Doctor Bots - Remote Presence Robots For MDs

SERVO MAGAZINE

FOR THE ROBOT INNOVATOR

www.servomagazine.com

June 2010

Simple Human Style Hands Within Your Robot's Grasp

✦ The Robotics Age
A Look Back At The Future Of Robotics

✦ Combat Zone
Dual-Differential RPM Sensing Or A Melty Brain/Translational Drift Robot

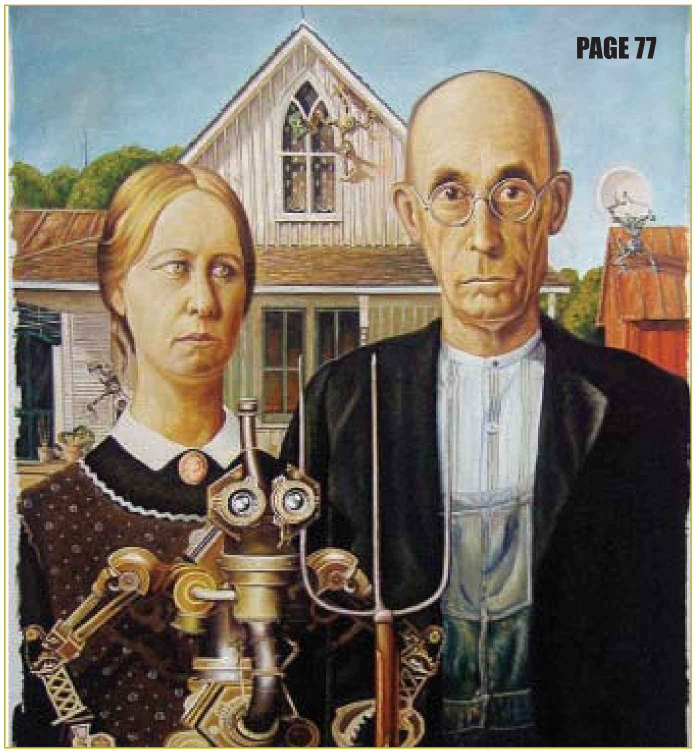
✦ Planning Out Your Software For Microcontrollers

U.S. $5.50 CANADA $7.00

06>

0 71486 02422 4

In This Issue ...



This article shows how you can implement PWM in your own robotics projects. By building these two simple devices — a general-purpose, low voltage light dimmer and a fan motor speed controller — you will clearly see the principles in action.

Controlling Motors and Lights With Pulse Width Modulation

By Jürgen G. Schmidt

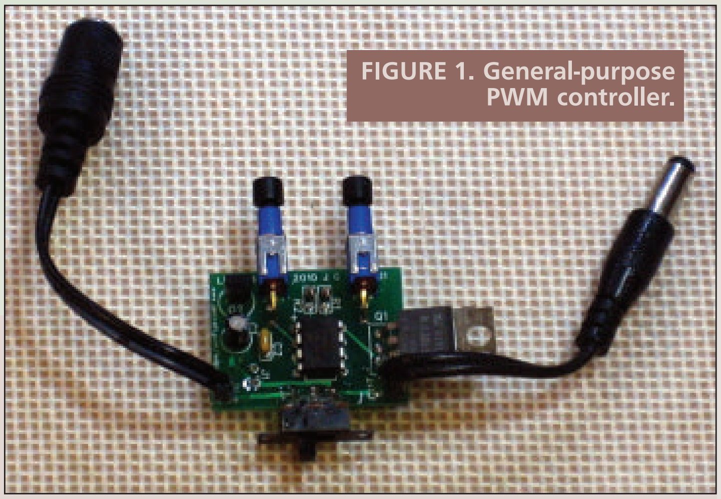


FIGURE 1. General-purpose PWM controller.

Background

There are switches everywhere! Most of them turn things either on or off, but sometimes we want something turned only partly on. Our cars are an example of this. We use an accelerator to control the engine speed. A simple on/off switch for the engine might be exhilarating but ultimately not very useful. The same is true for some of our lights, fans, and robots. We need to control how much they are turned on.

