# MOTOR CONTROL TUTORIAL

WEB PAGE: WWW.UMICH.EDU/~BOBDEN/

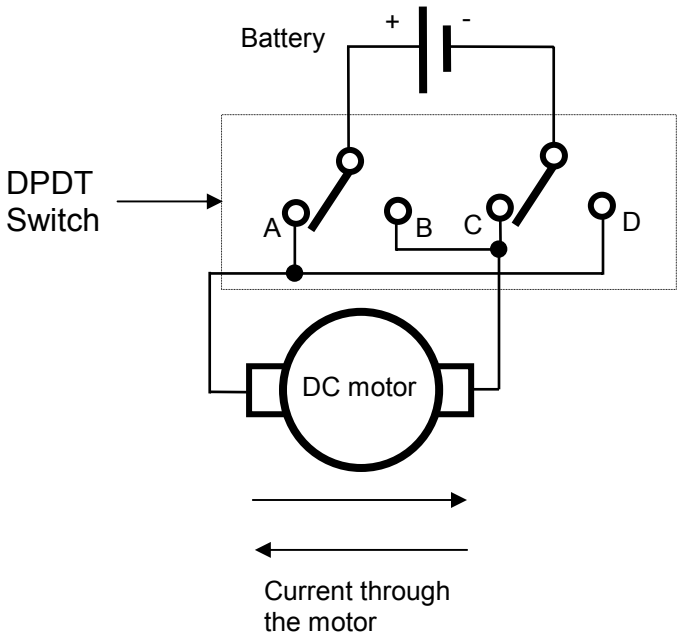
## MOTOR AND SOLENOID CONTROL:

DC motors, step motors, and solenoids are all similar in the sense that they are *inductive* loads that are difficult or impossible to drive directly using the low power analog and digital ICs that are common in most devices (and common in this Laboratory). A *solenoid* is just a coil of wire surrounding an iron core. The core can generally slide in and out of the coil along the axis. When a current is applied, the solenoid generates a force that tends to draw the iron core toward the center of the solenoid. This is the stuff of introductory physics texts, so if you are interested in the underlying physics, that's where you'll have to go for more information. From a practical standpoint, you can cleverly arrange the coil and the iron core such that when current is applied to the electric coil, a mechanical rod is either pushed away from or pulled toward the solenoid. Thus, solenoids are a simple *linear actuator*: they produce force and motion along a straight line, usually for very short distances, typically less than 1 cm. In general, you either turn them ON or OFF to make something happen. They are so useful that you can find then all over the place: electric door locks on cars, water valves for washing machines and sprinkler systems, pinball games, just about anywhere where you need to make a binary mechanical movement. Thus, controlling a solenoid is simply a matter of turning a relatively large electric current ON or OFF.

Electric motors, on the other hand, have much more sophisticated and interesting control aspects. *Stepper motors* typically require at least two sets of pulses feeding two coils within the motor. The sequence in which the coils are energized determines the motor direction. The rate at which the pulses are applied determines the motor speed (steps per second). Stepper motors are somewhat difficult to control unless you have a microprocessor handy, so we will discuss those later. You can find out more about stepper motors by reading the suggested primer on the web (see above). DC Brush motors are the simplest and most common motors on the planet. You will find these motors everywhere from slot cars (and many other toys), to automobile components, to battery-powered appliances (e.g. screw drivers, shavers). The control of DC motors basically boils down to two issues:

Control of **direction** (make the motor spin one direction or the other)
Control of **current** (depending on the type of load, this relates to speed, etc.)

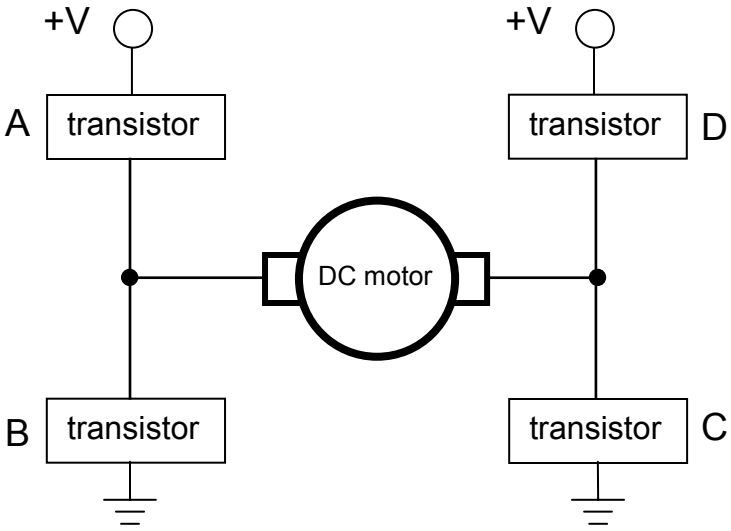
Control of direction is easy: you arrange switches to route the current either of two ways through the motor. This can be accomplished with what is known as a *double pole – double throw* switch. This type of switch has two separate conducting paths (*double pole*), and two possible output positions for each path (*double throw*). A double pole – double throw switch is usually called DPDT for short. It can be wired as shown:




