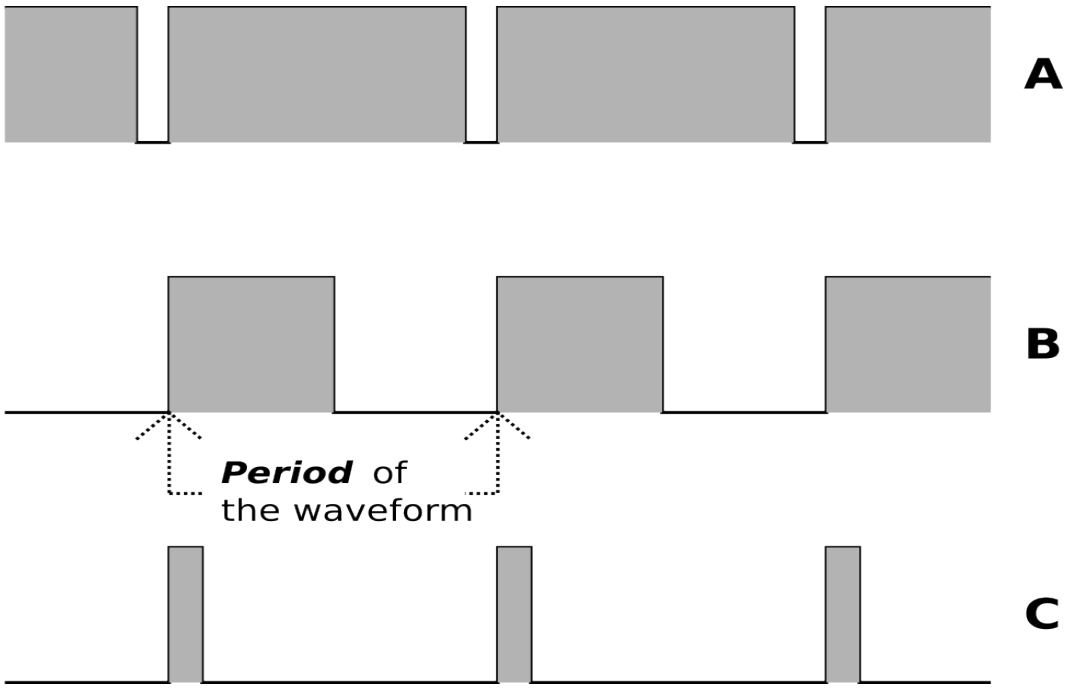
Digital Control – Microsystems

Part 5: Pulse Width Modulation - PWM

PWM

It's pretty much common sense that if you turn a motor on and off fast enough, say one running a fan, the fan's speed is going to be neither fully off nor fully on. It would run at a reduced speed, and that speed would reflect the average amount of time the switch was on. This is the essence of PWM, **P**ulse **W**idth **M**odulation. PWM is a way of getting analog (continuously variable) control over some types of devices. Motors, incandescent lights, and LED's are three commonly PWM'ed devices.



A PWM line controlling a motor's speed. The shaded area represents "on" time. A is 90% on, B is 50% on, and C is 10% on.

Some MCU's like the PIC12F683 have built-in circuitry for driving an output line as a PWM controller. When selecting a PIC for your project, look on the datasheet for a pin named CCP, which stands for **C**apture **C**ompare **P**WM. It is possible to do PWM without hardware support, but it's much less work to use the CCS compiler libraries which are written for the built-in peripheral. PWMing multiple lines simultaneously without hardware support can require "scheduling" the rest of your MCU's code around the PWM and can be a challenge.

The 12F683 has one CCP pin.

