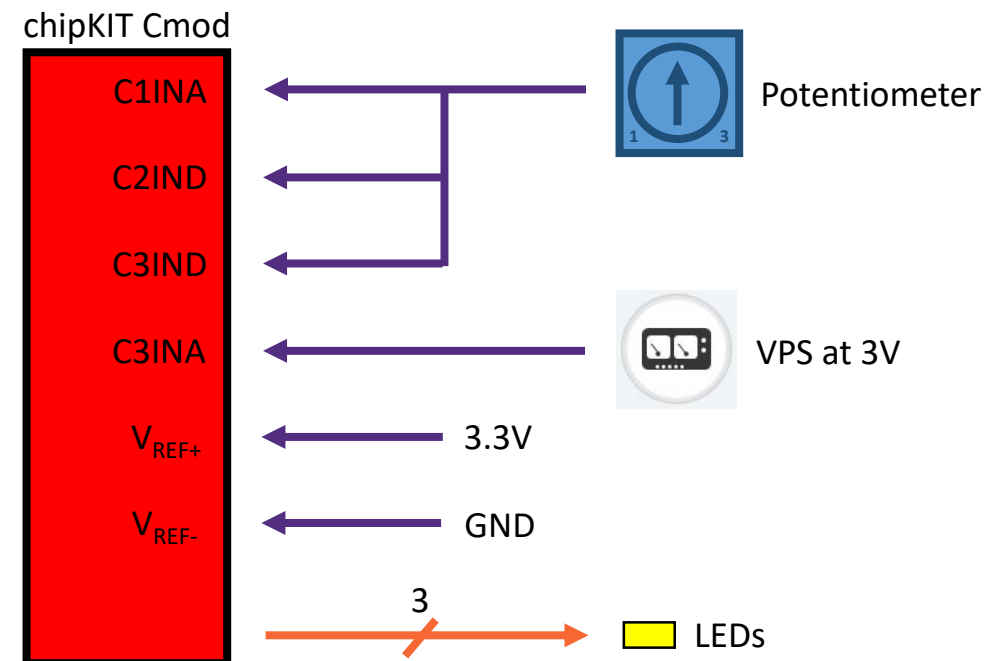
Lab Goals

- Output of the potentiometer will connect to 3 inputs of the MC.
- 3 LEDs will be connected to outputs of the MC, and indicate the statuses of the comparators.
- As the potentiometer is turned clockwise and the output voltage increases, the LEDs turn on in sequence.



*not exact trimpot positions

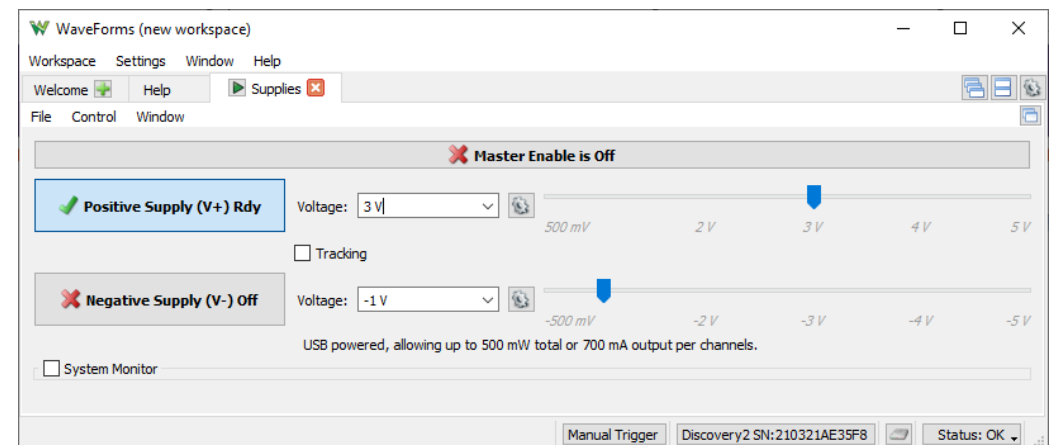
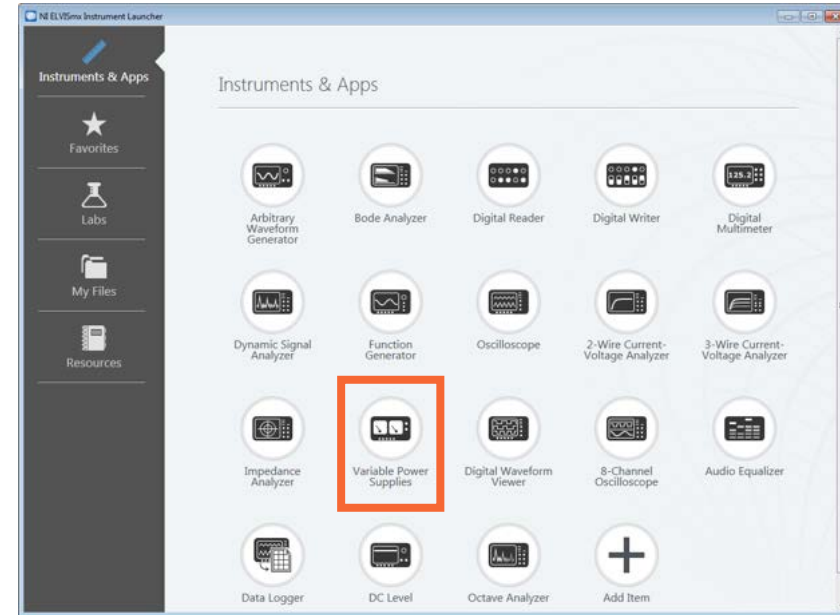
Simple Diagram





Notes

- Recall that the Cmod has a 3.3V output on pin 2.
- C1INA, C3IND, V_{REF+} , etc. correspond to specific pins on the MC. Look them up in the reference manual.
- VPS (variable power supply) on NI-Elvis II:
 - Output located in bottom-left section of Elvis board
 - Controlled through the *NI Instrument Launcher*
- If using the AD2, you can supply 3V to C3INA and power the MC over USB.
- Your code will feature a bunch of register setup before the while loop, then just a few lines inside the loop.
- If your LEDs turn on in the wrong order or at the wrong voltages,
 - Make sure you are connecting the right pins
 - Make sure you are supplying 3V (not 3.3V) to C3INA
 - Check your CVRCON values (CV_{REF} may be incorrect)
 - Comparators' polarities may need to be flipped



Lab 5: Interrupts

ECE 3720

Preview

The MC will count from 0 to 15 as in Lab 1, but with the addition of an interrupt that will turn on all the LEDs for a few seconds. Afterwards, the counting will resume where it left off. The interrupt will be triggered by a debounced button.

Topic	Slide
Interrupts	3
Interrupt Registers	4
ISR	6
Setting up the interrupt	7
Debouncing	8
Lab Goals	9

Interrupts

- When an interrupt is triggered,
 1. MC halts main work and saves state
 2. MC finds and executes interrupt service routine (ISR)
 3. MC restores main state and continues operation
- Possible interrupt sources include external pins, timers, and other peripherals.
- Interrupts remove the need to repeatedly check values while waiting for event.
 - Constantly checking like that is called “polling”, and it’s what we did in Lab 2.
- Once an interrupt is set up, it is handled automatically by the microcontroller
 - You will use registers to tell the MC what triggers the interrupt, and write an ISR to tell the MC what to do when it occurs.
 - You will *not* call the interrupt function at any point.
 - You will *not* check the value of a pin.

PIC32MX1XX/2XX devices generate interrupt requests in response to interrupt events from peripheral modules. The interrupt control module exists externally to the CPU logic and prioritizes the interrupt events before presenting them to the CPU.

PIC32 datasheet, pg. 87

Interrupt Registers

The following are the registers of interest for this lab

- **IECx** (datasheet pg. 92)
 - Used to enable or disable particular interrupts
- **INTCON** (pg. 90)
 - We will use this to set edge polarity of external interrupts
 - If set to rising edge, the interrupt will trigger when the pin transitions to HIGH
- **IFSx** (pg. 92)
 - Interrupt flag status register
 - Every interrupt has a “flag” that goes HIGH when the interrupt is triggered
 - The flag must be cleared before the interrupt can trigger again
- **IPCx** (pg. 93)
 - Interrupt priority control register
 - 1 is highest priority, 7 is lowest
 - A priority of 0 effectively disables the interrupt
 - Sub-priority does not matter for our purposes

Interrupt Bit Locations

- **Table 7-1 (on pg. 88 of the datasheet) provides the necessary information for using each interrupt.**
 - This is an important resource. You will want to refer to it again in future labs.
- Note that the number in the register name may not match the number in the peripheral name.
- The *vector number* identifies the interrupt source, and will be used when writing your ISR.

Interrupt Source ⁽¹⁾	IRQ #	Vector #	Interrupt Bit Location				Persistent Interrupt
			Flag	Enable	Priority	Sub-priority	
Highest Natural Order Priority							
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>	No
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>	No
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>	No
INT0 – External Interrupt	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>	No

PIC32 datasheet, pg. 88

ISR (Interrupt Service Routine)

- Tells the MC what to do when an interrupt occurs
- Use the following macro to identify the interrupt source:
__ISR(vector, ipl)
 - Note that there are two underscores
 - The first argument is the vector number for the interrupt source
 - We can omit the second argument, which sets a priority level
- Write your ISR function, with **__ISR()** between the return type and function name.
 - The interrupt takes no arguments and returns no data, so both are *void*.
 - **Make sure to clear the correct interrupt's flag at the end of the function.**

```
void __ISR(/* vector */) exampleFunction(void)
{
    // code to execute
    // clear flag in IFSX register
}
```

The function name
can be anything

Setting up the interrupt

- You must include the following line to enable interrupts:
INTEnableSystemMultiVectoredInt();
- In this lab we will use External Interrupt 0 (INT0).
 - One of 5 external interrupts on the PIC32
 - INT0 is the only one hard-mapped to a particular pin (find it in the reference manual)
- Use the registers in slide 4 to set the priority and polarity of INT0, then enable it.
 - It's also good practice to clear the interrupt's flag when setting it up.
- Then write your ISR as shown on the previous slide.
 - Make sure you don't put it inside your main function.

INT0 is mapped to this pin on the PIC32, but it has a different number on the chipKIT Cmod

PPS (Peripheral Pin Select) indicates that the peripheral can be mapped to a variety of pins. This will be covered in Lab 6.

TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

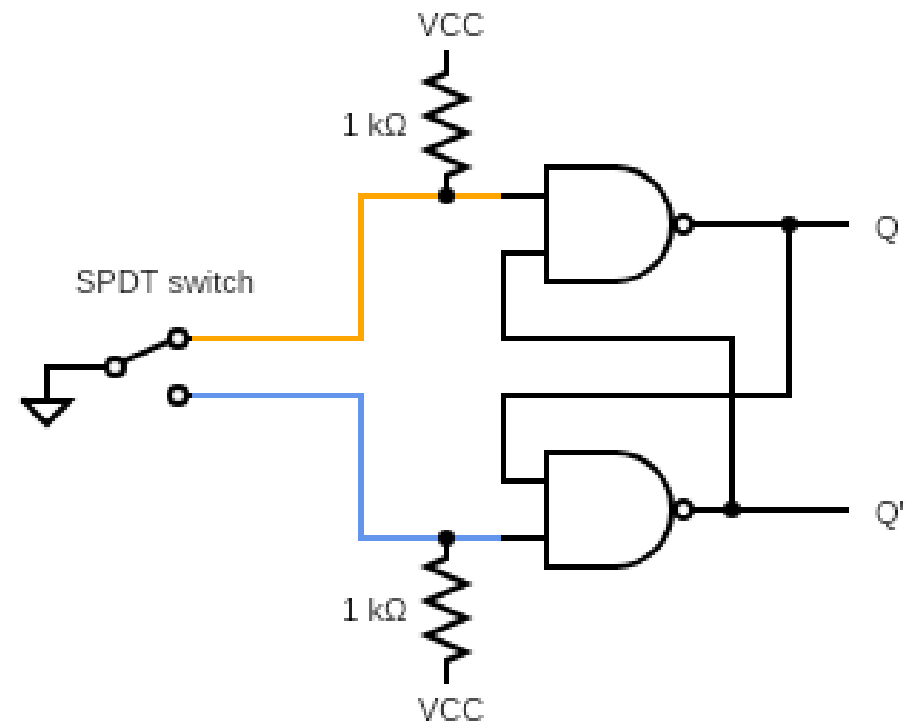
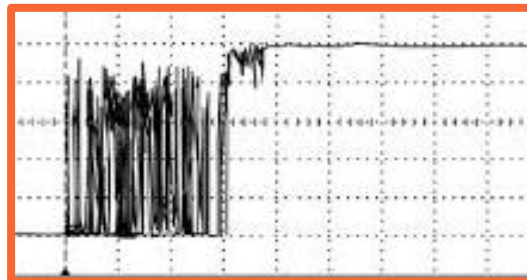
Pin Name	Pin Number ⁽¹⁾				Pin Type	Buffer Type	Description
	28-pin QFN	28-pin SSOP/SPDIP/SOIC	36-pin VTLA	44-pin QFN/TQFP/VTLA			
INT0	13	16	17	43	I	ST	External Interrupt 0
INT1	PPS	PPS	PPS	PPS	I	ST	External Interrupt 1
INT2	PPS	PPS	PPS	PPS	I	ST	External Interrupt 2
INT3	PPS	PPS	PPS	PPS	I	ST	External Interrupt 3
INT4	PPS	PPS	PPS	PPS	I	ST	External Interrupt 4

PIC32 datasheet, pg. 21

Debouncing

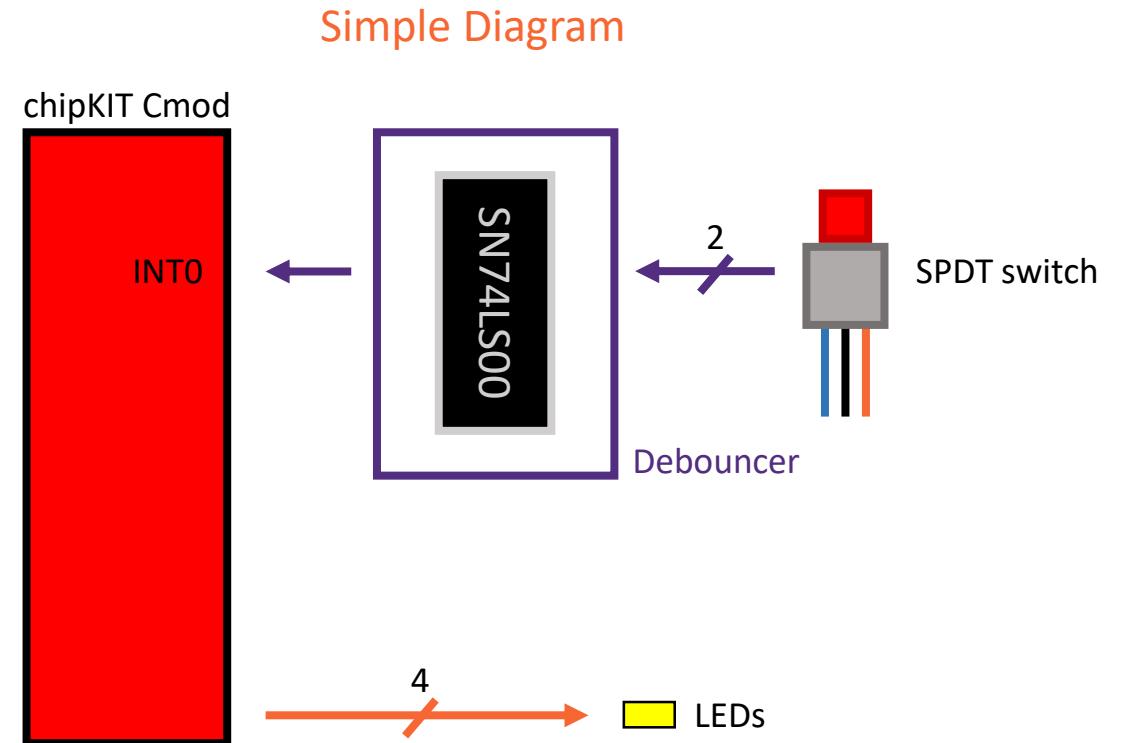
- Switches have a tendency to bounce rapidly between HIGH and LOW when first toggled.
 - This can be addressed with both hardware and software
- We will implement the debouncer seen at right using the [SN74LS00](#) and a SPDT (single pole, double throw) switch.
 - This is an example of an SR (set-reset) latch
 - $Q' = !Q$
 - When the switch transitions between throws, the pull-up resistors and NAND gates ensure the output remains unchanged it settles.
- Connect one of the outputs to INTO
 - You can wire both outputs to LEDs first to check their behavior

Switch output bouncing
between LOW and HIGH



Lab Goals

- Recreate Lab 1, but with an interrupt that causes all the lights to turn on and stay on for a few seconds.
- The interrupt should be triggered by the debounced output of the SPDT switch.
- When the interrupt ends, the MC should resume counting where it left off.



Lab 6: Peripheral Pin Select

ECE 3720

Preview

Peripheral Pin Select will be used to enable a second external interrupt. The two outputs of a rotary encoder will trigger the interrupts, causing a count variable, displayed on LEDs, to increase or decrease depending on the direction of rotation.

Topic	Slide
Peripheral Pin Select (PPS)	3
Using PPS	4
Grayhill 61C Optical Encoder	6
Lab Goals	7
Determining Direction of Rotation	8

Peripheral Pin Select (PPS)

- Peripheral Pin Select allows the inputs and outputs of certain peripherals (such as timers or external interrupts) to be mapped to various external pins.
- Recall that in Lab 5 we only used the one interrupt already mapped to a pin.
- In this lab, you will use PPS to access an additional external interrupt.
- See section 11.3 (pg. 145) of the PIC32 datasheet for details on PPS.

TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number ⁽¹⁾				Pin Type	Buffer Type	Description
	28-pin QFN	28-pin SSOP/SPDIP/SOIC	36-pin VTLA	44-pin QFN/TQFP/VTLA			
INT0	13	16	17	43	I	ST	External Interrupt 0
INT1	PPS	PPS	PPS	PPS	I	ST	External Interrupt 1
INT2	PPS	PPS	PPS	PPS	I	ST	External Interrupt 2
INT3	PPS	PPS	PPS	PPS	I	ST	External Interrupt 3
INT4	PPS	PPS	PPS	PPS	I	ST	External Interrupt 4

PIC32 datasheet, pg. 21

The Peripheral Pin Select (PPS) configuration provides an alternative to these choices by enabling peripheral set selection and their placement on a wide range of I/O pins. By increasing the pinout options available on a particular device, users can better tailor the device to their entire application, rather than trimming the application to fit the device.

PIC32 datasheet, pg. 145

Using PPS (method 1)

- You can remap pins with the following macros:
 - PPSInput(grp, fn, pin)**
 - PPSOutput(grp, pin, fn)**
- grp** (“group”) refers to the box in column 4 of table 11-1 or 11-2.
- fn** (“function”) is the name of the peripheral
 - Col. 1 of table 11-1 or col. 4 of table 11-2
- pin** is the name of the pin
 - Col. 4 of table 11-1 or col. 1 of table 11-2
 - RP in RPxx stands for “remappable pin”

```
PPSInput(1, T2CK, RPC0); // map Timer 2 ext input to C0
PPSOutput(1, RPB3, U1TX); // map UART 1 transmit output to B3
```

TABLE 11-1: INPUT PIN SELECTION

Peripheral Pin	[pin name]R SFR	[pin name]R bits	[pin name]R Value to RPin Pin Selection
INT4	INT4R	INT4R<3:0>	0000 = RPA0 0001 = RPB3 0010 = RPB4 0011 = RPB15 0100 = RPB7 0101 = RPC7 ⁽²⁾ 0110 = RPC0 ⁽¹⁾ 0111 = RPC5 ⁽²⁾ 1000 = Reserved .
T2CK	T2CKR	T2CKR<3:0>	.
IC4	IC4R	IC4R<3:0>	.
SS1	SS1R	SS1R<3:0>	.
REFCLKI	REFCLKIR	REFCLKIR<3:0>	1111 = Reserved
INT3	INT3R	INT3R<3:0>	0000 = RPA1 0001 = RPB5 0010 = RPB1 0011 = RPB11 0100 = RPB8 0101 = RPA8 ⁽²⁾ 0110 = RPC8 ⁽²⁾ 0111 = RPA9 ⁽²⁾ 1000 = Reserved .
T3CK	T3CKR	T3CKR<3:0>	.
IC3	IC3R	IC3R<3:0>	.
U1CTS	U1CTSR	U1CTSR<3:0>	.
U2RX	U2RXR	U2RXR<3:0>	.
SDI1	SDI1R	SDI1R<3:0>	1111 = Reserved
INT2	INT2R	INT2R<3:0>	0000 = RPA2 0001 = RPB6 0010 = RPA4 0011 = RPB13 0100 = RPB2 0101 = RPC6 ⁽²⁾ 0110 = RPC1 ⁽¹⁾ 0111 = RPC3 ⁽¹⁾ 1000 = Reserved .
T4CK	T4CKR	T4CKR<3:0>	.
IC1	IC1R	IC1R<3:0>	.
IC5	IC5R	IC5R<3:0>	.
U1RX	U1RXR	U1RXR<3:0>	.
U2CTS	U2CTSR	U2CTSR<3:0>	.
SDI2	SDI2R	SDI2R<3:0>	.
OCFB	OCFBR	OCFBR<3:0>	1111 = Reserved
INT1	INT1R	INT1R<3:0>	0000 = RPA3 0001 = RPB14 0010 = RPB0 0011 = RPB10 0100 = RPB9 0101 = RPC9 ⁽¹⁾ 0110 = RPC2 ⁽²⁾ 0111 = RPC4 ⁽²⁾ 1000 = Reserved .
T5CK	T5CKR	T5CKR<3:0>	.
IC2	IC2R	IC2R<3:0>	.
SS2	SS2R	SS2R<3:0>	.
OCFA	OCFAR	OCFAR<3:0>	1111 = Reserved

Group 1 (rows 1-4)

Group 2 (rows 5-8)

Group 3 (rows 9-12)

Group 4 (rows 13-16)

Using PPS (method 2)

- The second method is explained in sections 11.3.4 and 11.3.5 of the datasheet (pgs. 145-148).
- Refer to tables 11-1 and 11-2. To map a peripheral's input/output to a pin, write the corresponding value from column 4 to the control register in column 3.