I had just finished replacing my halogen workbench light with an LED light strip I made from three watt LEDs. Now, I needed a dimmer for it. The LEDs work best at a constant voltage, so a PWM–based control seemed like the way to go. PWM (Pulse Width Modulation) consists of turning a device on and off very rapidly. The proportion of the time that the device is switched on is determined by the width of the pulses; therefore the term pulse width modulation. You will find PWM used in many projects, particularly with robot motors.

A simple alternative to PWM is to use a variable resistor in series with the device you want to control. This wastes a lot of power as heat, and as the voltage drops the device you are controlling may stop working too soon.

The 555 timer chip is frequently used to generate PWM signals. This chip — along with a few resistors and a capacitor, some calculations, and experimenting — will yield an inexpensive and effective result. You can see the basic astable circuit description and calculations in the 555 datasheet (see **References**). However, I wanted a design with pushbutton controls for increasing and decreasing the speed or brightness, and I needed the flexibility to easily experiment with the various parameters of a PWM signal. I prefer to do this in software rather than hardware.

I use PIC microcontrollers in many of my projects, frequently as a replacement for discrete components. In this case, I selected the PIC12F683 since it has a hardware PWM component. My first PWM controller was incorporated into the overall design of the project but I soon found that I could use PWM for all sorts of things. This led to the design of a general-purpose PWM controller that could be inserted between a power supply and a device — usually a light or a fan.

The PWM controller ended up as a parasitic system. By that, I mean that it does not need a separate power supply. Instead, it taps into the device that is being controlled. I built my first one between male and female 2.1 mm power connectors (**Figure 1**). I tried this out on some of the 12 volt fans I have on my desk and workbench. Previously, in order to reduce noise and avoid blowing papers away, I had adopted various strategies for slowing down the fans. These included using two fans in series and using six volt power supplies for 12 volt fans. A general-purpose PWM controller would give me consistent and variable control of my fans.

Design and Testing

In the course of designing and testing this general-purpose PWM controller, I learned a few things. Backing up a little, a PWM signal has two key parameters: a period and a duty cycle (**Figure 2**). The signal is a square wave with the bottom at zero volts and the top at the driving voltage. In the case of my fans, this is 12 volts. The ratio of the width of the top section to the width of the bottom section is the duty cycle. If they are both the same length, then the duty cycle is 50%. If you make the top wider and the bottom narrower, then the duty cycle increases and the fan turns faster or the light is brighter, whichever the case may be. The total of one off and one on cycle is the period. If the period is too long, you will see the light flicker. In the case of the motors, they will hum or squeal at certain frequencies. Selecting the proper period or frequency (periods per second) is important for a successful PWM controller.

The circuit for the PWM controller is fairly straightforward (**Figure 3**). Parts selection is not critical — you probably have most of the parts on hand. The power to the target device is interrupted by a power switching transistor such as a TIP31 or MJE3055t in a TO-220 package. This will introduce a drop of about half a volt between the input and output side. If you use a Darlington transistor such as a TIP120, the voltage drop will be slightly higher, but still under one volt.

Figure 4 shows how to connect a TO-220 NPN transistor. A 78L33 or 78L05 voltage regulator steals some power to supply the 12F683 which, in turn, provides a PWM signal to the transistor. Bypass capacitors C1 and C2 provide additional regulation. Input voltage to the overall system is limited by the specifications on the power transistor and the voltage regulator. The MJE3055T and TIP31, for example, are rated at 60 volts. The 78L33 and 78L05 voltage regulators, however, have a maximum input of 30 volts. The unregulated 12 volt transformers for my fans actually put out around 16 volts and the LED power supply I use puts out less than 30 volts, so I'm safe with these.

