Designs using a PIC Microcontroller

ECE 401 Senior Design I

Week #5

Please visit Bison Academy for corresponding lecture notes,
homework sets, and videos
www.BisonAcademy.com

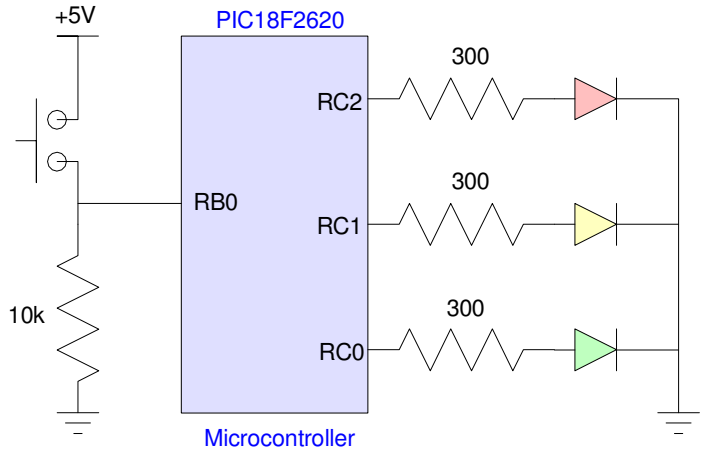
Designs using a PIC Microcontroller

Many of the projects in Senior Design I can be done in

- Hardware, or
- Software

Microcontrollers are just a tool:

- If the tool helps, use it.
- If the tool doesn't help, don't use it.



Reasons to Not Use a Microcontroller:

- No wiring up a microcontroller,
- No code needs to be written and downloaded
- Don't have to worry about program crashes
- Sometimes, it's a simpler design

Reasons To Use a Microcontroller

- Usually simplifies the hardware design
- Really frees up what you can do
- Makes revions as simple as downloading a new program

This Lecture:

Topics:

- Hardware: How to wire up a PIC chip so that you can make a light blink
- Downloading: How to get your code onto the PIC chip, and
- Coding: How to write simple C routines to make a light blink

i.e. How to make a light blink.

Only engineers get excited when a light blinks.

- It's a big deal.
- You were able to compile your code
- You were able to download your code, and
- Your code is running.

Once you get a light to blink, the rest is easy (sort of)...

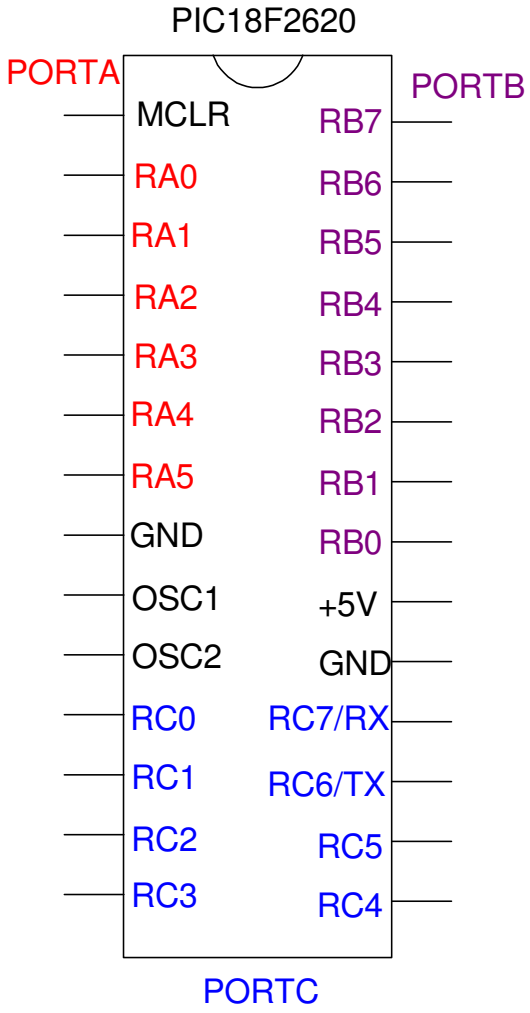
Hardware:

Only PIC18F2620 allowed in Senior Design I

- We have a boot-loader for this chip
  - *same as ECE 376*
- Students have experience using this chip
  - *same as ECE 376*
- The C compiler is free
  - *I like free*
- Coding is identical to that used ECE 376

28 I/O Pins

- Arranged into three ports
  - *PORTA*
  - *PORTB*
  - *PORTC*



Schematic for Minimum Connections:

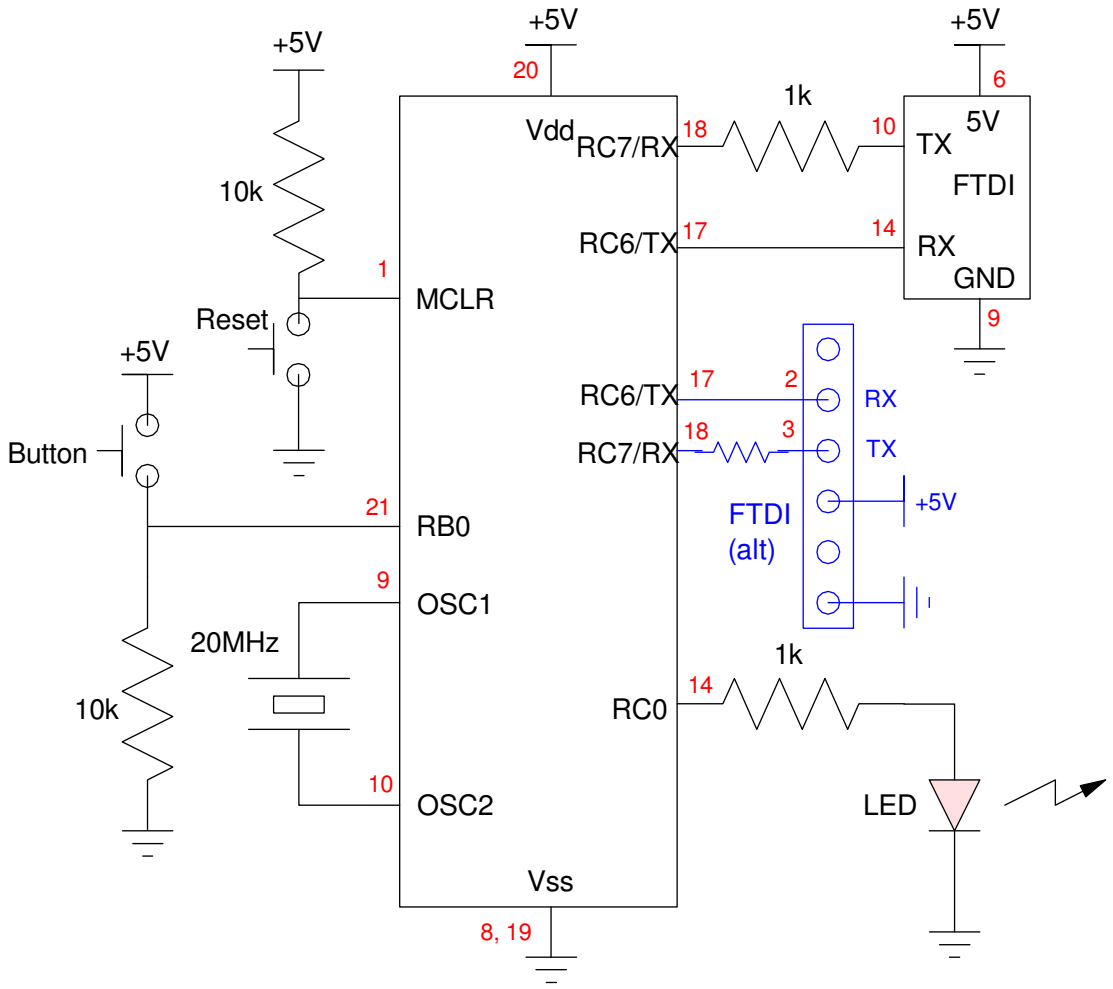
- Power & Ground
- 20MHz crystal
- Reset button
- FTDI
  - *download programs*