TABLE 11-1: INPUT PIN SELECTION

Peripheral Pin	[pin name]R SFR	[pin name]R bits	[pin name]R Value to RPN Pin Selection
INT4	INT4R	INT4R<3:0>	0000 = RPA0 0001 = RPB3 0010 = RPB4
T2CK	T2CKR	T2CKR<3:0>	0011 = RPB15 0100 = RPB7 0101 = RPC7 ⁽²⁾ 0110 = RPC0 ⁽¹⁾ 0111 = RPC5 ⁽²⁾
SS1	SS1R	SS1R<3:0>	1000 = Reserved . . .
REFCLKI	REFCLKIR	REFCLKIR<3:0>	1111 = Reserved

PIC32 datasheet, pg. 146

These are just the first sections of each table. Look in the datasheet for the full tables.

Example usage

```
T2CKR = 0b0110; // map Timer 2 ext input to C0
RPB3R = 0b0001; // map UART 1 transmit output to B3
```

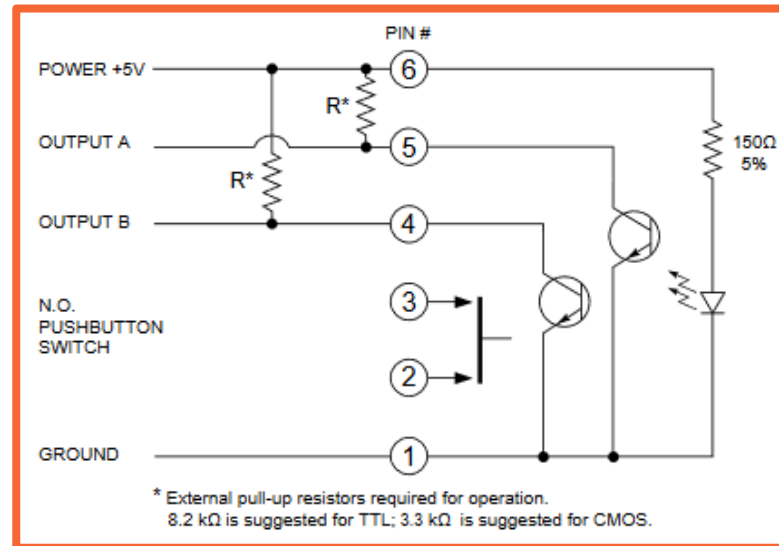
TABLE 11-2: OUTPUT PIN SELECTION

RPn Port Pin	RPnR SFR	RPnR bits	RPnR Value to Peripheral Selection
RPA0	RPA0R	RPA0R<3:0>	0000 = No Connect 0001 = U1TX 0010 = U2RTS
RPB3	RPB3R	RPB3R<3:0>	0011 = SS1
RPB4	RPB4R	RPB4R<3:0>	0100 = Reserved
RPB15	RPB15R	RPB15R<3:0>	0101 = OC1
RPB7	RPB7R	RPB7R<3:0>	0110 = Reserved
RPC7	RPC7R	RPC7R<3:0>	0111 = C2OUT
RPC0	RPC0R	RPC0R<3:0>	1000 = Reserved . . .
RPC5	RPC5R	RPC5R<3:0>	1111 = Reserved

PIC32 datasheet, pg. 148

Grayhill 61C Optical Encoder

- Datasheet available [here](#)
- Used to track rotational position
- Pay close attention to the diagram and table on page 2 of the datasheet.
 - Some of the resistors shown are internal and some are external.
 - Notice how the two outputs represent position. One of them changes with each tick of the dial.



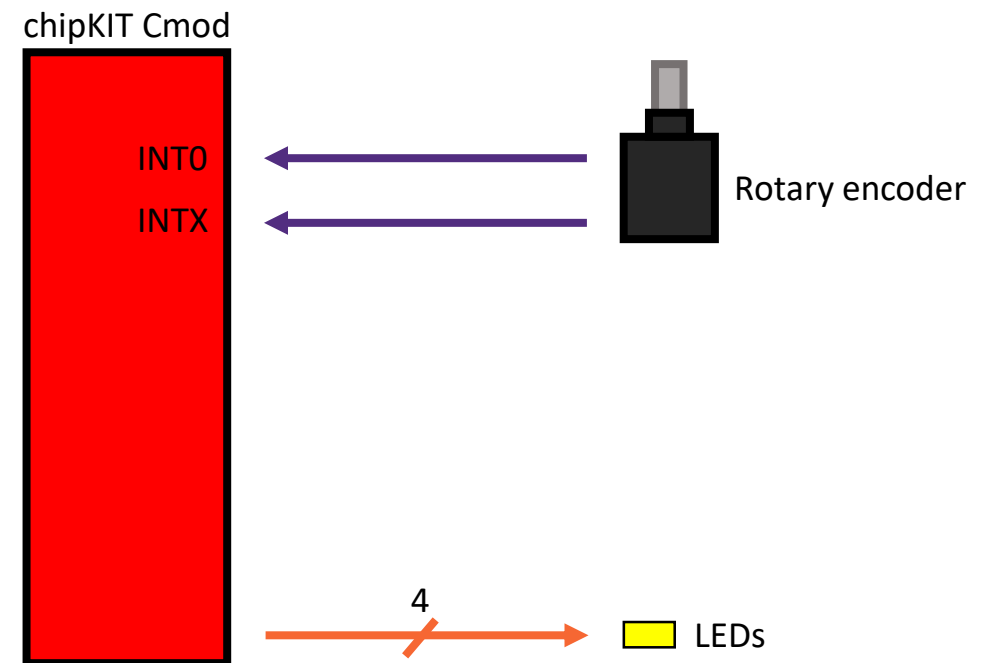
61C datasheet, pg. 2



Lab Goals

- Like Labs 1 and 5, a variable will count between 0 and 15 and be displayed on 4 LEDs.
 - The count should loop from 15 back to 0 and from 0 up to 15.
 - Unlike Labs 1 and 5, the count will not change automatically.
- Use PPS to enable a second interrupt.
 - Remember to pay attention to 5V tolerance
- Connect each output of the encoder to one interrupt.
- Increment the count on CW rotations and decrement on CCW rotations*.
 - Only modify the count in the ISRs
 - This means the count variable must be global
 - Do not save the previous state of the encoder

Simple Diagram



*Make sure to read the next slide for details on how to determine the direction of rotation.

Determining Direction of Rotation

- Pay close attention to the table from the 61C datasheet to determine whether to increment or decrement count.
- Each encoder output will have its own interrupt and ISR, which will be triggered any time the output changes.
 - Remember to clear the flag at the end of the ISR.
- Since the rotations cause both rising and falling edges, the interrupts' polarities will have to change accordingly.
 - e.g., if an interrupt was just triggered by a rising edge, it must next watch for a falling edge.
 - You will need to modify the interrupt's polarity from inside the ISR.
- For example, suppose the interrupt for output A is triggered:
 - You can read A and B to get the current position.
 - You know A just changed, so you can determine the previous position.
 - Then you have enough information to modify count and update the polarity.

Clockwise Rotation		
Position	Output A	Output B
1		
2	●	
3	●	●
4		●

● Indicates logic high; blank indicates logic low. Code repeats every 4 positions.

61C datasheet, pg. 2

This is an example of gray code, since successive values always differ by only one bit.

Lab 7: Timers

ECE 3720

Preview

One of the microcontroller's timers will be set up and used to trigger an interrupt at frequencies corresponding to musical notes. The interrupt will toggle the digital output to a piezo buzzer, causing it to produce the desired notes, and play a song.

Topic	Slide
Timers	3
Timer Registers	4
Timer Diagram	5
Piezo Buzzer	6
Code	7
Lab Goals	8
Notes	9

Timers

- The PIC32's timers operate by counting up on every cycle of the clock. When a certain value is reached, an interrupt can be triggered, and the counting resets.
- Timer 1 (type A)
 - Datasheet pg. 151
 - 16-bit (so max count value is 0xFFFF)
 - Includes real-time clock functionality (not used in this lab)
- Timers 2-5 (type B)
 - Datasheet pg. 155
 - Each individual timer is 16-bit
 - Timers 2-3 and timers 4-5 can be combined to form 32-bit timers
 - Even-numbered timer supplies the control logic
 - Odd-numbered timer supplies the interrupt

Two 32-bit synchronous timers are available by combining Timer2 with Timer3 and Timer4 with Timer5. The 32-bit timers can operate in three modes:

- Synchronous internal 32-bit timer
- Synchronous internal 32-bit gated timer
- Synchronous external 32-bit timer

Note: In this chapter, references to registers, TxCON, TMRx and PRx, use 'x' to represent Timer2 through Timer5 in 16-bit modes. In 32-bit modes, 'x' represents Timer2 or Timer4 and 'y' represents Timer3 or Timer5.

PIC32 datasheet, pg. 156

Timer Registers

The following are the registers of interest for this lab. The diagram on the next slide shows how they are used.

- **TxCON** (datasheet pg. 152/157)
 - Timer control register
 - Enable/disable timer
 - Select clock source
 - Select prescaler value
- **TMRx**
 - Holds the timer's running count value
 - Increments on each clock cycle (or every few cycles, depending on prescaler)
 - **You do not need to write to this register.**
- **PRx**
 - Holds value the timer should count to before resetting
 - **You will set this value to control how frequently the timer interrupt occurs.**
 - This value should only be changed in the ISR, or when timer is disabled.