Two pins of the microcontroller are connected to pushbuttons that connect to ground. The pins are held high with pull-up resistors — anything from 4.7K to 100K will work. The pushbuttons increase and decrease the percent activation. I designed a 1 inch by 1-1/2 inch printed circuit board (PCB) to hold all the parts, along with an optional power switch. The desired power connectors can be added on the ends of the PCB or it can be wired inline with a project. For space reasons, I did not include a programming connector on the PCB or the breadboard. Instead, I used an eight-pin IC clip to avoid repeatedly pulling and inserting the chip during testing (**Figure 5**).

Since I had spare pins on the chip, I also connected one to a heartbeat LED and another to my PC serial port to get debugging output from my breadboard setup. (See the **sidebars** on Heartbeat LED and Terminal Programs.) This allows me to monitor the actual duty settings and their effect on the fan and light.

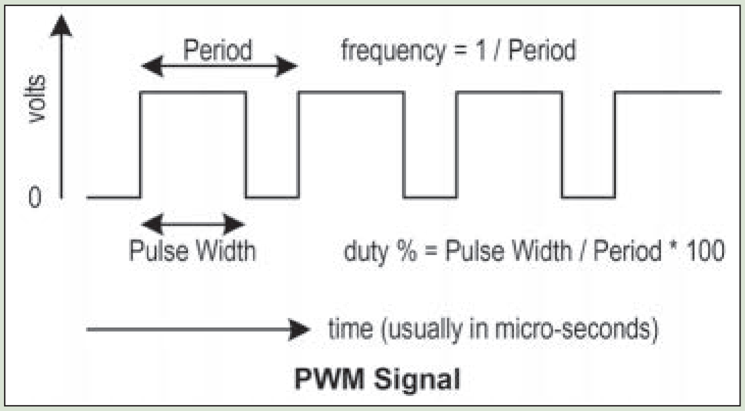

volts
0
Period
frequency = 1 / Period
Pulse Width
duty % = Pulse Width / Period * 100
time (usually in micro-seconds)
PWM Signal

FIGURE 2. PWM signal diagram.

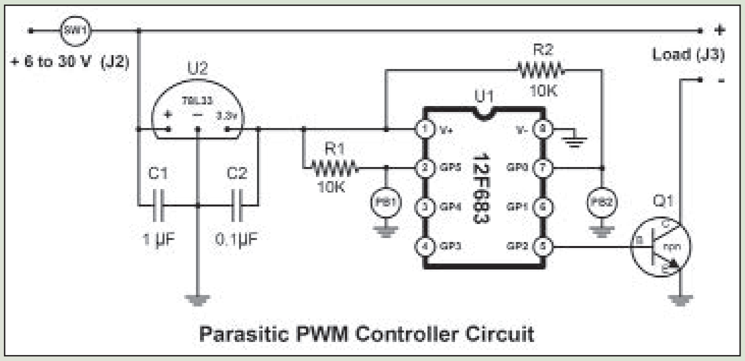
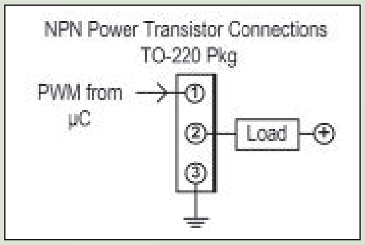


FIGURE 3. Circuit diagram.

FIGURE 4. T220 NPN power transistor connections.



Software

The initial software was written two years ago for the PICBASIC PRO compiler. This has a built-in HPWM function that requires three parameters: PWM channel, duty, and frequency. This program is shown in **Figure 6**.

My current microcontroller projects include handling multiple interrupts, as well as communicating with the Internet. PICBASIC isn't really suitable for this, so I recently switched to the Custom Computer Services (CCS) C compiler. While this change gave me additional flexibility, it also gave me additional complexity. Instead of a single PWM function, I now need to use three functions. One of these — the **setup_timer2()** function — requires some careful calculations to arrive at the right parameters for the desired frequency and duty. I had to read the datasheet and application notes several times to make sense of them.

