Getting Started on the PIC

Today:

- Basics of C
- Basics of CCS (the C compiler for the PIC
- Interacting with the PIC
- A basic circuit

Administrivia

- Starting after class, the full lab schedule will be in place
  - This goes for the next two weeks (although who covers the time will vary)
- In the following week (no later than Monday), you must be checked out on the PIC procedure by myself, Patrick, or John
- You may demonstrate widgets as soon as you are ready

Data Types (C)

- signed/unsigned integers
- **char**: characters
- **float**: 32-bit floating point number
- **void**: no type

Integer Data Types (somewhat specific to CCS)

- **int1**: a 1-bit number (the compiler will pack multiples into a single byte)
  - Also known as **short**
- **int8**: an 8-bit number (signed)
  - Also known as **int**
- **int16**
  - **long**
- **int32**

Aggregations (C)

- Arrays:

```
int foo[10];
int bar[4][3];
```

- Struct: binding primitive types into a group

```
Struct data {
   int i;
   float value;
}
```

- Unions: declared like structs, but the individual components overlap in memory

Enumerated Types

```
enum boolean {true, false};
boolean b, flag;

b = true;
flag = false;
if(flag == true) {...}
```

New Types

- **Typedef**: declaration of new types (think of it as the data component of a class):

```
typedef struct {
    int i;
    float value;
} data_element;

data_element a;

a.i = 10;
a.value = 2.78;
```

Pointers

- Similar to java “references” except no run-time checking is performed
- Very limited support in CCS

```
data_element a;
data_element *b;

a.i = 10;
b = &a;

b->value = 10.33;  // Sets a.value
```

Pre-Processor Directives

- In C, these are always preferenced with a '#' character
- All pre-processor directives are evaluated before the compilation stage
- For example, other files (in particular, header files) may be included using:

```
/* Include the header file for the 628*/
#include <16f628.h>
```

Pre-Processor Conditionals

```
#define DEBUG
#define CONST 5
        :
        :
#ifdef DEBUG
  printf("%d\n\r", CONST);
#endif
```

Inserting Assembly Code (CCS)

```
int count;


#ASM
   movlw 0x8
   movwf count
   movlw 0
#ENDASM
```

Augmentation of Function Declarations (CCS)

- #inline: place the code for the called function at the point of call (and do not call it as a subroutine)

```
#inline
int foo(int data) {
    :
}
```

Declaring Interrupt Service Routines (CCS)

- Function foo is called in response to an RTCC interrupt:

```
#int_RTCC
int foo(int data) {
    :
}
```

Addressing Specific Memory Locations (CCS)

- Addressing bytes:

```
/* Declare that variable b_port is
  located at byte 6 in the memory*/

#byte b_port = 6

b_port = 0x55;
```

- Addressing bits:

```
#bit b_port_3 = 6.3

b_port_3 = 0x1;
```

Configuring the PIC Chip (CCS)

- Each PIC has a set of configuration bits that determine the type of oscillator that is connected, whether the watch-dog timer is used, etc.
- These bits are set using the #FUSES directive:

```
#FUSES HS,NOWDT,NOPROTECT,PUT,BROWNOUT
```

  - high-speed cyrstal oscillator (HS), no watch-dog timer, no copy protection, power-up timer, and brown-out detector
  - For the instant – don't change these (under linux, we cannot reset these)

Declaring the Oscillator Speed

- Used to tell the compiler what the oscillator speed is:

```
#use delay (CLOCK=20000000)
```

RS232 Configuration (CCS)

- Configure one pin as transmit and the other receive:

```
#use RS232(BAUD=9600, XMIT= PIN_A2, RCV=PIN_A3)
```

  - Other parameters are possible
  - Directive is in force in the code until another #use RS232

RS232 Configuration (CCS)

- Almost any I/O pin may be used for serial I/O, but serial I/O hardware exists on some pins
- The I/O hardware will perform buffering and even raise interrupts when a byte arrives
- When other pins are used, the compiler substitutes the appropriate code
  - This allows for multiple serial ports on a single chip
  - Caveat: in order to receive a byte, the code must be listening

Digital I/O Configuration

- Any access to port A will result in code that first sets the tristate configuration before reading or writing from the port (this is the default):

```
#use standard_io(A)
```

- The port will be read/written without reconfiguration:

```
#use fast_io(A)
```

- Again: directive in force until the next one

Other Strangenesses (CCS)

- No case sensitivity unless you request it:

  ```
  #CASE
  ```

An Example Program

```
#include <16f628.h>
#use delay (CLOCK=20000000)
#FUSES HS,NOWDT,NOPROTECT,PUT,BROWNOUT
#use fast_io(B)
#use RS232(BAUD=9600,XMIT= PIN_A2,RCV=PIN_A3)
void flash_led(long len) {
        output_high(PIN_B0);
        delay_ms(len);
        output_low(PIN_B0);
        delay_ms(len);
}
```

An Example Program (cont)

```
main(){
   set_tris_b(0xF0);/* Set port B I/O config*/
   flash_led(250);
   flash_led(250);
   flash_led(250);
   while(1){
       printf("foo\n\r");
       flash_led(1000);
     }
}
```

Using the Compiler

```
Pooh> ccsc -I/courses/cs500/cs503/cs503/ccs/devices/
-I/courses/cs500/cs503/cs503/ccs/drivers/ led.c

CCS PCM C Compiler
Registered to: University of Massachusetts, Site License
Compiling: led.c
in /courses/cs500/cs503/cs503/examples/led/

     Linking
     Writing led.HEX
     Writing led.SYM
     Writing led.LST
     Writing led.COD
     Writing led.PJT
Memory usage:
      ROM=6%
      RAM=3% - 5%
No Errors

pooh>
```