The "+" and "-" terminals of the battery connect to the two conducting paths of the DPDT switch (the "common" terminals on the switch). The two conducting paths in the DPDT switch are mechanically coupled, so that the switch can be in one of two positions: connected to "A" and "C", or, alternatively, connected to "B" and "D". The switch is shown in the "A" and "C" position. With the switch in this position, conventional electric current will flow from the "+" terminal of the battery, through the DPDT switch to the "A" terminal, then through the motor from left to right (top arrow), then to the "C" terminal on the switch, then back to the "-" terminal of the battery, completing the electric circuit. Flip the switch to the opposite position, the current will flow in the opposite direction through the motor, and the motor will turn in the opposite direction. Simple as that.

So, to control motor direction, you simply control the direction of current flowing through the motor. Doing this with a mechanical switch is easy, but it is slow, and it is difficult or impossible to interface with a sophisticated controller. Modern motor controllers do this much better by using *power transistors* instead of switches. The transistors can be discrete parts, or they can all be built onto one large IC. When they are built onto one large IC for this purpose, it is often called a "Full Bridge" or "H-Bridge" motor controller, because the conductors and transistors are often arranged around the DC motor on the schematic in the form of the letter "H":

# MOTOR CONTROL & SENSING




Think of each of the 4 separate transistors as a switch: When you close "A" and "C", current will flow through the motor from left-to-right. When you close "DC and "B", electrical current will flow through the motor in the opposite direction (right-to-left), so the motor will go in the reverse direction. BUT, you need to be sure to use the transistors in a sensible manner: never activate "A" and "B" together (or "C" and "D"), or you will short out the power supply!

Advantages of using transistors over mechanical switches:

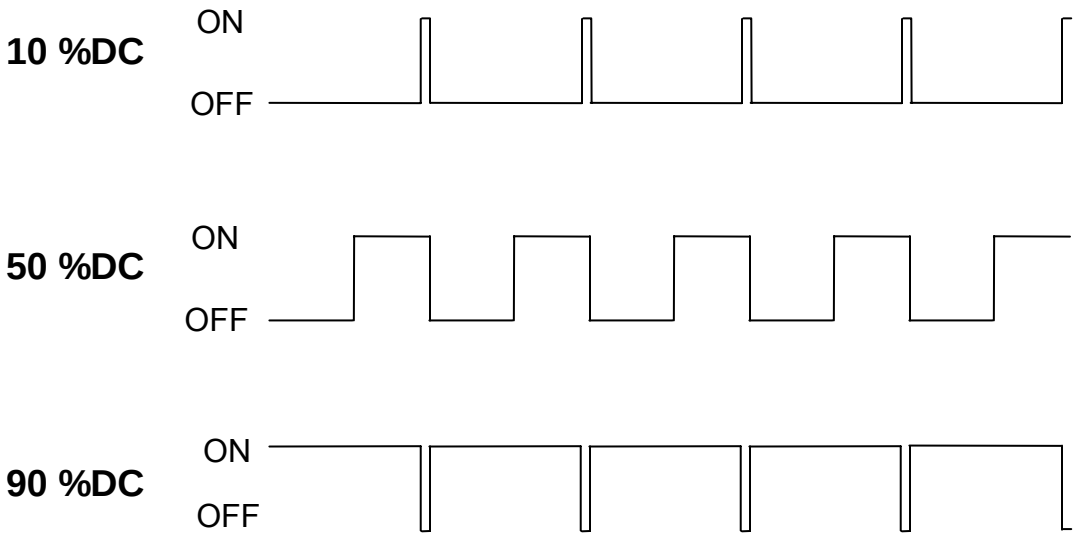
Transistors act very quickly (microseconds)
Transistors can be easily controlled by a microcontroller (or other interface)
Power transistors can be used (huge current)
In addition to controlling current (motor) direction, you can:
**Control total current** (i.e. motor power: torque and RPM)
Modes for current control: PWM (Pulse Width Modulation), Chopper amp.
**BRAKE**: Activate "B" and "C" to short both motor terminals to ground:
Motor acts as an electric generator with a huge load, so it acts as a BRAKE