To help in calculating the correct parameters for the CCS C PWM statements, I developed an Excel spreadsheet that provides the actual CCS C statements to include in the program. It has two boxes of calculations. The top box is for determining the results of various parameters using the

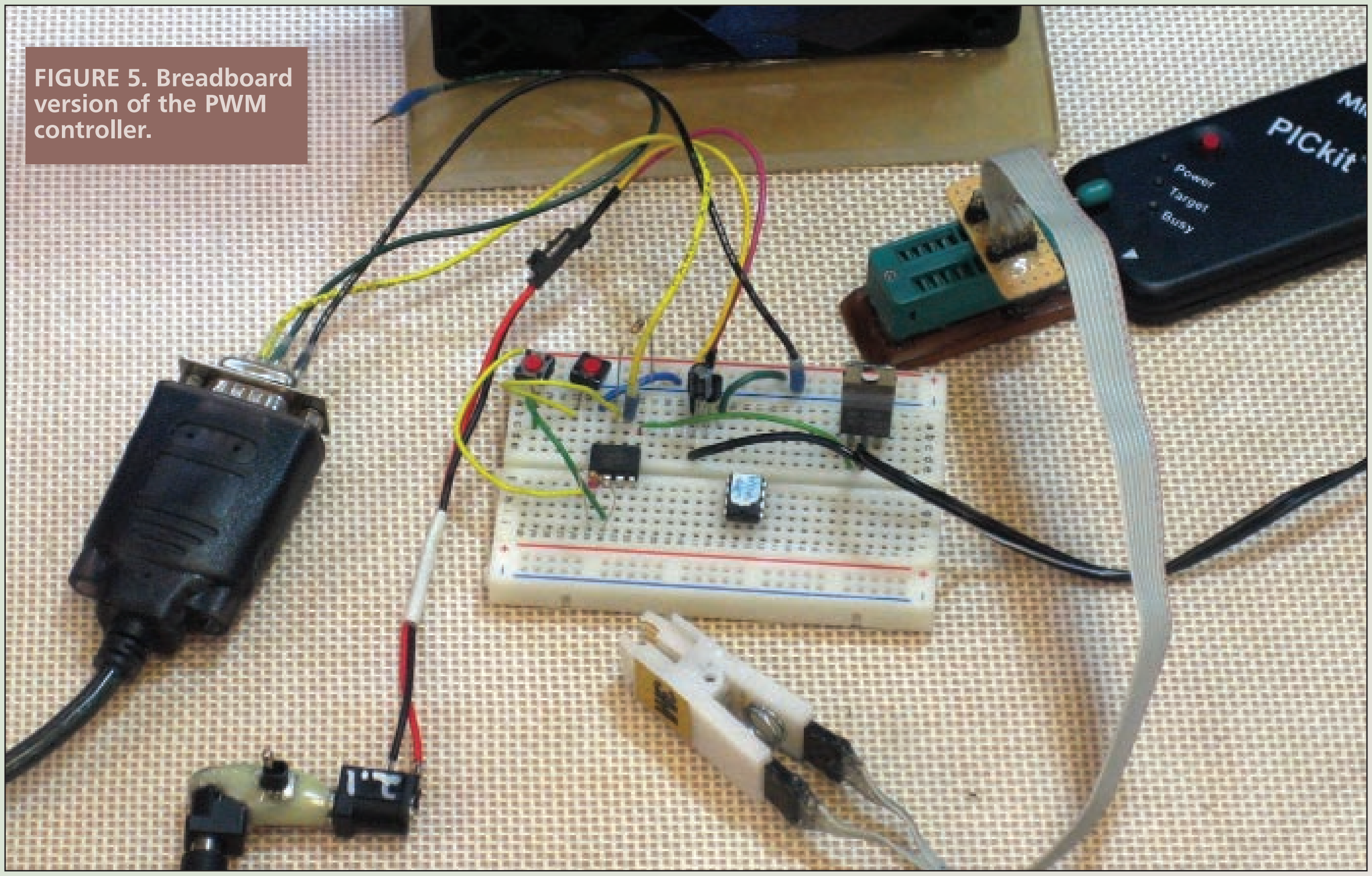


FIGURE 5. Breadboard version of the PWM controller.