PWM is based in timing, and in computers timing is based in counting. All microprocessors require an accurate oscillator which produces a stream of always identical pulses. This is the *system clock*. In our use of the PIC12F683, we're taking

advantage of the “good enough” built-in oscillator which produces an 8 Megahertz (8 million pulses per second) clock. This square wave is the master drum beat, the metronome, which sequences almost all of the MCU's inner workings. The PIC's built-in timers for instance, function by counting clock pulses and the pulses of other counters which are counting clock pulses. The interconnection of these counters is in part arranged by the setup code mentioned in the source code below. The PWM diagram above shows three duty cycles. The duty cycle is the percent on-time. Fully on is 100%, and fully off is 0% duty cycle. The beginning of an “on pulse” is called its *rising edge*. The time between rising edges is the same for all three of the waveforms. The time from rising edge to rising edge in a waveform is the waveform's *period.* Period is the reciprocal of frequency. If the time from rising edge to rising edge in the same waveform is 1 millisecond, the frequency is 1 kilohertz. This is the PWM frequency used in analog2.C, below. For comparison, the period of the MCU's built-in clock is 1 / 8000000 or 0.000000125 seconds (125 nanoseconds).

period = 1 / frequency
frequency = 1 / period

PWM does not (necessarily) change the frequency of the output. If you set up the PIC for a low PWM frequency which you can hear while the MCU controls a motor's speed, you will hear the timbre of the motor's vibration become harsher with increasing duty cycle (increasing power), but the pitch will not change. This sound is not due to the rotation of the motor's armature, rather to microscopic vibrations of the armature as it reacts to the motor magnets. It's similar to a poor quality loudspeaker in this regard. Generally, if conditions allow, most people prefer to choose a PWM frequency high enough that it won't be audible (above 20kHz).

On to cases.

Three lines of setup code must be present to activate the PWM as an output:

```
We're using the internal oscillator
#use delay(clock=8000000)

Designate the Capture and Compare pin as PWM output
setup_ccp1(CCP_PWM);

Sets the frequency of the PWM
setup_timer_2(T2_DIV_BY_16, 127, 1);
```

Using the PWM: the value of i becomes the PWM value

```
set_pwm1_duty(i);
```

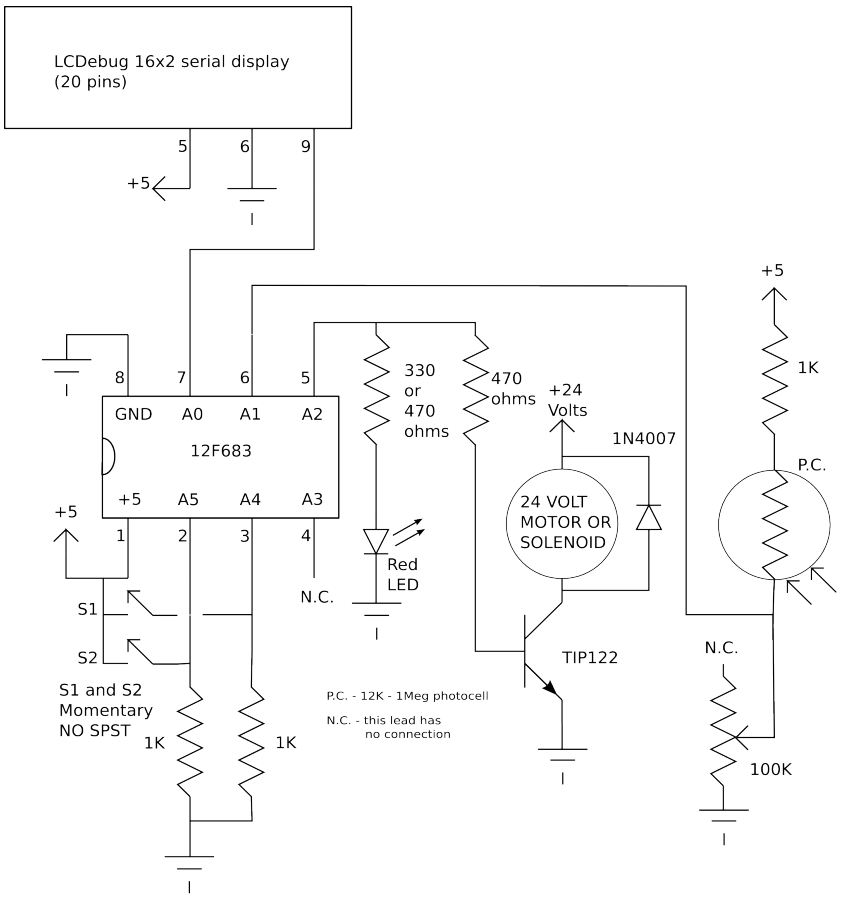
Part 5: Pulse Width Modulation - PWM

```
//analog2.c -  test of A2D with PWM

// read analog port
// print value of analog port 1 (PIC12F683 pin 6)
// set pwm value on CCP pin (PIC12F683 pin 5)

#include <12F683.h>
#device ADC=8
#include <STDLIB.h>
#fuses INTRC_IO,NOWDT,NOPROTECT,NOMCLR,NOBROWNOUT,NOIESO,NOFCMEN
#use delay(clock=8000000)
#use RS232(Baud=9600,Xmit=PIN_A0)  //PIN_A0 (PIC12F683 pin 7) sends to LCDebug
                                   //sets up the putc() and printf() functions

int i;

void main(void){
      setup_adc_ports(sAN1);              //make analog port 1 active as input
      setup_adc(ADC_CLOCK_INTERNAL);      //analog clock setup

      setup_ccp1(CCP_PWM);                //Enable Pulse Width Modulator Mode
      setup_timer_2(T2_DIV_BY_16, 127, 1);

      putc(12);                     //home and clear the display

      while(1){

            set_adc_channel(1); //select the channel, set A2D to pin 6 (GP1 as
AN1)
            delay_us(100);          //delay 100 microseconds for A2D to stabilize
            i = read_adc();         //take the reading pin 6

            printf("%3u\r",i);      //print a 3 digit number and Carriage Return

            set_pwm1_duty(i);
      }
}
```