RB0 = Input

- 0V: button not pressed
- 5V: button pressed

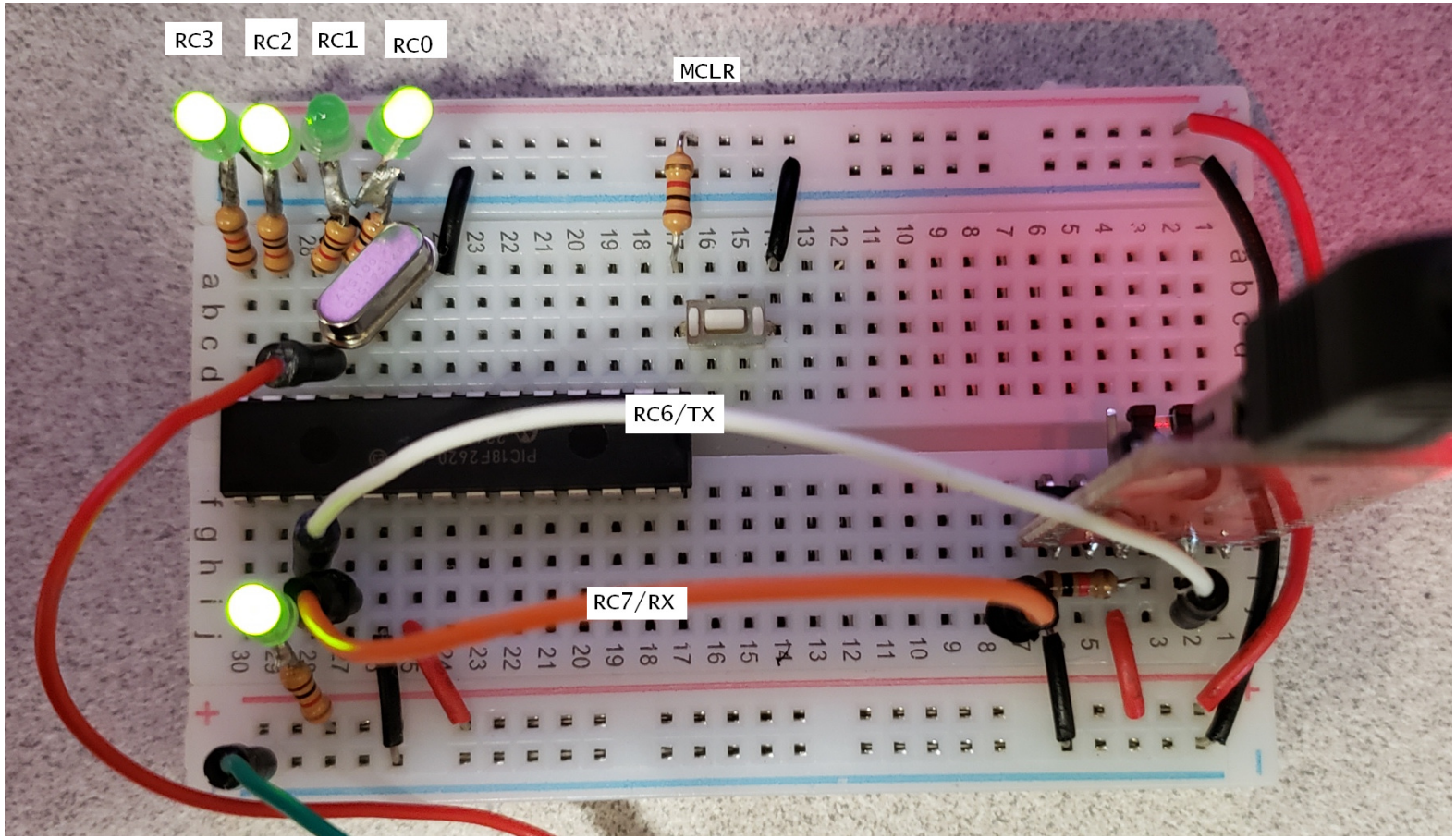
RC0 = Output

- 0: LED is off
- 1: LED is on



Minimum Connections on a Breadboard

- LEDs added to PORTC



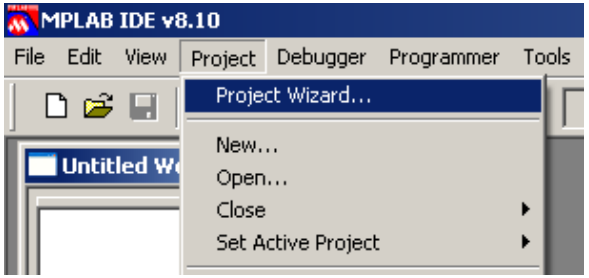
C Coding with MPLAB8

much easier to use than MPLABX

Step 1: Start with a working program.

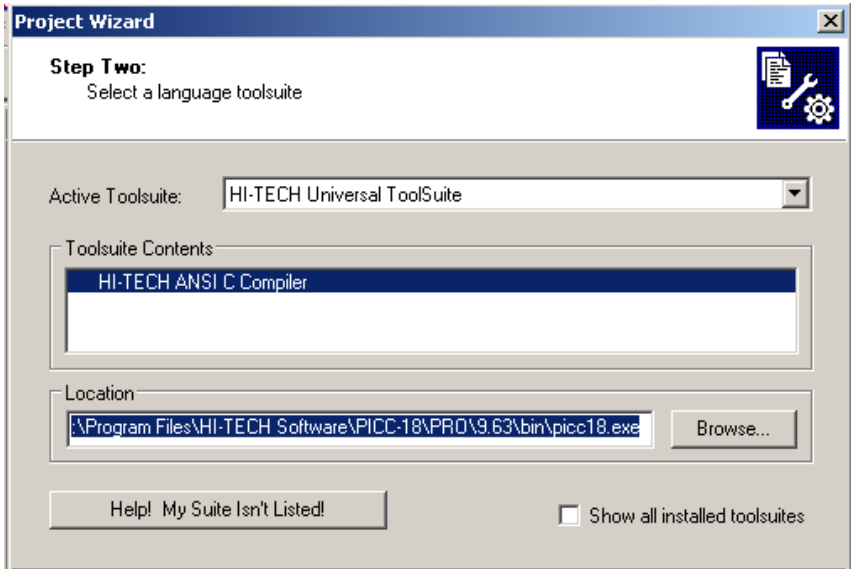
- Download sample code from Bison Academy
- Place in a directory where you can find it

  z:\ECE401\Clock

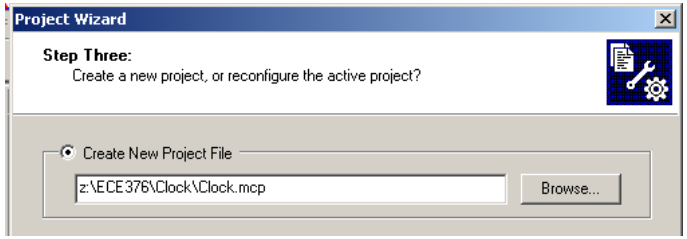


Step 2: Start MPLAB.

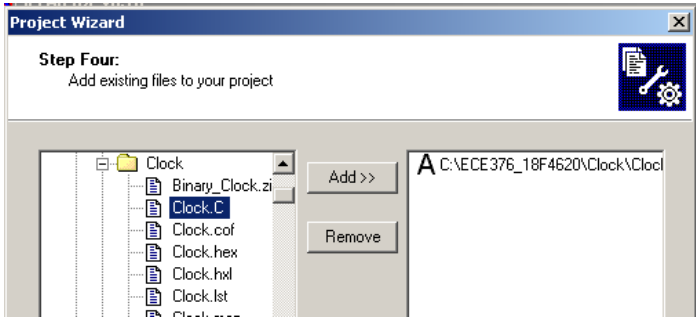
- Go to the program wizard
- Select your device:
  - *PIC18F2620 (or 4620)*
- Select the Hi-Tech C Universal Toolsuite.