HPWM calculations Excel spreadsheet.

Timer2 Calculations for PIC12F683 PWM
For CCS C Compiler

Miscellaneous Calculations

Crystal Frequency (Fosc)	8 000 000.	Hz	
Prescaler (1, 4, 16)	1.		
Cycle Time (Tosc)	0.000 000 5	Sec	2 000 000. Hz
PR2 (Timer2 period: 0-255)	200.		
PWM Period (overflow)	0.000 100 5	Sec	9 950.2 Hz
Duty Resolution	804.	←	1024 is maximum possible (10 bits)
PostScaler - sets interrupt (1-16)	16.	←	Not used for PWM
Interrupt period	0.001 608	Sec	
Crystal	8 000 000.	←	Enter crystal frequency (Hz)
Desired Period Frequency	16 000.	←	Enter desired PWM frequency

	PR2	Duty Range
Prescaler = 1	124.00	0 to 500
Prescaler = 4	31.00	0 to 128
Prescaler = 16	8.00	0 to 36

```
setup_timer_2(T2_DIV_1, 124, 1 );
setup_ccp1( CCP_PWM );
set_pwm1_duty( 250L );        // square wave output - 50% duty
```

formulas provided in the Microchip documentation. The bottom box accepts two values: a microcontroller oscillator frequency and a target PWM frequency. For this project, the oscillator frequency is the internally generated 8 MHz. I tested several computer fans — small and large — at different frequencies and discovered that a PWM frequency of 16 kHz worked best for most of them.

At this frequency, there was minimal humming or squealing. If you call up the spreadsheet, you will see these values entered. If your particular fan is noisy (especially at lower speeds), experiment with different frequencies. C statements are generated that yield the highest resolution or number of steps for the duty range. In this case, all three values for the Timer2 prescaler are allowed. If you change the target frequency to 4 kHz, you will see that one of the prescaler values is not allowed. For a detailed explanation of this, see the Microchip datasheets and application notes listed in the **References**. The spreadsheet is available on my website at **www.jgscraft.com/ledpwm** or the *SERVO* site at **www.servo magazine.com**.

In the course of converting the original PICBASIC program to C, I came

```
'—— 683PWMLED.bas ——
' PWM LED Dimmer
' Jurgen G Schmidt
' Target:         PIC12F683
' Compiler:       PICBasicPro 2.50b
' IDE:                     MicroCode Studio
'—- internal clock
'—- disable MCLR
@ DEVICE INTRC_OSC_NOCLKOUT, MCLR_OFF
DEFINE OSC 8
OSCCON = $70                 '8mhz for internal
ANSEL = 0                    'All Digital
CMCON0 = 7                   'Comparators off

'—— Port Assignments & Variables
pinHi   var     GPIO.0       'Brighter
pinLo   var     GPIO.5       'Dimmer
duty    var     BYTE

'—— Initialize System
read 0, duty                 'read EEPROM

MainLoop:
    hpwm    1,Duty,24000    'GPIO.2 at pin 5
    pause 200
    if pinHi = 0 then gosub Brighter
    if pinLo = 0 then gosub Dimmer
    goto MainLoop
end
'—————————————————
Brighter:
    if Duty => 255 then return
    Duty = Duty + 15
    write 0, duty   'save to EEPROM
    return
'—————————————————-
Dimmer:
    if Duty = 0 then return
    Duty = Duty - 15
    write 0, duty   'save to EEPROM
    return
'——————————————————
'—— end of 683PWMLED.bas ——
```

FIGURE 6. PICBASIC PRO listing.