We're using the PWM in 8-bit mode. This gives us 256 steps of control from full off (0) to full on (255). The PIC and the CCS compiler can also PWM with 10-bit resolution, which would give 1024 steps of control from full off (0) to full on (1023). The circuit for analog2.c builds on what has come before it. The CCP pin (called CCP1 in the datasheet) is PIC12F683 pin 5. We've dimmed an LED connected to that pin and now we'll use the same PWM code to control a motor speed. What we need to add to make that possible are a *transistor* and a *diode*. The type of diode is a 1N4007 rectifier, and the type of transistor is a TIP122 darlington. First, here's the schematic for the circuit.

Digital Control – Microsystems

Part 5: Pulse Width Modulation - PWM



There are only a very few ways to hook a load to a microcontroller. With the exception of lighting an LED, there will always have to be some kind of interface device between the MCU and the actuator or load. This is because the Input/Output (I/O) pins we've been connecting to can only supply the most feeble current, 10 to 20mA (milliamps, thousandths of an amp) maximum, while small to medium motors need hundreds of milliamps to several amps (in art making, 5 amps is on the big end). The most basic way to get a MCU output up to a usable amount of power is to amplify the output with a transistor.

Here is a small program which shows the difference in coding PWM with a 10-bit output versus an 8-bit output. The 10-bit version gives 1024 steps of output power as the control variable (long i) is changed from 0 to 1023 and back. The 8-bit version gives 256 steps of output power as the PWM control variable (int j) is changed from 0 to 255 and back. The 10 bit version is smoother than the 8-bit version because it take four times as many steps of change in the output to span the same 0 to 100% range of duty cycle. It also will change more slowly if the time between updates to the PWM control variable are the same as the 8-bit version. Normally the extra smoothness of the 10-bit variable isn't the best reason for using the 10-bit mode. Mostly it's a help in setting the span of the output range without losing too much adjustability because of chopping off the undesirable head and tail parts of the speed range.

```
// HBridge1.C  H-Bridge and relay reverser / speed control with PIC12F683 8-pin mcu

// Shows PWM motor control. Pot on A-to-D controls acceleration rate as motor goes from
```