Timer Diagram

FIGURE 13-1: TIMER2-TIMER5 BLOCK DIAGRAM (16-BIT)

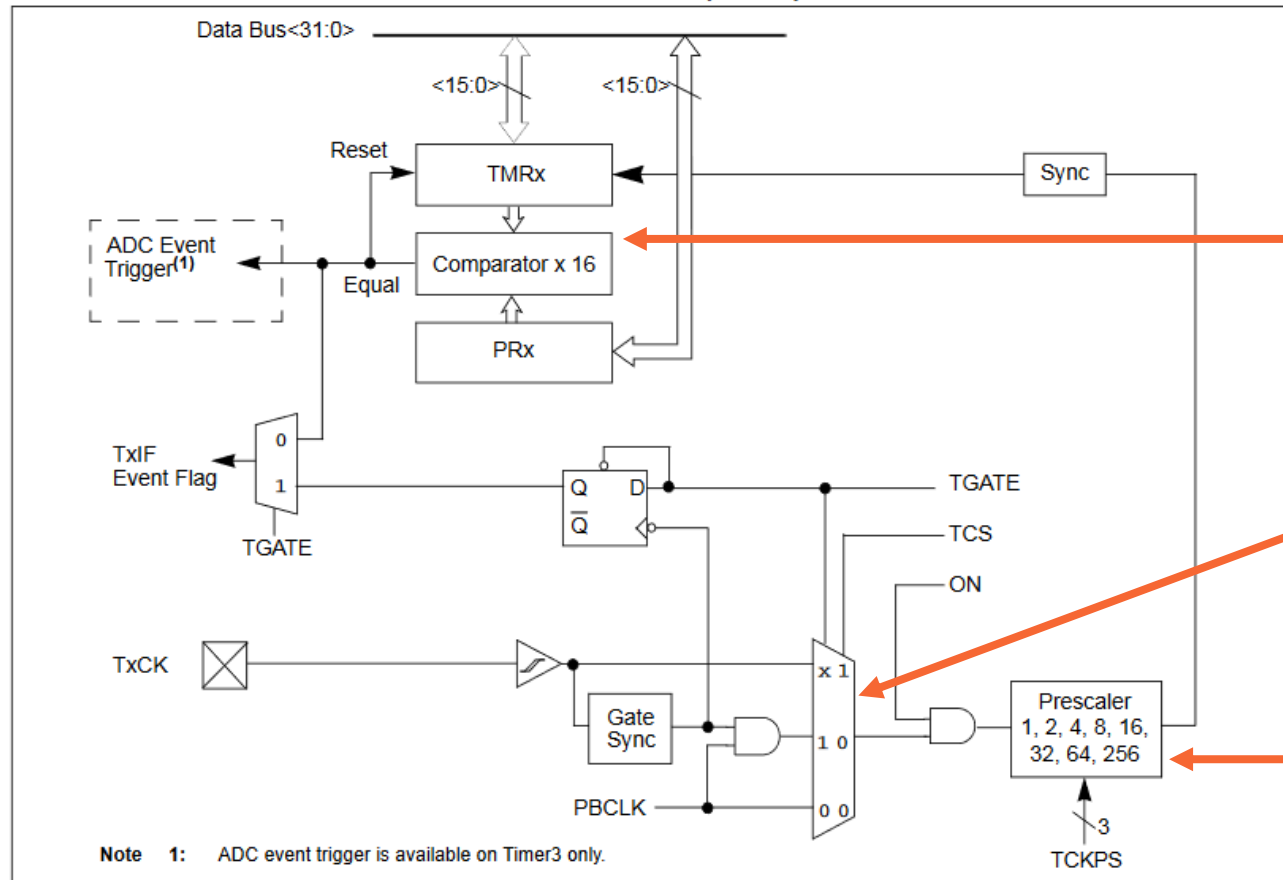


Diagram for timers 2-5 is shown

- Timer 1's diagram is on pg. 151.
- The 32-bit timer diagram is on pg. 156.

When $TMRx == PRx$,

- TMRx gets reset
- Corresponding timer interrupt is triggered

Select clock source

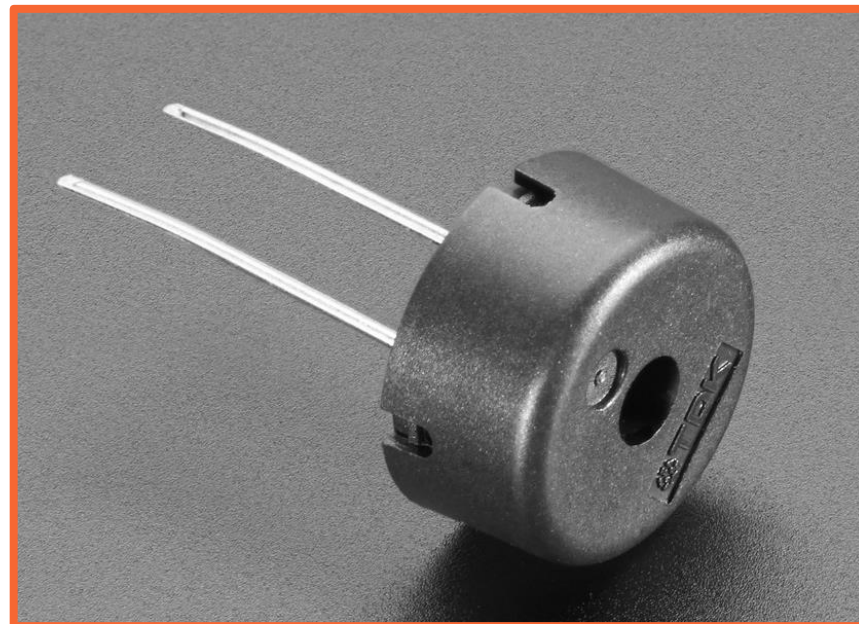
- We'll use the Peripheral Bus Clock (PBCLK), which is derived from the built-in system clock.
- The alternative would be connecting an external source (TxCK).

Prescaler divides the clock frequency

- e.g., a prescaler value of 4 would cause the timer to count 4 times slower than the clock.

Piezo Buzzer

- A piezoelectric material changes shape when exposed to an electric field.
- Changing the voltage across the buzzer at a high frequency causes the material inside to vibrate at the same frequency, producing a note.
- **We will use timers to toggle an output voltage on and off at specific frequencies in order to play a song.**



Code

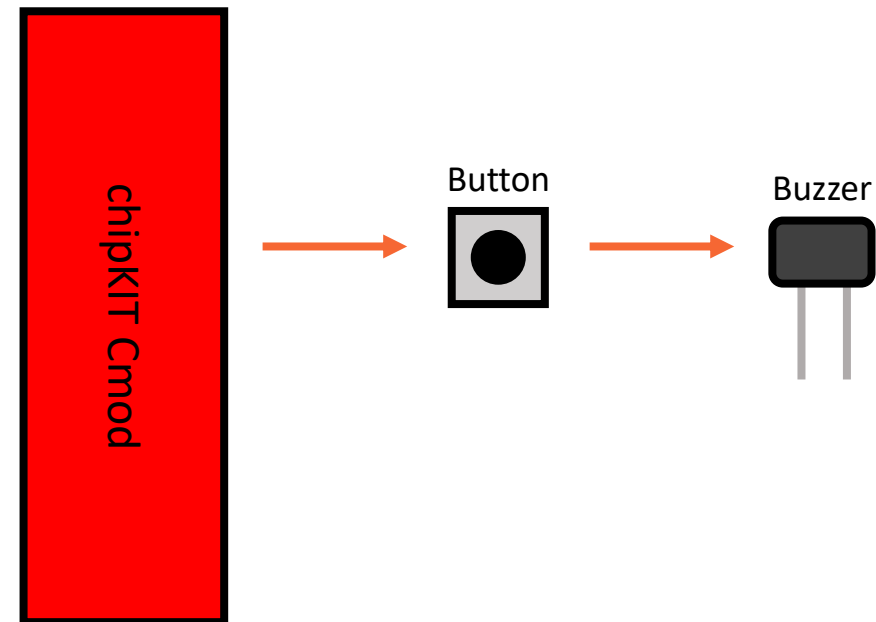
- For this lab you are provided a skeleton code (main_skeleton.c) which you will complete.
- The definitions at the top of the program (a through CC) represent the periods of notes, in terms of PIC32 clock cycles.
 - Recall that period is the inverse of frequency.
 - See the “notes” slide for info on how these are calculated.
- Definitions q through $edot$ represent note lengths. q is a quarter note, e is an eighth note, etc.
- Notice you are given two arrays
 - $delay$ is full of note lengths
 - $music_notes$ contains notes in the order they are to be played
- **You do not need to change anything inside the $while(1)$ loop**
 - The if statement steps through the notes of the song, remaining on each note until j reaches the value of $delay[i]$.
 - Notice that j is not changed anywhere in the given code. You must increment j somewhere.

```
#define r 3000
#define a 4545
#define b 4050
#define C 3817
#define C_ 3610
#define D 3402
#define D_ 3216
#define E 3031
#define F 2866
#define F_ 2703
#define G 2551
#define G_ 2410
#define A 2273
#define A_ 2146
#define B 2025
#define CC 1911
#define q 400
#define qdot q * 1.5
#define e q/2
#define s e/2
#define t32 s/2
#define sdot s+t32
#define h q*2
#define hdot q+e
#define edot e+s
#define num_notes 52
```

Lab Goals

- Connect the buzzer to an output of the MC with a button in between so it only makes sound when the button is pressed.
- Set up a timer using TxCON.
- Set up an interrupt triggered by your chosen timer.
- Use the interrupt to toggle the output (when you want to play sound). This will produce a note dependent on how frequently the interrupt occurs.
- The interrupt should also update the note being played.

Simple Diagram





Notes

- Middle C example:
 - Middle C has a frequency of 261.6 Hz
 - MC clock runs at ~2 MHz
 - $2\text{M cycles/s} * 1\text{s}/261.6 = 7645$ cycles per middle C period
 - Divide by 2 to get cycles per inversion \rightarrow `#define C 3817`
- *r*, which appears in *music_notes*, represents a rest. No sound should be played when this is the current note.
- Consider the possible values to be loaded into PRx. Will a 32-bit mode timer be necessary?

Lab 8: Pulse Width Modulation

ECE 3720

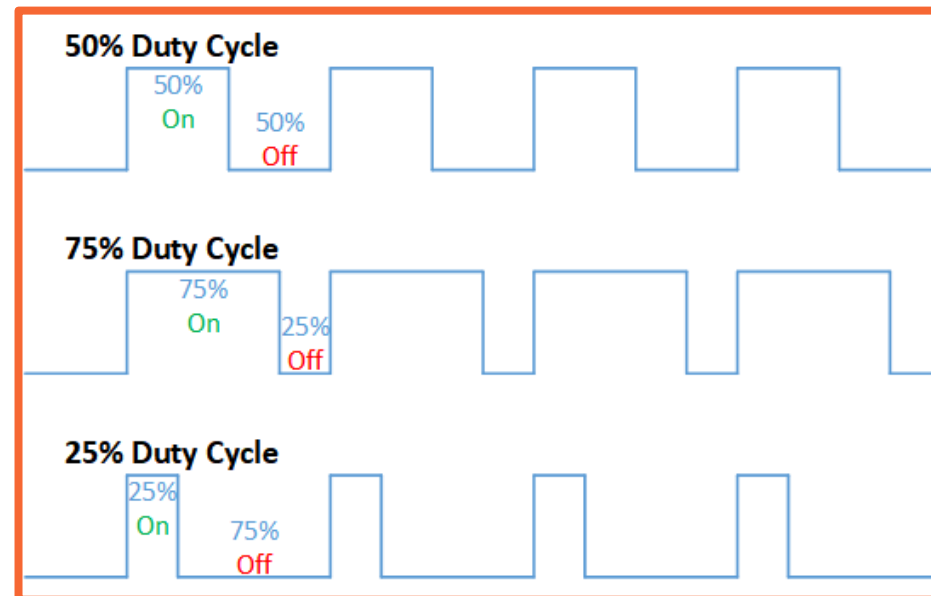
Preview

The output compare peripheral will be used to produce a PWM signal to control the speed of a motor via a motor driver. A button-triggered interrupt will be used to select the duty cycle of the PWM signal.