up with some improvements. The original program saved the speed or brightness setting in EEPROM so that when I turned the fan or light back on, it would start at the same level. The problem with fans is that you can slow them down to a 10% level when they are running, but you can't start them at that level. I added a feature that would boost the initial fan speed if it was below a certain level to get it started, and then drop back down to the slower saved speed. This resulted in two versions of the program: one for lights that did not have the "boost" feature; and one for

Heartbeat LED

When I first started working with microcontrollers, I would frequently chase software bugs that were in reality hardware issues. Now when I prototype, I ALWAYS include a heartbeat LED on the breadboard. If I'm working with a development board — which usually has one or more LEDs — I make one of them the heartbeat. I never take it out until I'm finished. Blinking an LED at the beginning of an embedded program (and throughout) verifies that your system is alive. The code is very simple; just a few lines to turn the LED on and off. Get this working first and then later on if the LED is not blinking, there is probably something wrong other than your code. Once I implemented the heartbeat LED consistently, I've saved myself considerable aggravation. Where initially I suspected my code, I discovered that batteries had depleted, programming or prototyping wires had come loose, a critical component had been harvested from a breadboard for use elsewhere, and so on. These are simple, silly things that are easily fixed, but if overlooked can make you doubt your sanity. If the LED does not blink on power-up, I check the hardware and environment before I mistrust my code.

Using a heartbeat LED and writing the universal microcontroller equivalent of "Hello world" is also useful when starting with new hardware. It verifies that you have a viable configuration. It can also provide feedback on the correct oscillator or crystal settings.

Once I have the heartbeat, I add the serial output routines for more detailed diagnostics. Then it's on to the rest of the project. Once you have the heartbeat and serial routines under control, you have a good framework for proceeding.

Terminal Programs

For debugging, I connect my PIC projects directly to a terminal program running on my development PC. I have not had any trouble connecting an output pin from my microcontrollers directly to a PC serial port, even at 3.3 volts. I just have a ground wire to pin 5 and another from the serial output pin to pin 2 on the DB-9 connector. With the disappearance of physical serial ports on PCs, I'm usually connecting to a serial-to-USB cable, such as shown in **Figure 5**.

Microsoft Vista and Windows 7 do not include a serial terminal program. If you are still using Windows XP — as I am for my development systems — then you can use Hyperterm. I find it a nuisance to work with and have found some free alternatives that work well on all systems. Aside from the simplicity of use, some also support TCP and UDP which makes them handy for testing TCP/IP communications programs. My favorites are:

The Hercules Setup utility from HW Group (**www.hw-group.com/products/hercules/index_en.html**).

TeraTerm Pro is available from **www.ayera.com/teraterm/**.

NetBurner has utilities for monitoring and debugging communications. These are available from **www.netburner.com/support/public_downloads.html**. The serial terminal program is mtty.exe.

None of these programs require installation — they run straight from the exe so they are easy to carry around and use. Some of them (Hercules and TeraTerm) do not list the available COM ports; that is, you have to know ahead of time which COM port you want to use.