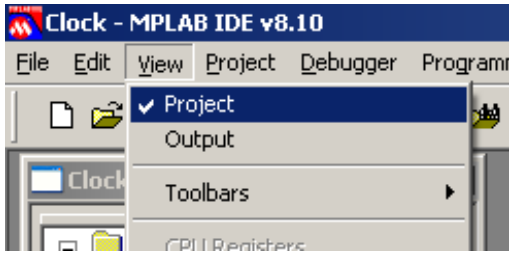
Change the path to where the files are located



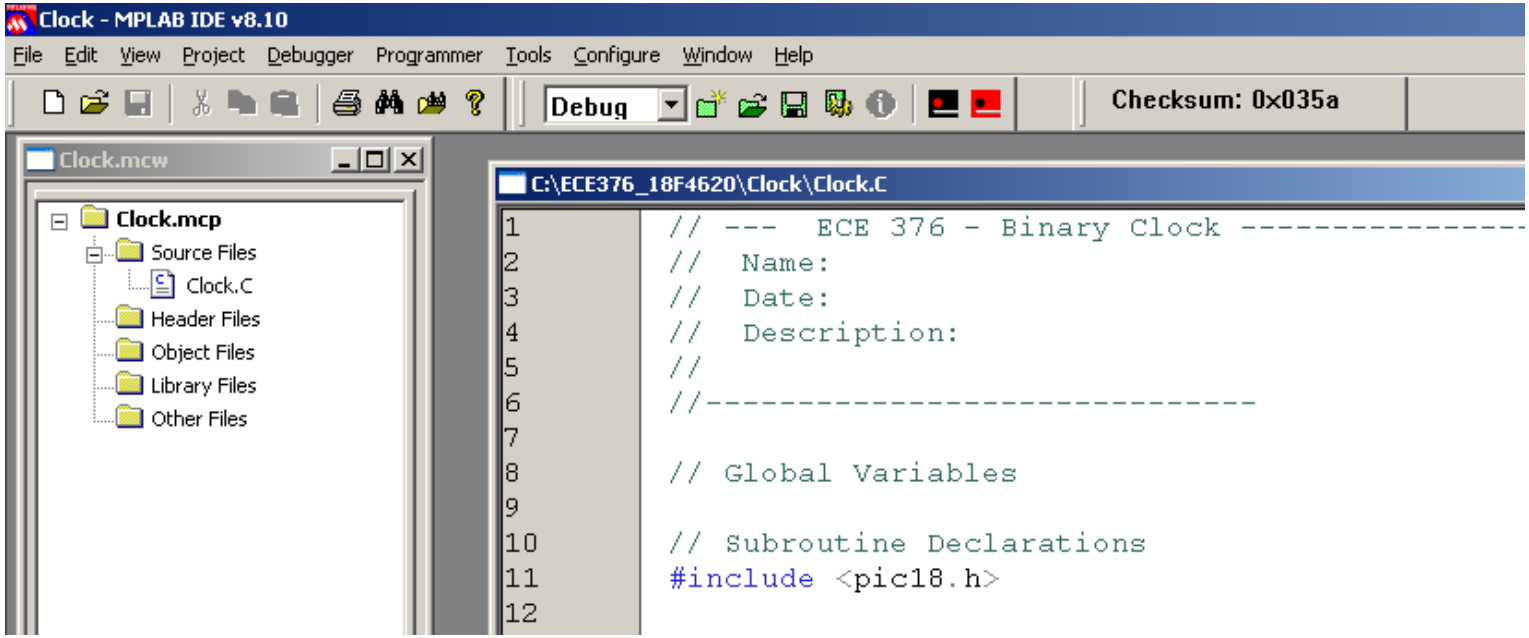
Select the C program you want to compile (usually the name of the directory)



Select View Project



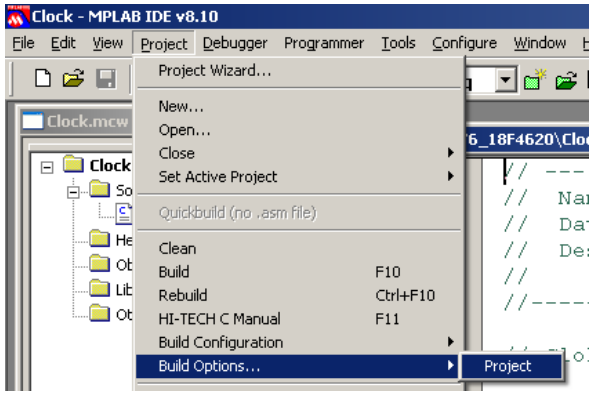
You should get the following screen:



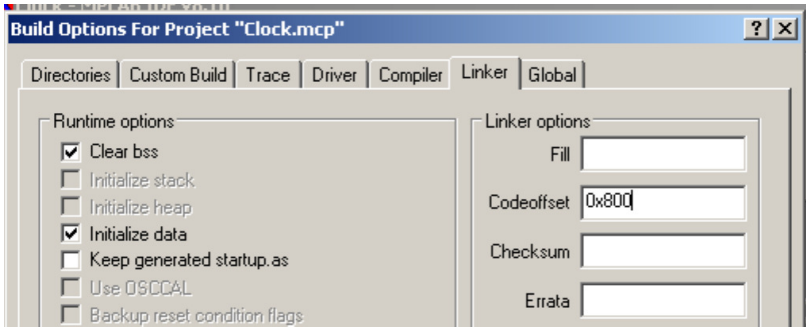
*** important *** Offset your code by 0x800

- *Your code needs to start at 0x800 - after the boot-loader.*

Go to Project - Build Options - Project



Under Linker, offset the code by 0x800



if your code compiled yesterday but fails today, you probably don't have the 0x800 offset.