CONTROL OF MOTOR POWER:

Basically, we control motor power (Torque and RPM) by controlling the amount of electric *current* that passes through the motor. There are two common ways to control the current flowing through a motor: PWM and Current Control using a current amplifier (such as a "chopper amplifier")

**PWM** (Pulse Width Modulation)

PWM essentially amounts to turning a motor (or solenoid, or any other load) ON and OFF very fast. You turn the load ON and OFF so fast that the system does not have time to respond dynamically with the system. The trick here is to control *how long* the relative ON and OFF times are. The ratio of ON time to Total time (Total time = ON + Off time) is called the **Duty Cycle**, and is often expresses as a % (%Duty Cycle, or %DC).

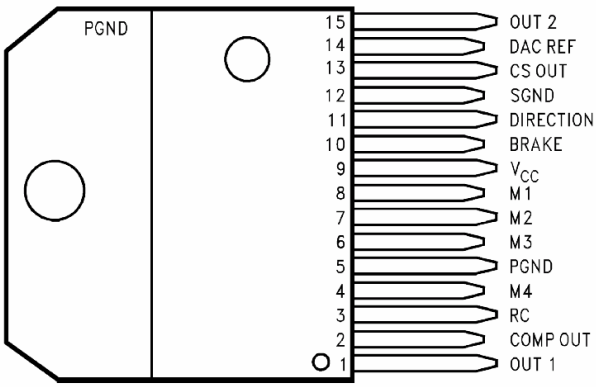



To achieve PWM control of motor current, you simply control the amount of time each pulse is ON during each pulse cycle. Note, generally the pulse period remains constant, you only adjust (modulate) the *width* of each ON pulse. So, you get the same number of pulses in any given time interval, but with increasing %DC, the motor is ON more of the time, so it gets more power. You can implement PWM with a microcontroller and discrete transistors (just use the microcontroller to control both which transistors are active, and to control their %DC). Alternatively, you can get pre-packaged integrated circuits (ICs) that have a built-in PWM function, such as the L298 (made by ST-Microelectronics).

**Current Amplification & Control** (using a "chopper" amplifier)

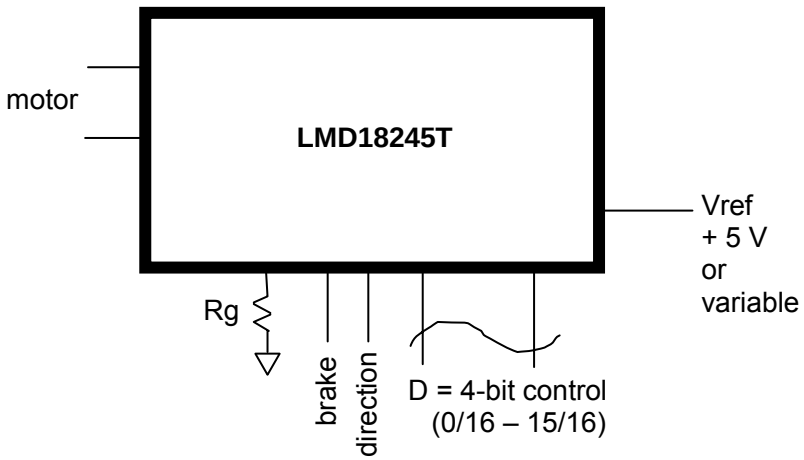
Chopper current amplifiers can be packaged together with the transistors in an "H" bridge configuration on a single IC to produce a motor driver chip, often called a *Full Bridge Motor Driver*. One excellent example is the LMD18245, made by National Semiconductor. Unlike PWM, chopper amplifiers use internal current monitoring and feedback to control the electrical current flowing through a motor coil. This is a bit more sophisticated than PWM, but the level of motor control that you can achieve is superb. We will use the LMD18245 for your laboratory exercises. If you wish to read more about how this IC functions, you can download the datasheet by going to the National Semiconductor web page (www.national.com) and searching for the chip using the part number and their search engine.