Topic	Slide
Pulse Width Modulation (PWM)	3
Output Compare (OC)	4
OC Diagram	5
Using OC for PWM	6
L293DNE	7
Lab Goals	8

Pulse Width Modulation (PWM)

- By turning a digital output on and off in a regular pattern, an average voltage can be produced that is proportional to the percentage of time the output is HIGH.
 - The percentage is called the duty cycle, as seen below.
- This is used to control the speed of DC motors, the position of servos, the brightness of LEDs, etc.



Output Compare (OC)

- The PIC32 does not have a dedicated PWM peripheral. However, the output compare peripheral can be used to generate PWM signals.
- Output compare works with the MC's timers to trigger an event at a specified point the timer's cycle.
 - Recall that the timer counts up in the TMRx register until it reaches the value in PRx.
 - Similarly, output compare triggers an event when TMRx matches OCxR

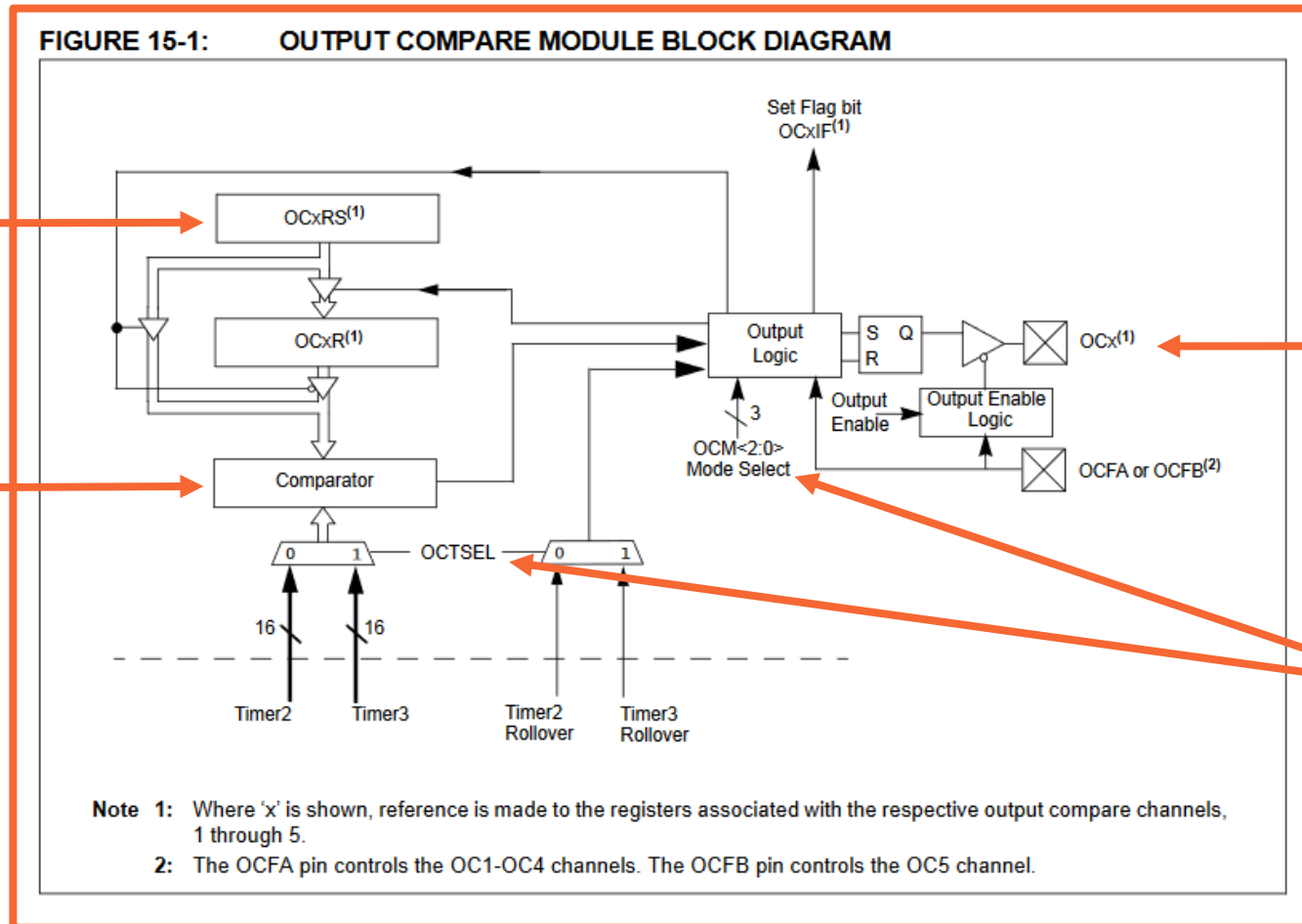
Registers:

- **OCxCON** (datasheet pg. 164)
 - Used to enable OC, select the timer, and select the mode of operation
- **OCxR**
 - Holds the value to be compared to TMRx
 - **Similar to PRx, this register should not be written to while the timer is running**
- **OCxRS**
 - Copies its value to OCxR when previous PWM cycle completes
 - **You should write the desired OCxR value here**

The Output Compare module is used to generate a single pulse or a train of pulses in response to selected time base events. For all modes of operation, the Output Compare module compares the values stored in the OCxR and/or the OCxRS registers to the value in the selected timer. When a match occurs, the Output Compare module generates an event based on the selected mode of operation.

PIC32 datasheet, pg. 163

Output Compare Diagram



OCxRS is copied to OCxR

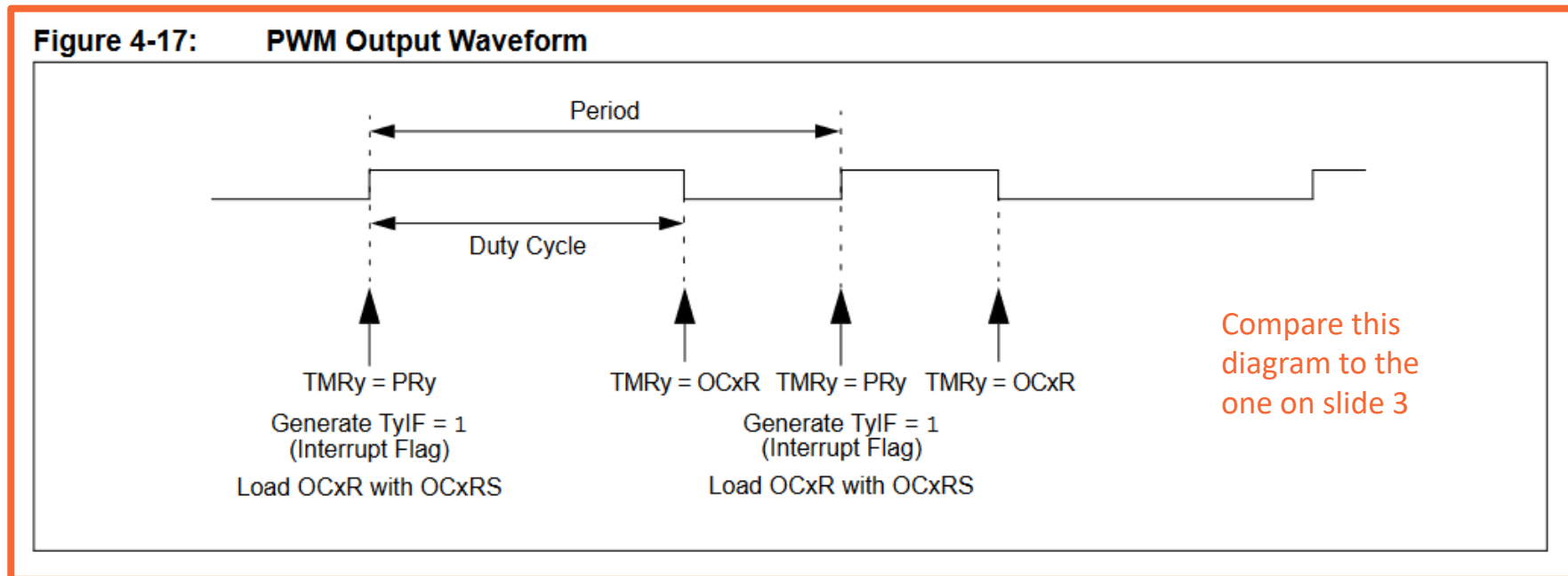
OCxR is compared to timer value

OC output can be mapped to a pin with PPS

Values in OCxCON select timer and output logic

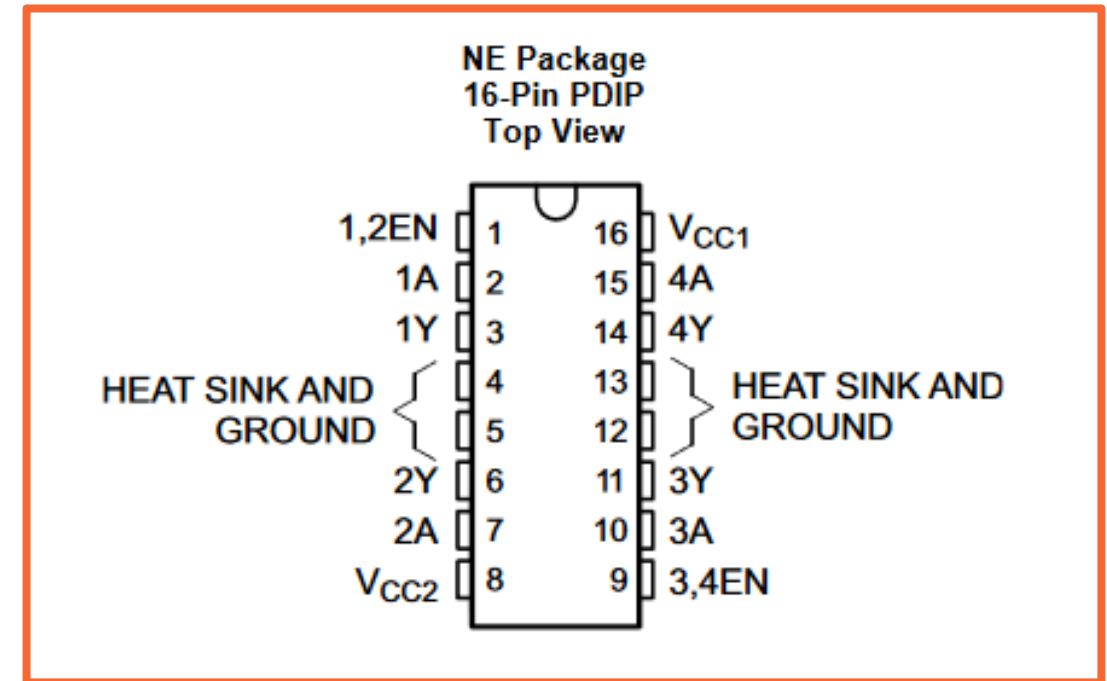
Using OC for PWM

- The OC's PWM mode can be selected in OCxCON. Note that we do not need fault detection for this lab.
- Use peripheral pin select (covered in Lab 6) to map OC's output to a pin.
- The diagram below illustrates how the timer and OC registers are utilized in PWM application.