Compile your code

Project Build All (or F10)

You should get the following message

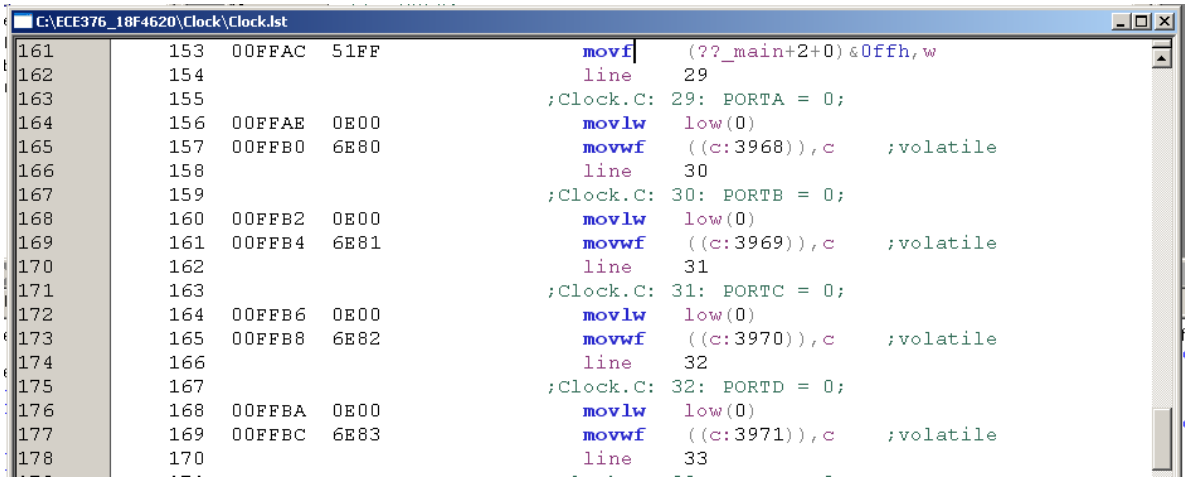
```
Memory Summary:
Program space        used    76h (   118) of 10000h bytes   (  0.2%)
Data space           used     3h (     3) of   F80h bytes   (  0.1%)
EEPROM space         used     0h (     0) of   400h bytes   (  0.0%)
ID Location space    used     0h (     0) of     8h nibbles (  0.0%)
Configuration bits   used     0h (     0) of     7h words   (  0.0%)
```

This tells you your code compiled and uses up 118 bytes (out of 64k), 3 bytes of RAM (out of 4k), etc.

This also creates some files

Clock.lst

This shows how your C code converts to assembler. A section looks like the following



Clock.hex

This is the machine code you download to your processor

```
:04000000C7EF7FF0D7
:10FF8E00000E926E000E936E000E946E000E956E25
:10FF9E00000E966E0001FF6F0F0EC16E0001FF5135
:10FFAE00000E806E000E816E000E826E000E836E4D
:10FFBE00000E846E000E00010001FD6F000E0001A8
:10FFCE00FE6F010E00010001FD2500010001FD6F15
:10FFDE00000E00010001FE210001FE6FFDC083FF37
:10FFEE00836601D001D002D08228826EEAD700EF5C
:02FFFE0000F011
:00000001FF
```

Note that the reason we like C so much is

- It compiles to assembler fairly directly
- Meaning it is efficient, and
- C has things like multiply, divide, loops, arrays.

C-Coding

For Senior Design I, the programs don't need to be that complicated

All you need for this course are

- Counters
- if-statements
- while-loops
- subroutines

Input & Output Pins

Each I/O pin on a PIC can be either input or output

- Input: Read the buttons or other devices.
  - *5V = logic 1*
  - *0V = logic 0*
- Output: Drive something like an LED
  - *Logic 1 = 5V*
  - *Logic 0 = 0V*

Note: The maximum current for output pins is 25mA

Initializing I/O Pins

TRISx register determines which pins are input & output

- TRISA controls PORTA
- TRISB controls PORTB
- TRISC controls PORTC

Each bit of TRISx sets the status or PORTx

- TRISA = 0x00 *all pins of PORTA are output (0 means output)*
- TRISB = 0xFF *all pins of PORTB are input (1 means input)*
- TRISC = 0x0F *bits 4..7 are output, bits 0..3 are input*

Writing to Output Pins

You can write to all eight bits at once

```
PORTA = 0x00;   all pins on PORTA are 0V
PORTB = 0xFF;   all pins on PORTB are 5V
PORTC = 0x01;   pin #0 is 5V, the rest are 0V
```

You can also address each bit separately

```
RA0 = 1;  Port A bit #0 is 5V, other pins are unchanged
RB3 = 0;  Port B bit #3 is 0V
RC7 = 1;  Port C bit #7 is 5V
```

Note: when initializing the I/O ports, you need to include the code

```
ADCON1 = 0x0F;
```

For more details on this, please refer to ECE 376 on analog inputs and outputs.