**Connection Diagram**




**Top View**
**15-Lead TO-220 Molded Power Package**
**Order Number LMD18245T**

The Motor Driver Chip shown above has 15 pins, each of which is connected to configure the chip to do what you want it to do. This will be done for you. The resulting circuit will have the following features:




$$\text{Current to motor} = \frac{V_{ref}\, D}{0.00025\, R_g}$$

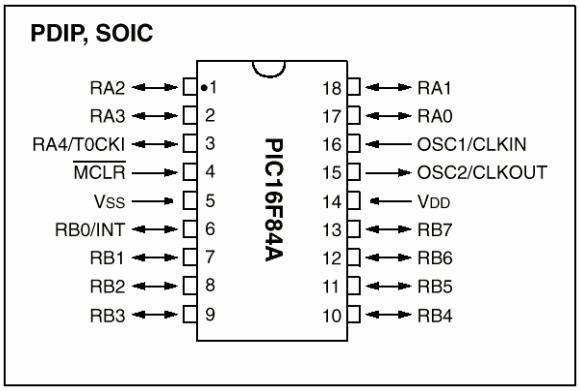
So, if you set the $V_{ref}$ to +5.0 V and use the maximum value of **D** (15/16), you can easily select the gain resistor $R_g$ to set the maximum motor current ($I_{max}$) as:

$$R_g = 18750 / I_{max}$$

# MOTOR CONTROL & SENSING

MICROCONTROLLERS:

Microcontrollers are just about everywhere because they are simple, robust, inexpensive, and incredibly powerful. Fortunately, they are also very simple to program and to implement in electromechanical designs. Shown at right is the pinout diagram of a very inexpensive microcontroller. They can be purchased on-line for about $4.00 each. You will use one of these to control the motor in your project.




Here is the basic external anatomy of the PIC16F84A microcontroller:

Note that each pin is numbered on this diagram: 1-18
Pins 15 & 16 are the Oscillator. They set the clock speed (0-20 MHZ)
Each internal instruction executes in 4 clock cycles
Pin 14 is +5V positive power
Pin 5 is Ground
Pin 4 is the Master Clear (resets the microcontroller if it goes to GROUND)
Pins 17, 18, 1, 2, & 3 are Register A (RAx): They are just 5 bits of digital I/O
Pins 6-13 are Register B (RBx): This is a full 8-bit digital I/O (input/output) port

What can you do with this (or any other) microcontroller?
**INPUT:** You can **read** the digital state of any of the register pins (RAx or RBx).
**OUTPUT:** You can **write** a digital value to any or all of the register pins (RAx or RBx).
**Logical Operations:** conditional branching, time delays, simple mathematical operations, etc., based on the input states or on internal numerical values.

This is pretty much how we control the entire world nowadays.

Programming the Microcontroller:

Programming a microcontroller is remarkably simple once you have everything set up. First you need the programmer hardware (shown at right). This is a small device that plugs into the serial port of a computer. Then you need the drivers for the programmer hardware. Together, these two things will "burn" your program onto the microcontroller. The program will remain on the microcontroller, intact, whether or not the power is turned on. This is because the program on the microcontroller is stored in "non-volatile" memory, which is retained regardless of the power applied to the microcontroller.

# MOTOR CONTROL & SENSING

Programming: We will use a "C" compiler, which is actually very simple. The compiler is made by Custom Computer Services, Inc. It is called PIC-C. The initial motor control program will be posted on Prof. Dennis' web page, and will be already burned onto the microcontrollers, so everything should work when you first hook it up.

"C" Programming: The list of statements in PIC-C language is shown below to the left. The numerical constants that you can use are shown to the right.

**Statements**

| STATEMENT | EXAMPLE |
|---|---|
| **if (expr)** stmt; [**else** stmt;] | `if (x==25)`<br>`  x=1;`<br>`else`<br>`  x=x+1;` |
| **while** (expr) stmt; | `while (get_rtcc()!=0)`<br>`  putc('n');` |
| **do** stmt **while** (expr); | `do {`<br>`  putc(c=getc());`<br>`} while (c!=0);` |
| **for** (expr1;expr2;expr3) stmt; | `for (i=1;i<=10;++i)`<br>`  printf("%u\r\n",i);` |
| **switch** (expr) {<br>**case** cexpr: stmt; //one or more case<br>[**default**:stmt]<br>... } | `switch (cmd) {`<br>`  case 0: printf("cmd 0");`<br>`  break;`<br>`  case 1: printf("cmd 1");`<br>`  break;`<br>`  default: printf("bad cmd");`<br>`  break; }` |
| **return** [expr]; | `return (5);` |
| **goto** label; | `goto loop;` |
| label: stmt; | `loop: I++;` |
| **break;** | `break;` |
| **continue;** | `continue;` |
| expr; | `i=1;` |
| ; | `;` |
| {[**stmt**]}<br>^<br>\|<br>Zero or more semicolon separated | `{a=1;`<br>`  b=1;}` |