L293DNE

- Datasheet available [here](#)
- Rather than attempt to drive a motor with the PIC32's limited output power, we use a motor driver IC.
- Power for the motor is supplied to the driver, while the MC provides the PWM input that will control the driver output and the motor's speed.
- **You will want to reference the pin descriptions on page 3 of the L293 datasheet, and the functional block diagram on page 7.**

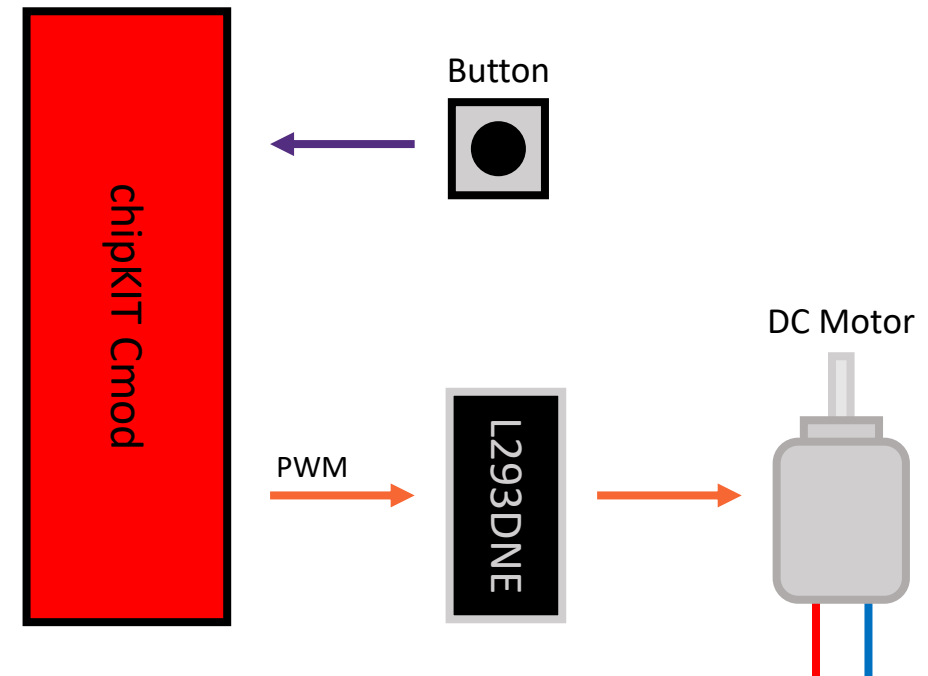


L293DNE datasheet, pg. 5

Lab Goals

- Set up the output compare and timer peripherals to output a PWM signal to the L293DNE.
 - OC should not be enabled before the timer.
 - Neither should be enabled until after its registers are set up.
- Wire the driver and motor as shown on page 7 of the L293 datasheet.
 - This will require the use of a diode.
- Set up an external interrupt to be triggered by a button.
- On each press of the button, the PWM duty cycle should increase by 25% .
 - When the duty cycle is at 100%, it should next go back to 0%.
 - Note that a duty cycle of 25% likely will not cause the motor to turn, but you should be able to hear it attempting to do so.

Simple Diagram



Lab 9: Serial Peripheral Interface

ECE 3720

Preview

The PIC32's SPI module will be used to send data serially to an external shift register, which will display its value on LEDs. Each time a button is pressed, the MC will send a new value to the shift register.

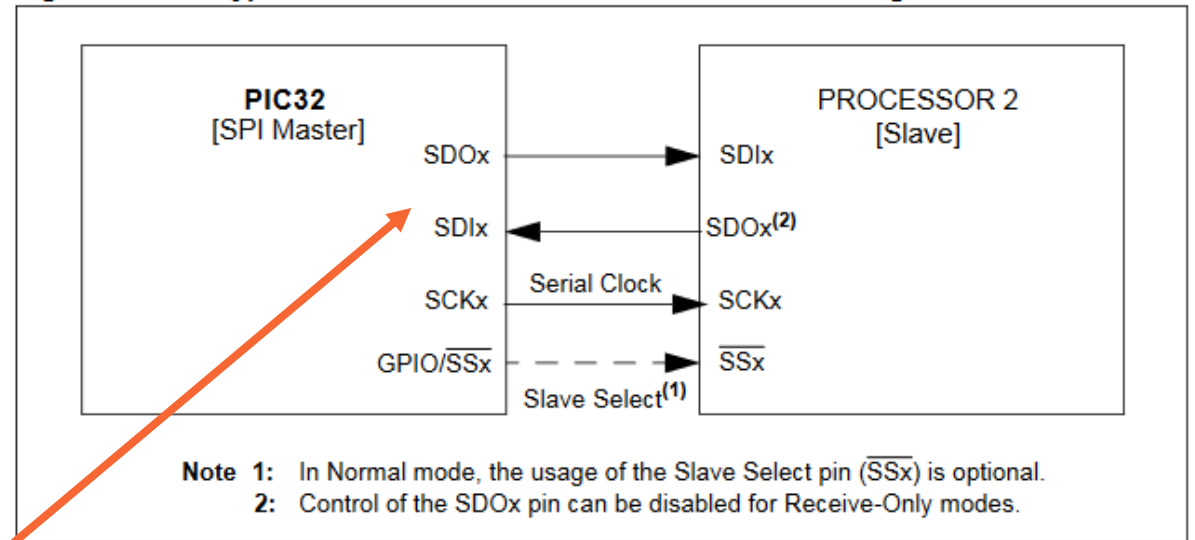
Topic	Slide
Serial Peripheral Interface	3
SPI Module Diagram	4
Using SPI	5
SN74HC595 (shift register)	6
Lab Goals	7
Notes	8

Serial Peripheral Interface (SPI)

- SPI is a communication protocol often used for short-distance communication in embedded systems.
 - Serial communication sends data one bit at a time over a single channel.
- Master-slave architecture
 - Each can send data to the other.
 - Master provides the clock and slave select signals (slave select tells that particular device to listen).
 - PIC32 can operate as master or slave. We'll use it in master mode.

The terms **MOSI** (Master Out, Slave In) and **MISO** (Master In, Slave Out) are often used to refer to the two data channels.

Figure 23-2: Typical SPI Master-to-Slave Device Connection Diagram



PIC32 FRM – Section 23. SPI, pg. 4

The SPIx serial interface consists of four pins:

- SDIx: Serial Data Input
- SDOx: Serial Data Output
- SCKx: Shift Clock Input or Output
- \overline{SSx} : Active-Low Slave Select or Frame Synchronization I/O Pulse

PIC32 FRM – Section 23. SPI, pg. 2

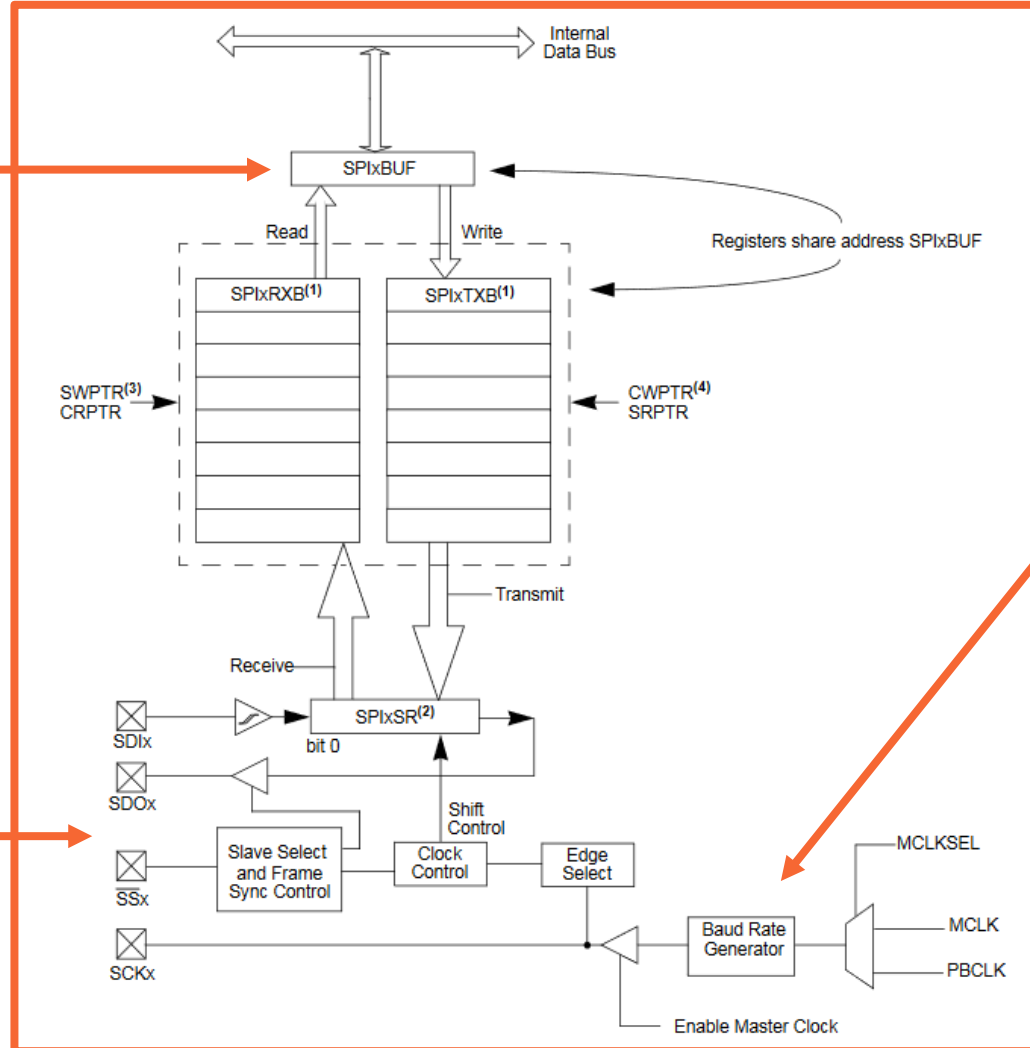
SPI Module Diagram

Write transmit data and read received data from SPIxBUF

- Data propagates automatically to SPIxTXB (transmit) or from SPIxRXB (receive)
- **You will only write to SPIxBUF**