Sample Code: Write 1, 2, 3 to Port A, B, C

defines PORTx, TRISx

start of the program

set all pins to output

write 1, 2, 3 to PORTA, B, C

stop (infinite looop)

```
#include <pic18.h>

void main(void)
{

   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   ADCON1 = 0x0F;

   PORTA = 1;
   PORTB = 2;
   PORTC = 3;

   while(1);

}
```

Compilation Results:

```
Memory Summary:
    Program space        used    2Eh (    46) of 10000h bytes   (  0.1%)
    Data space           used     1h (     1) of   F80h bytes   (  0.0%)
    EEPROM space         used     0h (     0) of   400h bytes   (  0.0%)
    ID Location space    used     0h (     0) of     8h nibbles (  0.0%)
    Configuration bits   used     0h (     0) of     7h words   (  0.0%)
```

This C code compiles into 23 lines of assembler (46 bytes: each instruction is two bytes)

Note:

- The while(1); statement at the end is a *stop* command.
- If you remove it, the program ends
- When that happens, it restart at address 0x0000

Program #2: Make RC0 blink at 220Hz

define a 16-bit variable, i

All pins are output

start with PORTC cleared

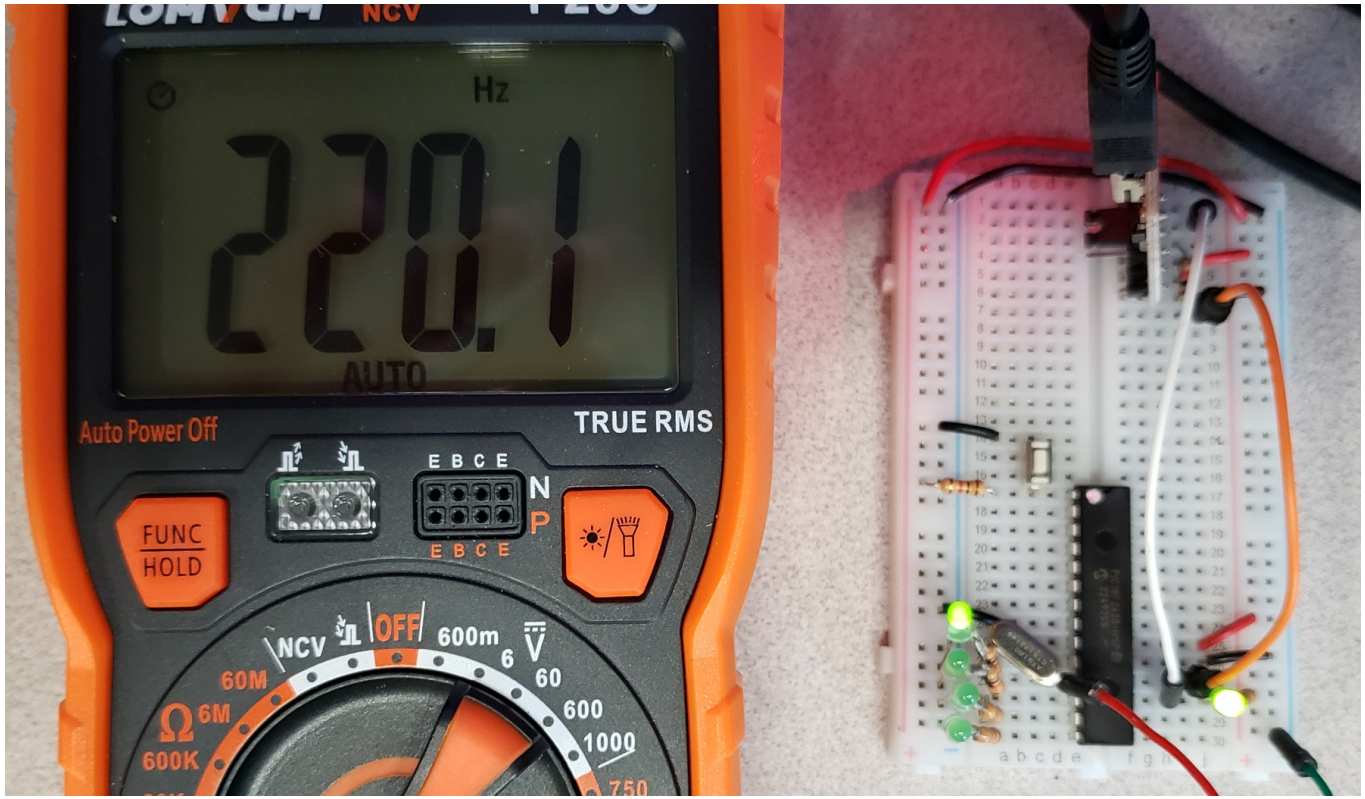
infinite loop
toggle PORTC pin 0
wait 1419 counts (220Hz)

```
#include <pic18.h>

void main(void)
{
   unsigned int i;

   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   ADCON1 = 0x0F;
   PORTC = 0;

   while(1) {
      RC0 = !RC0;
      for(i=0; i<1419; i++);
      }
}
```

note: 1419 is found using trial an error
larger numbers take longer to execute (lower frequency)
use trial and error to get the frequency to 220Hz