Note: Items in [ ] are optional

| **Constants:** | |
|---|---|
| 123 | Decimal |
| 0123 | Octal |
| 0x123 | Hex |
| 0b010010 | Binary |
| 'x' | Character |
| '\010' | Octal Character |
| '\xA5 | Hex Character |
| '\c' | Special Character. Where \c is one of:<br>\n Line Feed- Same as \x0a<br>\r Return Fee - Same as \x0d<br>\t TAB- Same as \x09<br>\b Backspace- Same as \x08<br>\f Form Feed- Same as x0c<br>\a Bell- Same as \x07<br>\v Vertical Space- Same as \x0b<br>\? Question Mark- Same as \x3f<br>\' Single Quote- Same as \x60<br>\" Double Quote- Same as \x22<br>\\ A Single Backslash- Same as \x5c |
| "abcdef" | String (null is added to the end) |
| "abc" "def" | Strings with auto-concatenation (same result as above) |

| **Identifiers:** | |
|---|---|
| ABCDE | Up to 32 characters beginning with a non-numeric. Valid characters are A-Z, 0-9 and _ (underscore). |
| ID[X] | Single Subscript |
| ID[X][X] | Multiple Subscripts |
| ID.ID | Structure or union reference (First ID is a variable) |
| ID->ID | Structure or union reference (First ID is a pointer to variable) |

The mathematical operators and built-in functions are shown below:

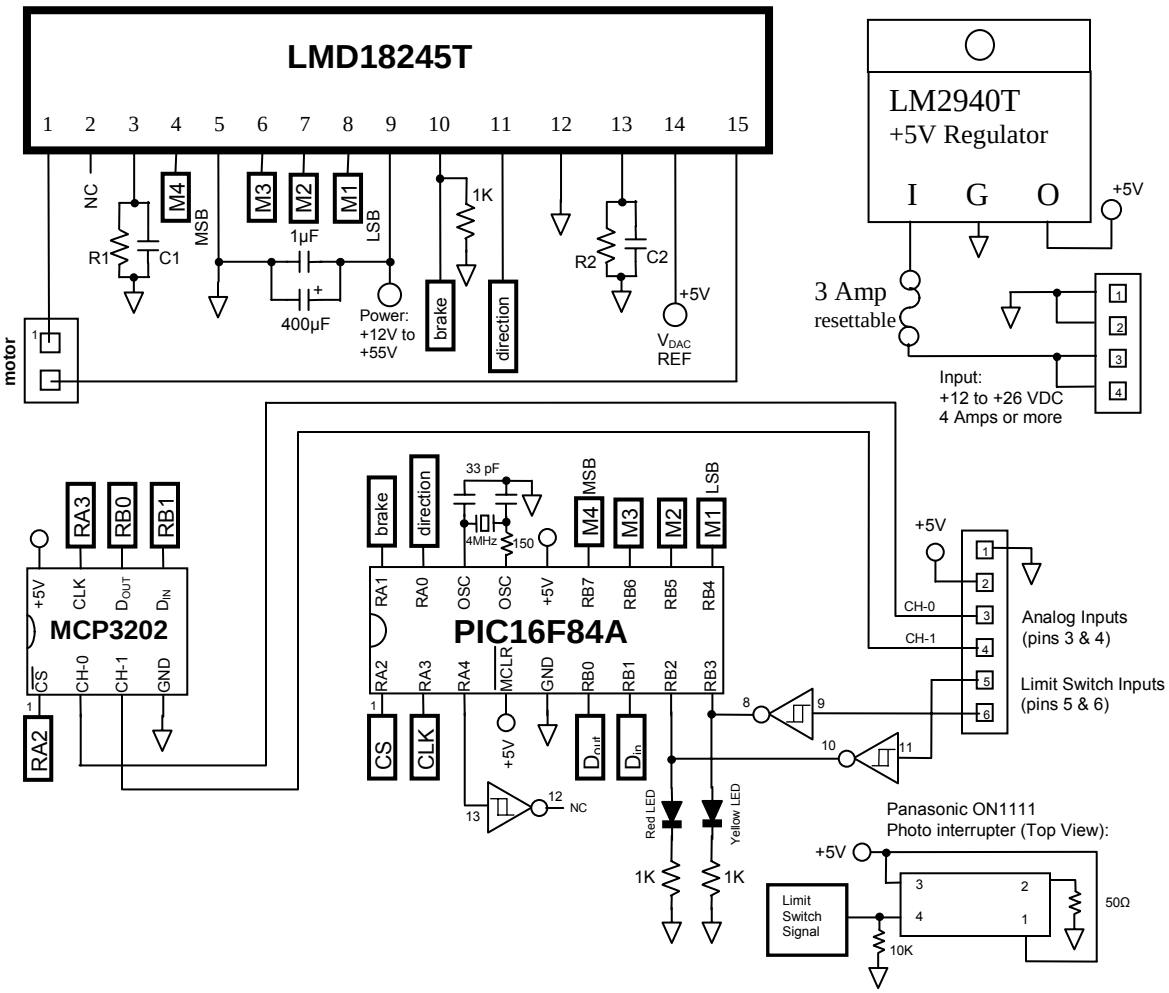
**Operators**

| | |
|---|---|
| + | Addition Operator |
| += | Addition assignment operator, x+=y, is the same as x=x+y |
| &= | Bitwise and assignment operator, x&=y, is the same as x=x&y |
| & | Address operator |
| & | Bitwise and operator |
| ^= | Bitwise exclusive or assignment operator, x^=y, is the same as x=x^y |
| ^ | Bitwise exclusive or operator |
| \|= | Bitwise inclusive or assignment operator, xl=y, is the same as x=xly |
| \| | Bitwise inclusive or operator |
| ?: | Conditional Expression operator |
| - - | Decrement |
| /= | Division assignment operator, x\=y, is the same as x=x/y |
| / | Division operator |
| == | Equality |
| > | Greater than operator |
| >= | Greater than or equal to operator |
| ++ | Increment |
| * | Indirection operator |
| != | Inequality |
| <<= | Left shift assignment operator, x<<=y, is the same as x=x<<y |
| < | Less than operator |
| << | Left Shift operator |
| <= | Less than or equal to operator |
| && | Logical AND operator |
| ! | Logical negation operator |
| \|\| | Logical OR operator |
| %= | Modules assignment operator x%=y, is the same as x=x%y |
| % | Modules operator |
| *= | Multiplication assignment operator, x*=y, is the same as x=x*y |
| * | Multiplication operator |
| ~ | One's complement operator |
| >>= | Right shift assignment, x>>=y, is the same as x=x>>y |
| >> | Right shift operator |
| -> | Structure Pointer operation |
| -= | Subtraction assignment operator |
| - | Subtraction operator |
| sizeof | Determines size in bytes of operand |

**BUILT-IN FUNCTIONS**

| **Built-In Function List By Category** | | | |
|---|---|---|---|
| **RS232 I/O** | | **Parallel Slave I/O** | |
| getc() | 72 | setup_psp() | 112 |
| putc() | 94 | psp_input_full() | 93 |
| gets() | 73 | psp_output_full() | 93 |
| puts() | 95 | psp_overflow() | 93 |
| printf() | 92 | **Delays** | |
| kbhit() | 80 | delay_us() | 67 |
| set_uart_speed() | 106 | delay_ms() | 66 |
| **I2C I/O** | | delay_cycles() | 66 |
| i2c_start() | 75 | **Processor Controls** | |
| i2c_stop() | 75 | sleep() | 121 |
| i2C_read | 74 | reset_cpu() | 99 |
| i2c_write() | 76 | restart_cause() | 99 |
| i2c_poll() | 73 | disable_interrupts() | 68 |
| **Discrete I/O** | | enable_interrupts() | 69 |
| output_low() | 89 | ext_int_edge() | 70 |
| output_high() | 89 | read_bank() | 96 |
| output_float() | 88 | write_bank() | 128 |
| output_bit() | 87 | **Bit/Byte Manipulation** | |
| input() | 77 | shift_right() | 119 |
| output_X() | 90 | shift_left() | 118 |
| input_X() | 77 | rotate_right() | 102 |
| port_b_pullups() | 91 | rotate_left() | 101 |
| set_tris_X() | 105 | bit_clear() | 63 |
| **SPI two wire I/O** | | bit_set() | 64 |
| setup_spi() | 113 | bit_test() | 64 |
| spi_read() | 122 | swap() | 127 |
| spi_write() | 122 | make8() | 84 |
| spi_data_is_in() | 121 | make16() | 85 |
| | | make32() | 85 |
| | | **Capture/Compare/PWM** | |
| | | setup_ccpX() | 108 |
| | | set_pwmX_duty() | 103 |

## SCHEMATIC FOR THE MOTOR DRIVER BOARD:




NC = no connection
M4, M3, M2, M1: 4 bit data for current control (M4 = MSB)