The Resulting Assembly Code

```
..................... void flash_led(long len)
..................... {
.....................          output_high(PIN_B0);
*
0021:  BSF    06.0
.....................          delay_ms(len);
0022:  MOVF   26,W
0023:  MOVWF  28
0024:  CALL   00E
.....................          output_low(PIN_B0);
0025:  BCF    06.0
.....................          delay_ms(len);
0026:  MOVF   26,W
0027:  MOVWF  28
0028:  CALL   00E
..................... }
0029:  RETLW  00
.....................
```

Downloading to the Chip

```
Pooh> picp /dev/ttyS0 16F628 -wp led_pwm3g.HEX

Unrecognized record format: ';'

Pooh>
```

- Don't worry about the error...

Clearing the Program Memory

Blanking the Chip (erasing program memory):

```
Pooh> picp /dev/ttyS0 16F628 -ep
program memory: blank

Pooh>
```

Checking that a chip is blank:

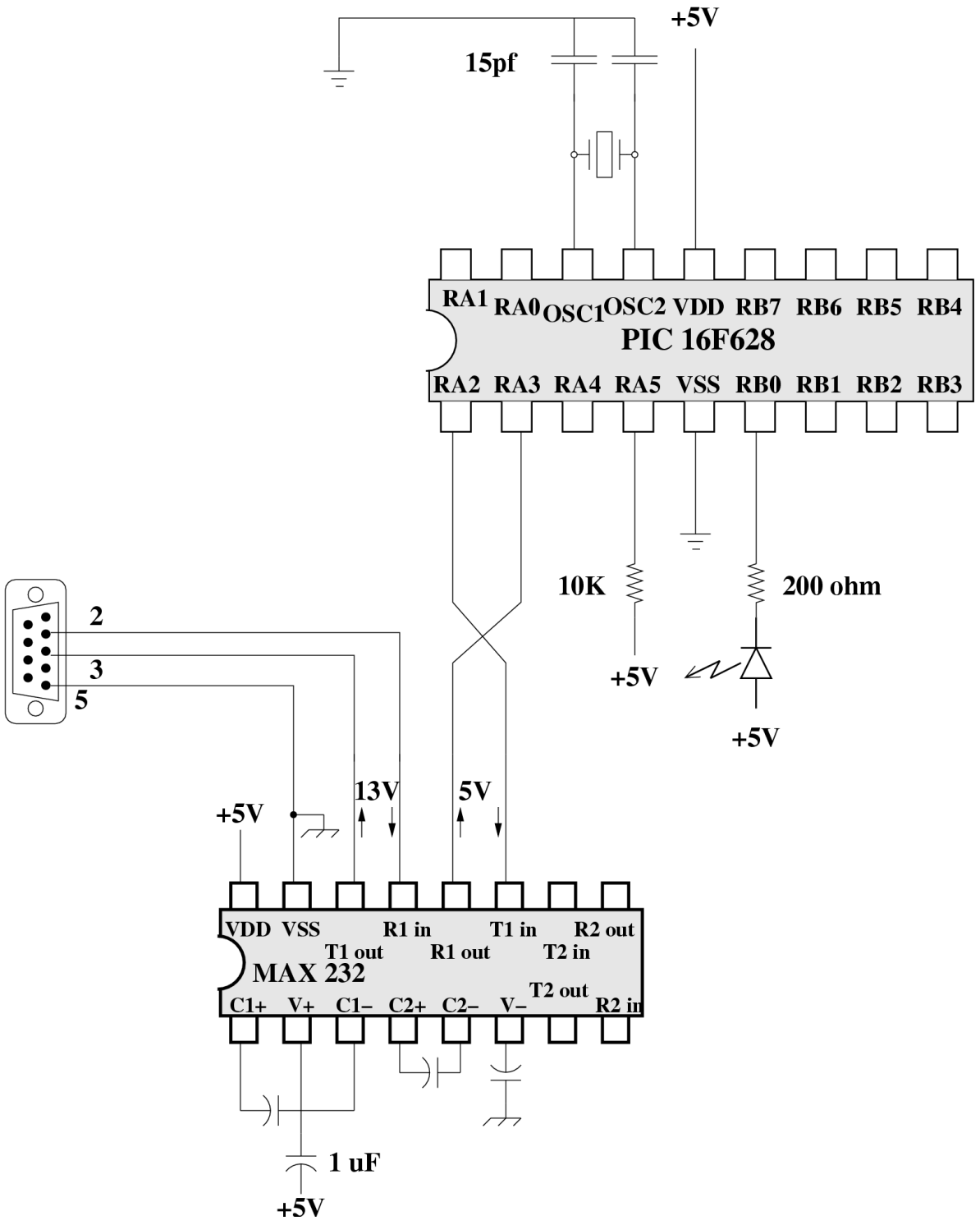
```
Pooh> picp /dev/ttyS0 16F628 -bp
program memory: not blank

Pooh>
```

Note: picp allows you to blank other components of the memory (but it does not work right now)

A Basic Circuit

Note: LED is on when
B0 is off.



Warnings
(i.e., do not blow up the chips)

- Never allow an output pin to drive more than 25mA
  - Translation: for the time being, use 200 ohm or greater resistors in series with any LED that you attach to the PIC
- Always check your circuit before power up
  - Be careful not to short power/ground
  - Do not reverse power and ground

Next Time

- Lab: follow up on PIC programming (you must be checked out on this)
- Thursday lecture: continue with PIC architecture