Checking the Frequency on RC0

- Use an osilloscope
- Use a frequency counter on a multimeter
- Adjust the number 1419 until you get 220Hz



Program #3: Subroutines and Wait loops

Subroutines can make progrms easier to write and use

Example: Write a subroutine which waits X ms

- Use a for-loop to kill time
- Adjust the count so that Wait(1000) waits 1000ms

```
void Wait(unsigned int X)
{
   unsigned int i, j;
   for (i=0; i<X; i++)
      for (j=0; j<617; j++);
   }
```

Write a program which

- Counts in binary
- One count per second

Note:

- It's now very easy to change the wait time

```
// Subroutine Declarations
#include <pic18.h>

// Subroutines
void Wait(unsigned int X)
{
   unsigned int i, j;
   for (i=0; i<X; i++)
      for (j=0; j<617; j++);
   }

// Main Routine

void main(void)
{
   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   ADCON1 = 0x0F;
   PORTC = 0;

   while(1) {
      PORTC += 1;
      Wait(1000);
      }
  }
```

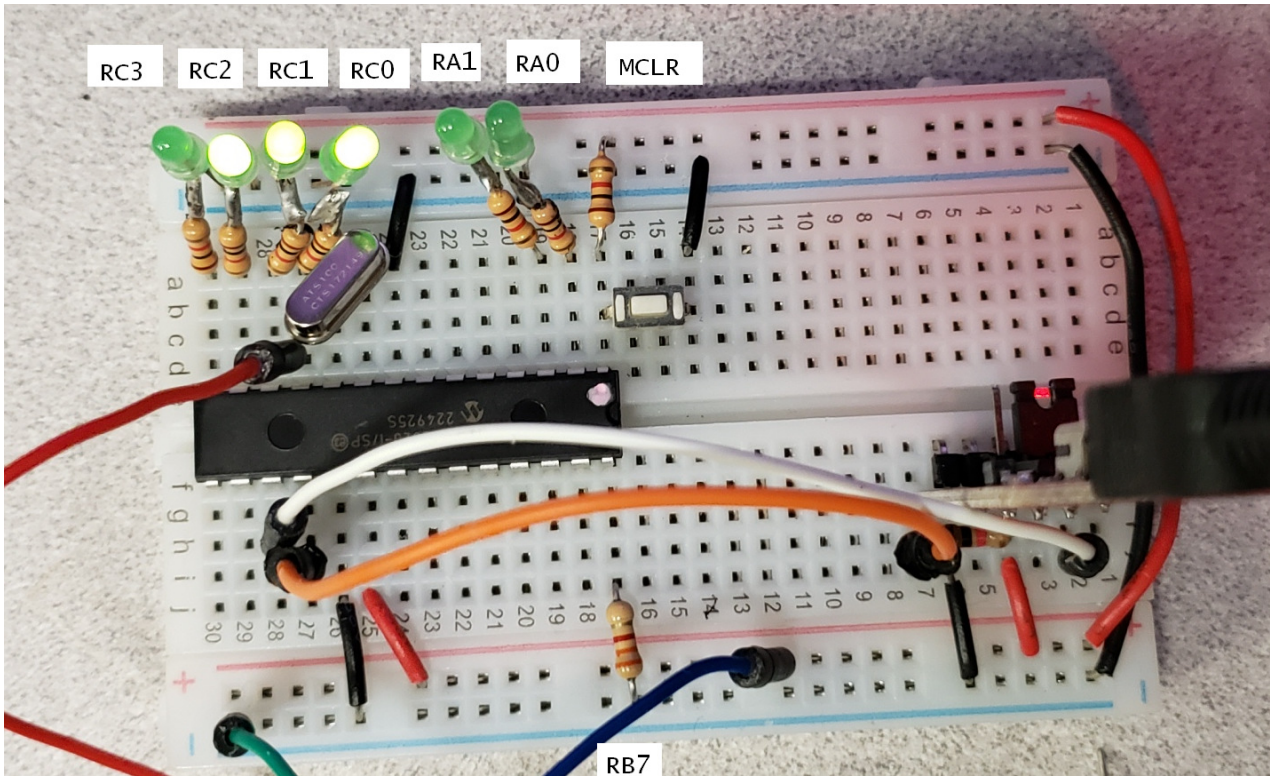
Program #4: Counter

- Beep every time button RB0 is pressed and released
- After 10 button presses, turn on the light on RC0 for one second

Again, use subroutines

```
void Wait(unsigned int X)
{
   unsigned int i, j;
   for (i=0; i<X; i++)
      for (j=0; j<617; j++);
   }

void Beep(void)
{
   unsigned int i, j;
   for (i=0; i<50; i++) {
      RA1 = !RA1;
      for (j=0; j<200; j++);
      }
   }
```

```
// Main Routine

void main(void)
{
   unsigned int COUNT;

   TRISA = 0;                          PORTA = output
   TRISB = 0xFF;                       PORTB = input
   TRISC = 0;                          PORTC = output
   ADCON1 = 0x0F;

   COUNT = 0;

   while(1) {                          infinite loop
      while(RB7);                      wait until you detect a rising edge on
      while(!RB7);                     PortB bit #7

      Beep();                          when found, beep

      COUNT += 1;                      increment a count
      PORTC = COUNT;                   send the cout to PORTC

      if (COUNT >= 10) {               after 10 counts
         RA0 = 1;                         turn on RA0
         Wait(1000);                      for 1000ms
         RA0 = 0;                         turn RA0 off
         COUNT = 0;                       then clear the counter
         PORTC = COUNT;
         }
      }
   }
```