R1 = 20K, C1 = 1000 pF: sets the monostable chopper period to ~1.1 RC ~ 20 µs
R2 = 4.7K, C2 = 2200 pF, sets peak current at 3.75 Amps
Alternatively, add a parallel resistor to R2 to increase the output current:

$R_g = 18750/I_{max}$increase

Thus, to set $I_{max}$ = 6.0 Amps, add a parallel resistor = ~9.3K to R2

Sample “C” code for the microcontroller:

```
//         mot-lmd1.c      RGD 8/15/02

#include <16F84a.H>
#use delay (clock=4000000)   //  4 MHz crystal
#byte PORTA = 0x05
#byte PORTB = 0x06


unsigned int i;              // buffer variable for the detected state of PORTb
unsigned int j;
unsigned int k;
unsigned int address;
unsigned int step;           //  this is the counter for keeping track of which step is
current

short int flip;   // variable for flashing LED


////////////////////////////////////////
// SUBROUTINES



void init_ports(void) {

  SET_TRIS_A(0b00000000);  // PORTA = all outputs
  SET_TRIS_B(0b00000000);  // PORTB = all outputs
  SETUP_COUNTERS(RTCC_INTERNAL, WDT_1152MS);  //  set Watch Dog
Timer prescaler to 1152 ms


  // Default output values:
    PORTA = 0b00000000;   //  all bits are OUTPUTs
    PORTB = 0b00000000;   //  all bits are OUTPUTs

  // Default variable values
  step = 0;
  flip = 0;
  i = 0;
  j = 0;
  k = 0;
  address = 0;
}
```

```
////////////////////////////////////////////////////////////////////

void step_0(void)  {        //  this is the first of 16 steps
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b00001111;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_1(void)  {        //
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b01101110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_2(void)  {        //
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b10111011;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_3(void)  {        //
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b11100110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_4(void)  {        //
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
```

```
  OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
  PORTB = 0b11110000;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_5(void)  {        //
  restart_wdt();           //  Reset the WDT
  OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
  OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
  PORTB = 0b11100110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_6(void)  {        //
  restart_wdt();           //  Reset the WDT
  OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
  OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
  PORTB = 0b10111011;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_7(void)  {        //
  restart_wdt();           //  Reset the WDT
  OUTPUT_BIT(PIN_A0, 1);    //  set motor 2 direction
  OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
  PORTB = 0b01101110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////

void step_8(void)  {        //
  restart_wdt();           //  Reset the WDT
  OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
  OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
  PORTB = 0b00001111;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////
```

```
void step_9(void)  {        //
   restart_wdt();           //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
   PORTB = 0b01101110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////////

void step_10(void)  {       //
   restart_wdt();           //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
   PORTB = 0b10111011;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////////

void step_11(void)  {       //
   restart_wdt();           //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 0);    //  set motor 1 direction
   PORTB = 0b11100110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////////

void step_12(void)  {       //
   restart_wdt();           //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b11110000;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////////

void step_13(void)  {       //
   restart_wdt();           //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b11100110;       //  set the chopper amp values for motors 1 & 2
```