Output pins

- SDOx and SSx must be mapped with PPS
- SCKx is hard mapped to a specific pin



Baud Rate Generator

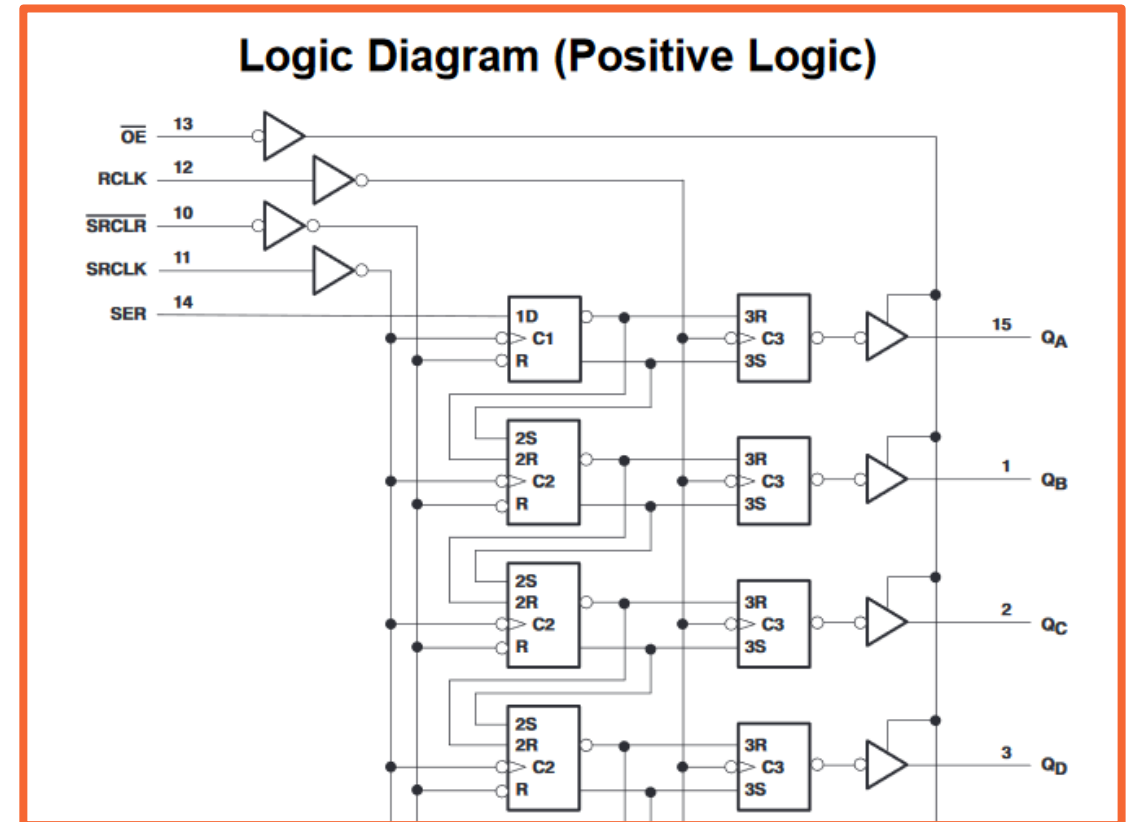
- Baud rate refers to rate of data transfer (bits/s)
- Here, the BRG divides the clock to achieve desired rate

Using SPI

- The document **Section 23 – Serial Peripheral Interface** is available on Canvas in the Lab 9 module.
 - Contains all the information about the PIC32's SPI capabilities
- Follow the steps in **Section 23.3.3.1 Master Mode Operation**.
 - This covers the majority of what you need to do in your code.
 - Study pages 18-20 for a better understanding.
 - See the notes at the end of these slides for details on what values you will set.
- Registers of interest
 - SPIxBUF //write or read data
 - SPIxCON //configure SPI module (pg. 8)
 - SPIxSTAT //SPI module status (pg. 13)
 - SPIBRG //divisor for baud rate generator

SN74HC595

- Datasheet available [here](#)
- Study the logic diagram on page 1
 - Features two registers (the columns of 8 flip-flops)
 - Shift register (left column) receives data from **SER** one bit at a time. On each tick of **SCLK**, the newest bit is stored in the top FF, and previous bits are each shifted down one.
 - Storage register (right column) is connected to outputs Q_A - Q_H . It updates to match the shift register on each tick of **RCLK**.
- Find the timing diagram on page 8
 - Notice that the outputs update on rising edges of RCLK.
 - Outputs go to high impedance state when OE goes high, and get cleared when SRCLR goes low. Neither should be left floating.

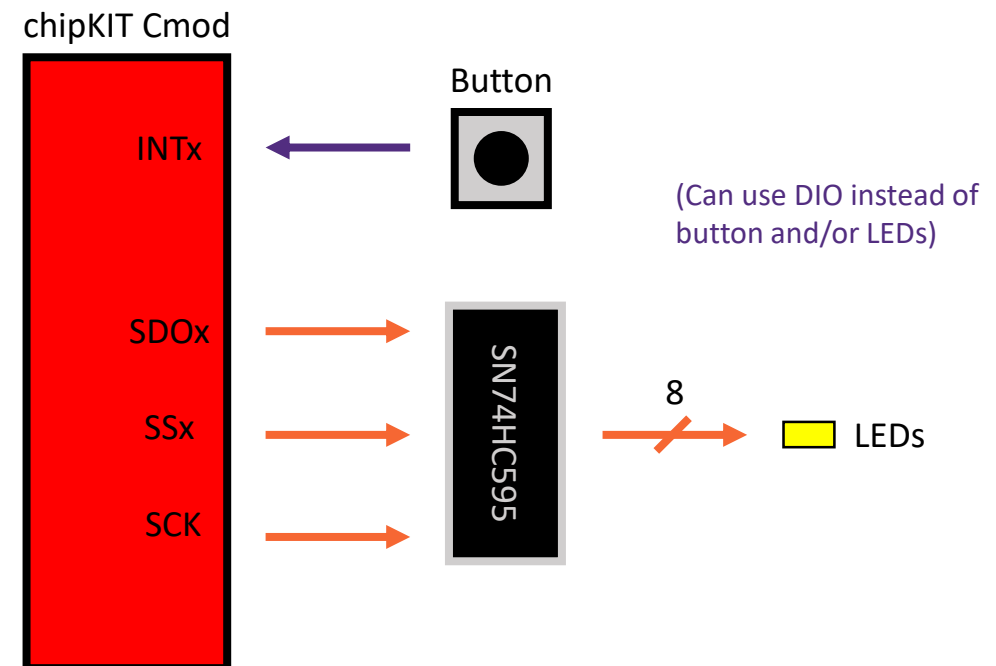


SNx4HC595 datasheet, pg. 1

Lab Goals

- Use the PIC32 in SPI master mode to send the values from the given array to the shift register.
- Connect the SN74HC595 inputs as shown in the diagram, and its outputs to LEDs (or digital reader).
- Set up an external interrupt. Each time it's triggered, the next value in the array should be written to SPIxBUF and appear on the LEDs.

Simple Diagram



Array of values to display:

```
char spiChars[18] = {0, 1, 4, 8, 16, 32, 64, 128, 255, 254, 253, 251, 247, 239, 223, 191, 127};
```

Notes

- The SPI module can transmit 8-, 16-, or 32-bit data. What data width will be needed for this lab, and how is that mode selected? (see section 23.3.1)
- The following is a checklist of values to set (located in SPI1CON, unless otherwise specified)
 - IEC1bits.SPIRX = 0
 - IEC1bits.SPITX = 0 // Disable SPI interrupts
 - ON = 0 // Disable SPI module during setup
 - SPI1BUF
 - ENHBUF = 0 // Don't want enhanced buffer mode
 - CKP, CKE // See figure 23-9 in Section 23 document (pg. 20)
 - SPIBRG = 1000 // Should result in baud rate slow enough to observe transmission with oscilloscope
 - SPIROV // In SPI1STAT
 - MSTEN, MSSEN
 - ON = 1 // Enable SPI module after setup is complete

Lab 10: Analog to Digital Conversion

ECE 3720

Preview

A variable, analog voltage will be produced with a force-sensitive resistor and serve as an input to the PIC32's ADC module. The ADC will convert the voltage into a digital value, which will be displayed in binary on the MC's output LEDs. Increasing the force on the FSR should cause the LEDs to count up.

Topic	Slide
Analog to Digital Conversion (ADC)	3
ADC Diagram	4
Using ADC	5
Force-Sensitive Resistor (FSR)	6
Lab Goals	7
Notes	8

Analog to Digital Conversion (ADC)

- ADC is the process of reading an analog signal, which has infinite possible values, and producing a digital representation of it using a finite number of bits.
- This is done any time an analog value (typically a voltage) must serve as an input to a computer.