Counting rising edges on RB7

C Language Summary

Character Definitions:

Name	bits	range
char	8	-128 to +127
unsigned char	8	0 to 255
int	16	-32,768 to +32,767
unsigned int	16	0 to 65,535
long	32	-2,147,583,648 to +2,147,483,647
unsigned long	32	0 to 4,294,967,295
float	32	3.4e-38 to 3.4e38
double	64	1.7e-308 to 1.7e+308
long double	80	3.4e-4932 to 3.4e+4932

Arithmetic Operations

Name	Example	Operation
+	1 + 2 = 3	addition
-	3 - 2 = 1	subtraction
*	2 * 3 = 6	multiplication
/	6 / 3 = 2	division
%	5 % 2 = 1	modulus
++	A++	use then increment
	++A	increment then use
--	A--	use then decrement
	--A	decrement then use
&	14 & 7 = 6	logical AND
|	14 | 7 = 15	logical OR
^	14 ^ 7 = 9	logical XOR
>>	14 >> 2 = 3	shift right. Shift in zeros from left.
<<	14 << 2 = 56	shift left. Shift zeros in from right.

Defining Variables:

```
int A;                  A is an integer
int A = 3;              A in an integer initialized to 3.
int A, B, C;            A, B, and C are integers
int A=B=C=1;            A, B, and C are integers, each initialized to 1.
int A[5] = {1,2,3,4,5};         A is an array initialized to 1..5.  Note:  A[0]=1.
```

Arrays:

```
int R[52];              Save space for 52 integers
int T[2][52];           Save space for two arrays of 52 integers.
```

note: The PIC18F2626 only has 3692 bytes of RAM, so don't get carried away with arrays.

General C Commands:

Conditional Expressions:

```
!          not.   !PORTB means the compliment of PORTB.
=          assignment
==         test if equal.
>          greater than
<          less than
>=         greater than or equal
!=         not equal
```

IF Statement

```
if (condition expression)
{   statement or group of statements
   }
```

example: if PortB pin 0 is 1, then increment port C:

```
if (RB0==1) {
   PORTC += 1;
   }
```

IF - ELSE Statements

```
if (condition expression)
{   statement or group of statements
   }
else {
   alternate statement or group of statements
   }
```

Example: if PortB bit 0 is 1, then increment port C, else decrement port C:

```
if (RB0==1)
   PORTC += 1;
   }
else
   PORTC -= 1;
   }
```

SWITCH (CASE)

```
switch(value)
{
   case value:  statement or group of statements
   case value:  statement or group of statements
   defacult:    statement or group of statements
   }
```

WHILE LOOP

```
while (condition is true) {
   statement or group of statements
   }
```

DO LOOP

```
do {
   statement or group of statements
   } while (condition is true);
```

FOR-NEXT

```
for (starting value; do while true; changes) {
   statement or group of statements
   }
```

Infinite Loop

```
while(1) {
    statement or group of statements
    }
```

note: Zero is false. Anything other than zeros is true. while(130) also works for an infinite loop.

Subroutines in C:

To define a subroutine, you need to

- Declare how this subroutine is called (typically in a .h file)
- Declare what the subroutine is.

The format is

returned_variable_type = subroutine_name(passed_variable_types).

Example: Write a subroutine which returns the square of a number:

```
// Subroutine Declarations

int Square(int Data);

// Subroutines

int Square(int Data) {
   int Result;
   Result = Data * Data;
   return(Result);
   }
```

Standard C Code Structure

So that others can modify your code more easily, a standard structure is to be used. This places all code in the following order:

```
//-----------------------------------
//  Program Name
//
//  Author
//  Date
//  Description
//  Revision History
//-----------------------------------

// Global Variables

// Subroutine Declarations
#include <pic.h>          // where PORTB etc. is defined
```

```
// Subroutines

// Main Routine

void main(void)
{

   TRISA = 0;
   TRISB = 0xFF;
   TRISC = 0;
   ADCON1 = 15;
   PORTA = 1;
   PORTC = 3;

   while(1) {
      PORTC = PORTB;
      };
   }
```

I/O Pin Names

- C is case sensitive
- C is spelling sensitive

The names of the I/O registers (8 bits) and individual bits are as follows

- PORTA, B, C are connected to I/O pins on the PIC18F2620
- PORTD & E are not

Address	Register Name	Bit							
		7	6	5	4	3	2	1	0
0xF80	PORTA	-	-	RA5	RA4	RA3	RA2	RA1	RA0
0xF81	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0xF82	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
0xF83	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
0xF84	PORTE	-	-	-	-	RE3	RE2	RE1	RE0
0xF92	TRISA	-	-	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
0xF93	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
0xF94	TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
0xF95	TRISD	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
0xF96	TRISE	-	-	-	-	TRISE3	TRISE2	TRISE1	TRISE0