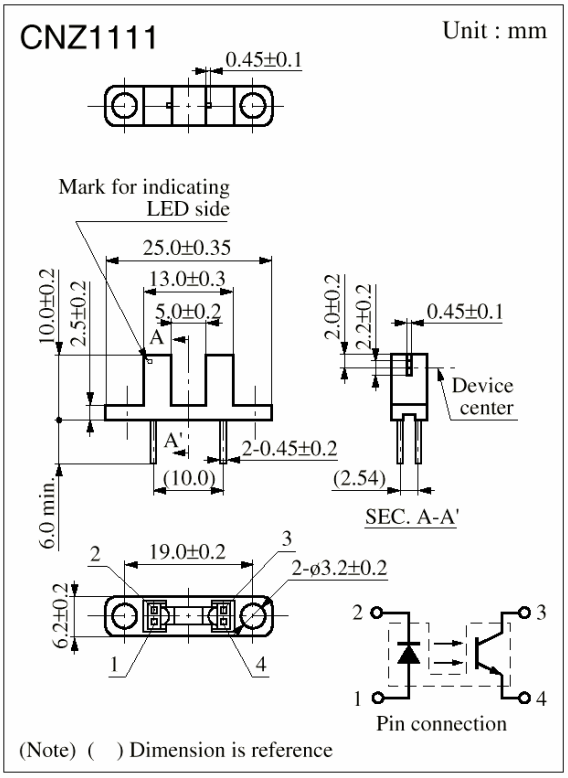
```
}

////////////////////////////////////////////////////////////////////////

void step_14(void)  {       //
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b10111011;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////////

void step_15(void)  {       //  this is the last of 16 steps
   restart_wdt();          //  Reset the WDT
   OUTPUT_BIT(PIN_A0, 0);    //  set motor 2 direction
   OUTPUT_BIT(PIN_A3, 1);    //  set motor 1 direction
   PORTB = 0b01101110;       //  set the chopper amp values for motors 1 & 2
}


////////////////////////////////////////////////////////////////////////

void dwell(void)  {         //  this is what the controller does while waiting between
steps
   restart_wdt();          //  Reset the WDT
   delay_us(i);            //  delay between steps sets motor speed
   delay_us(i);            //  delay between steps sets motor speed
}

////////////////////////////////////////////////////////////////////////



////////////////////////////////////////
////////////////////////////////////////
////////////////////////////////////////
```

```
// PIC16F84A goes here at RESET

void main() {

     restart_wdt();         // Reset the WDT
     init_ports();          // Initialize ports
     i = 255;
     j = 0;

 CYCLE:                     // Run continuously
  restart_wdt();            // Reset the WDT

  switch(step) {
     case 0: step_0();
        break;
     case 1: step_1();
        break;
     case 2: step_2();
        break;
     case 3: step_3();
        break;
     case 4: step_4();
        break;
     case 5: step_5();
        break;
     case 6: step_6();
        break;
     case 7: step_7();
        break;
     case 8: step_8();
        break;
     case 9: step_9();
        break;
     case 10: step_10();
        break;
     case 11: step_11();
        break;
     case 12: step_12();
        break;
     case 13: step_13();
        break;
     case 14: step_14();
        break;
     case 15: step_15();
        break;
  }
```

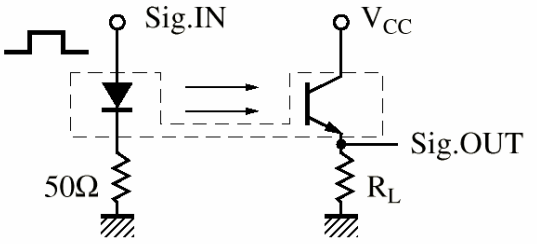
```
  dwell();                    //  delay between steps to set motor speed

  step = step + 1;
  if(step >= 16) step = 0;        //  set upper limit for step counter
  if(step == 0) i = i - 5;        //  speed up the step rate

  if(i <= 15) i = 15;             //  set maximum step rate


 goto CYCLE;


} // main
```

PHOTO INTERRUPTERS:

Photo interrupters are extremely common devices that are used in many places, including detectors for people passing through doorways, and to detect the movement of the ball within a computer mouse. A photo interrupter is just a paired optical *emitter* and *detector*. The emitter is usually an LED, which emits infrared light (so you can not see it). The detector is usually just a phototransistor, pointed directly at the output of the LED. The LED is left ON continuously, so it is always detected by the phototransistor. When an object passes between the emitter and detector, the light beam is interrupted and the phototransistor can no longer detect the light from the LED. This results in a loss of current through the phototransistor, which we can easily detect.




For your laboratory exercise, we will use the photo interrupter shown at the right: the Panasonic CNZ1111. The pin connections are shown at the lower right of the top figure. In the bottom figure you can see the simple electrical circuit that will be used to hook up the photo interrupter. The output signal (Sig. OUT) will be used as a logic level signal (ON or OFF) to tell the microcontroller when your device has reached the end of travel. Thus, we will use two photo interrupters as *limit switches*. When the microcontroller detects that the light path has been interrupted, it will reverse the direction of the DC motor that is powering your system. Since you are using a microcontroller, you can do many more sophisticated things than just changing direction: you can gradually slow the system by ramping the power down; you can stop for a fixed delay then reverse or continue; you can stop the motor and activate the BRAKE function on the motor driver IC until you receive another command; or you can do just about anything else you can think of.



## What you will do:

Locate this tutorial on the web and download it
(http://www-personal.umich.edu/~bobden/bob_me350.html)

Get the Motor Controller from Prof. Dennis
Get two photo interrupters from Prof. Dennis

Design and build your Project, including the photo interrupters as limit switches

Connect the Motor Controller as directed to the motor and sensors in your project (It should operate…the crude software will already be on the microcontroller)

Improve the performance of your system by making modifications to the control algorithm that is programmed on your microcontroller.