Part 5: Pulse Width Modulation - PWM

```
// zero speed to full speed, reverses, and repeats. Output lines are PWM duty cycle
// and direction. Outputs can be connected to driver transistor and relay interface (cheap,
// fault-tolerant) or LMD18200T H-Bridge (small, modern).
//
// Pressing the pushbutton causes 3 trips through the while loop
//
// LMD18200T H-Bridge motor driver 3 Amps 55 Volts max

//LMD 18200 pins
//pin 1             bootstrap cap .01uF to pin 2
//pin 2             Motor lead 1 ; Bootstrap cap also to pin 1
//pin 3             direction - to PIC12F683 pin 3
//pin 4             brake input - ground
//pin 5             PWM input - to CCP1 pin PIC12F883 pin 5
//pin 6*     +V for chip AND motor 12 - 24VDC (NOT 5 volts)
//pin 7             ground
//pin 8             current sense - not connected
//pin 9             thermal flag - not connected
//pin 10     Motor lead 2 ; Bootstrap cap also to pin 11
//pin 11     bootstrap cap .01uF to pin 10
// *add 220uF and 1uF caps to ground at this pin

//12F683 PINS
// PIN 2 (PIN_A5) -- 1k pullup and pushbutton to ground
// PIN 3 (PIN_A4) -- direction
// PIN 5 (PIN_A2) -- PWM out
// PIN 6 (PIN_A1) -- AN1 analog input


#include <12F683.h>
#device ADC=8
#include <STDLIB.h>
#fuses INTRC_IO,NOWDT,NOPROTECT,NOMCLR,NOBROWNOUT,NOIESO,NOFCMEN
#use delay(clock=8000000)

long i; // variable to hold 10-bit PWM value
int j; // variable to hold 8-bit PWM value
int k; // scratchpad variable

int analog_val;
short direction = 0;

void main(void){
       setup_adc_ports(sAN1);            // setup PIC12F683 pin 6 as analog
       setup_adc(ADC_CLOCK_INTERNAL);
       set_adc_channel(1);        // ADC channel for 10k voltage divider pot

       setup_ccp1(CCP_PWM);       //Enable Pulse Width Modulator Mode

 /* Cycle time will be
       1/clock * 4 * t2div * pr2.
       for an 8 MHz clock,
       (1/8000000) * 4 * 255 * 1 = .0001275 sec = 127.5 microsec = 7843Hz */
```

```
	setup_timer_2(T2_DIV_BY_1, 255, 1);

// this while loop shows 10-bit mode. Variable containing PWM value must be a long

	while(1){

		while(input(PIN_A5)); // wait for button press

		for(k=0; k<3; k++){
			output_bit(PIN_A4, direction);

			for (i=0; i< 1024; i++){

 // calling set_pwm1_duty() with a long causes compiler to use 10-bit mode
				set_pwm1_duty(i);
				analog_val = read_adc(); //take the reading pin 6
				delay_ms(analog_val);
			}

			for (i=1023; i>0; i--){
				set_pwm1_duty(i);
				analog_val = read_adc();
				delay_ms(analog_val);
			}
			direction = (~direction);
		}
	}
}

// this while loop shows 8-bit mode. Variable containing PWM value must be an int

/*
	while(1){

		while(input(PIN_A5)); // wait for button press

		for(k=0; k<3; k++){
			output_bit(PIN_A4, direction);

			for (j=0; j<255; j++){

// calling set_pwm1_duty() with an int causes compiler to use 8-bit mode
				set_pwm1_duty(j);

				analog_val = read_adc();//take the reading pin 6
				delay_ms(analog_val);
			}

			for (j=255; j>0; j--){
				set_pwm1_duty(j);
				analog_val = read_adc();
				delay_ms(analog_val);
			}
```

```
                    direction = (~direction);
                }
        }
}
*/
```