Example: Suppose we had an analog input of 0-4 V and a 2-bit ADC:

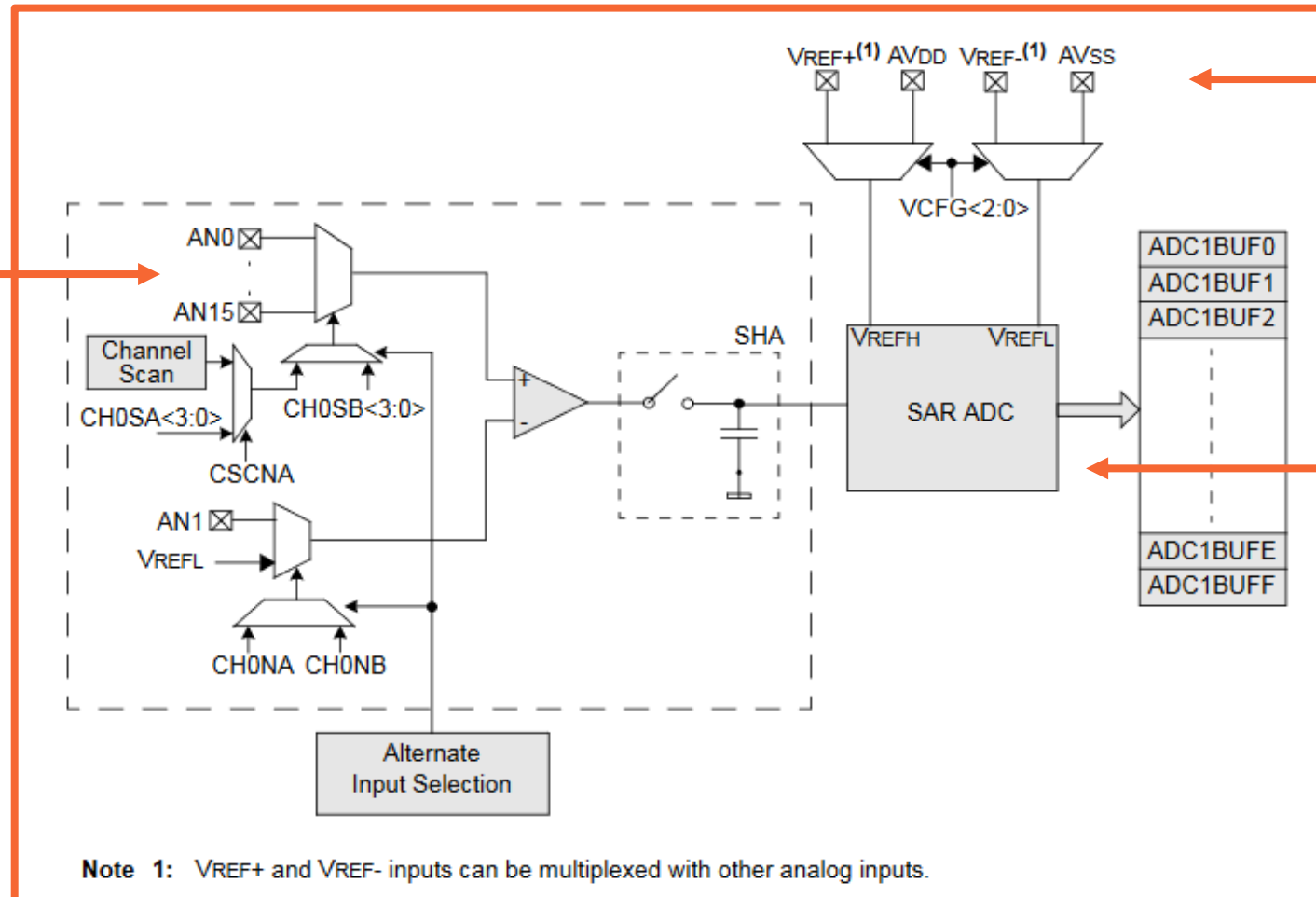
Analog input value	Digital representation
0-1 V	00
1-2 V	01
2-3 V	10
3-4 V	11

- Note that the precision of the digital representation depends on the number of bits available. 2 bits gives us 4 possible values, 3 bits would allow 8, etc.
- **We will be using a 10-bit ADC and a voltage range of 0 to 3.3V**

ADC Module Diagram

Multiplexers to select ADC inputs

- Positive input can be any of the analog read pins.
- Negative input is AN1 or low reference voltage.
- Settings for MUXA and MUXB (we only use A)



Reference Voltages

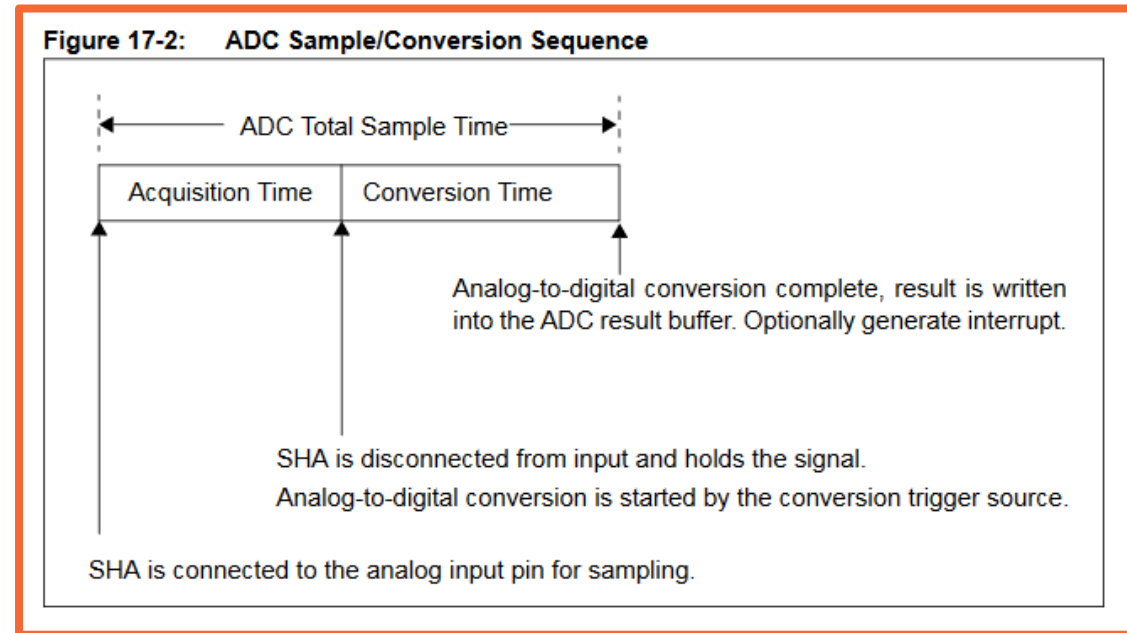
- $V_{REF+/-}$ are external pins.
- AV_{DD} and AV_{SS} are the PIC32's internal voltages (3.3V and 0V).

Sampling and Conversion

- Sample and hold amplifier (SHA) reads input and holds.
- Successive approx. register (SAR) ADC converts to digital.
- Results stored in ADC buffers.

Using ADC

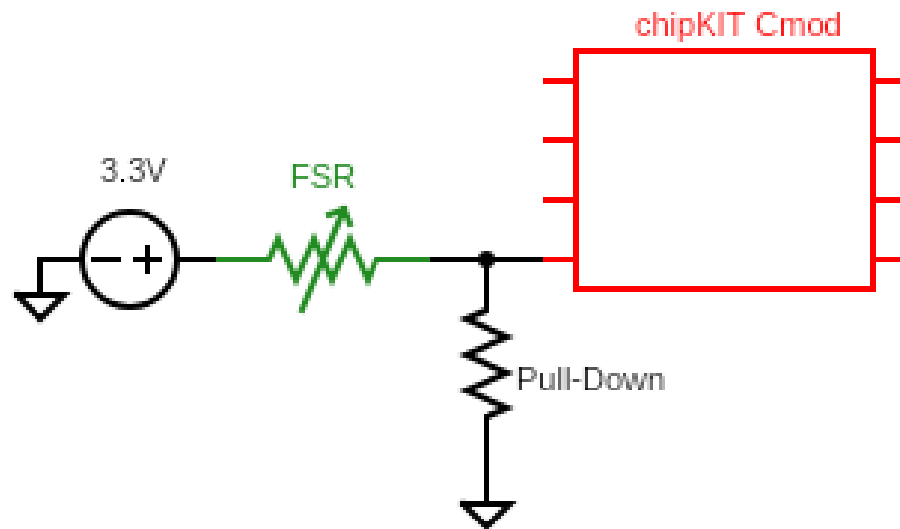
- The document **Section 17 – 10-Bit AD Converter** is available on Canvas in the Lab 10 module.
 - Contains all the information about the PIC32's ADC capabilities
 - See section 17.3.1 (pg. 12) for a more detailed explanation of ADC module's operation.
- Follow the steps in section **17.4 ADC Module Configuration**.
 - Sections 17.4.1-17.4.15 provide detail on each of the steps.
 - See the notes at the end of these slides for details on how to set up ADC for this particular lab.
 - **You will set up an ADC interrupt to trigger after every 4 samples and display their average.**



PIC32 FRM – Section 17. ADC, pg. 12

Force-Sensitive Resistor (FSR)

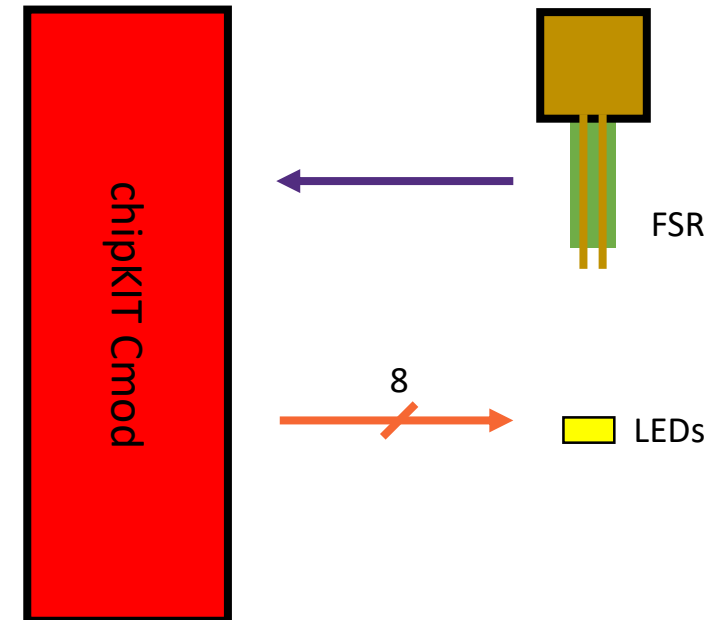
- Very high resistance normally.
- Resistance decreases as force is applied.
- Used to create a variable, analog voltage.



Lab Goals

- Wire the FSR to supply a variable, analog voltage (0-3.3V) to an analog-read-capable pin.
- Set up the PIC32's ADC module to sample and convert the voltages (section 17.4).
- Set up the ADC interrupt:
 - Should trigger after every 4 samples.
 - Average the 4 converted values (first 4 ADCBUFs).
 - Output the 8 most significant bits (10-bit ADC, so $\gg 2$).
- Display the converted value on 8 LEDs or with digital reader.


Simple Diagram





Notes

- Make sure to follow all 15 steps of section 17.4 (step 13 should appear last in the code).
- We don't have AD1PCFG; use ANSELx instead.
- Only use MUXA.
 - The ADC module can be set up to alternate between two inputs, which use MUXA and MUXB to select their ADC inputs, respectively. Since we are only using one input, we only need MUXA.
- Make sure sampling and conversion trigger automatically.
- Take 4 samples before each interrupt.
 - It's common to average multiple samples with an ADC, in order to ensure the result accurately reflects the input.
- Use 12 for your acquisition time (step 11)
- Use 6 for your prescaler (step 12)



Conversion time is the time required for the ADC to convert the voltage held by the SHA. The ADC requires one ADC clock cycle (TAD) to convert each bit of the result, plus two additional clock cycles. Therefore, a total of 12 TAD cycles are required to perform the complete conversion. When the conversion time is complete, the result is written into one of the 16 ADC result registers (ADC1BUF0 through ADC1BUFF).