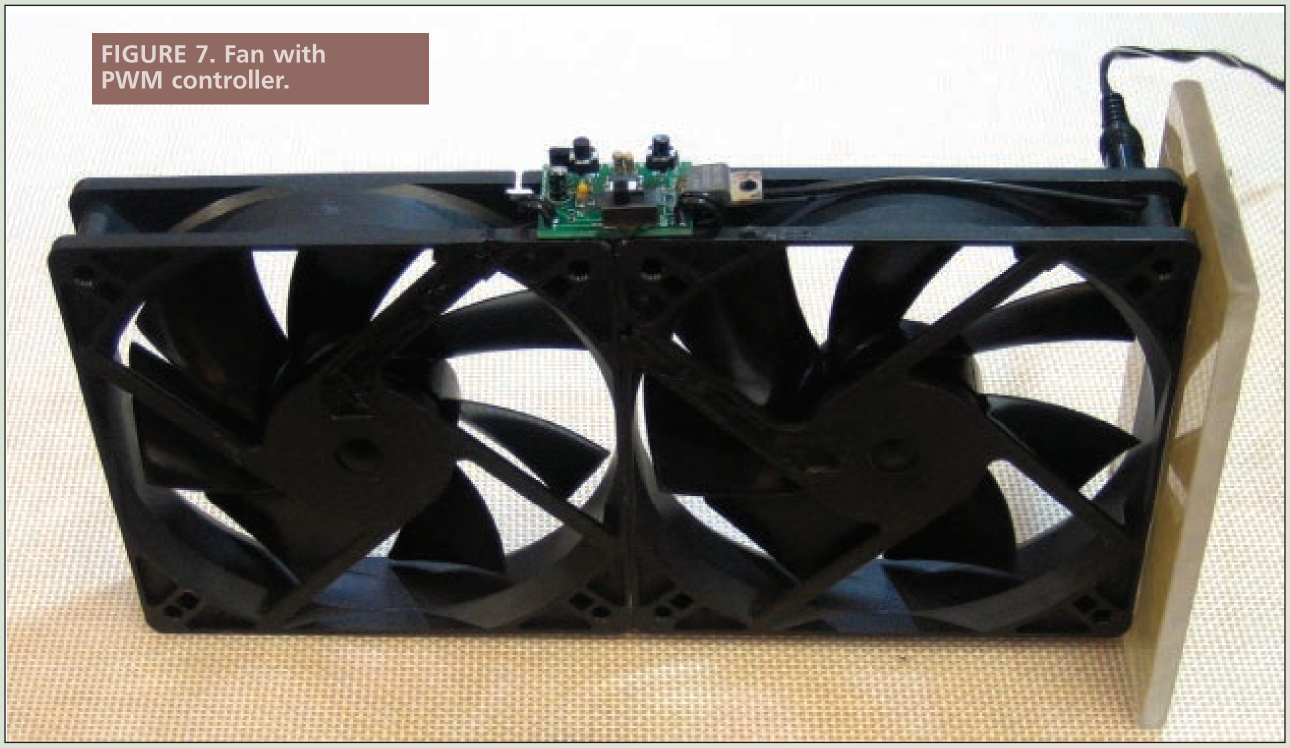


FIGURE 7. Fan with PWM controller.

fans with the boost feature.

The latest version combines both features into a single device. There is a setup routine that sets the controller to fan or light mode. If you hold down the "increase" button during power-up, the controller is put into light mode. If you hold down the "decrease" button while turning on the power, the controller is put in fan mode. This setting is saved in EEPROM. Since I use these PWM controllers as parts of other projects, I only need one version now that I can easily switch back and forth, depending on what I am working on.

With the appropriate hardware, this code can be adapted to a variety of uses. When driving a robot, for example, you can gradually ramp up the duty value in a loop so that the wheels don't spin from the sudden full-on activation. If you use it to control lights, you could gradually turn them on and off.

When you look at the C code, you'll see statements bracketed by **#ifdef DEBUG** and **#endif** statements. The serial output and heartbeat LED statements are enclosed in these so they can be disabled easily in the release version.

Finished

Once I knew the circuit would work, I sent the Gerber files to Silver Circuits for manufacturing the boards. I refined the software while I waited for the boards to come back. The board has a position labeled J1 which is where the incoming power is connected if you are not including a switch in the design. If you are including the switch, external connector J2 is connected to the pads marked "IN." Power to the load — via external connector J3 — is connected to the pads labeled "OUT." The square pads are positive. Please note the circuit was primarily designed for controlling low voltage lights and brushless computer fans. There is not any built-in protection against transients or surges coming in from the load.

Figure 7 shows the controller attached to the side of a pair of fans. I can now easily adjust the speed on these from "Hurricane" to "Mild Breeze." **Figure 8** shows the controller integrated into my under-the-shelf LED light bar. This photo only shows the first three of the eight LEDs that are

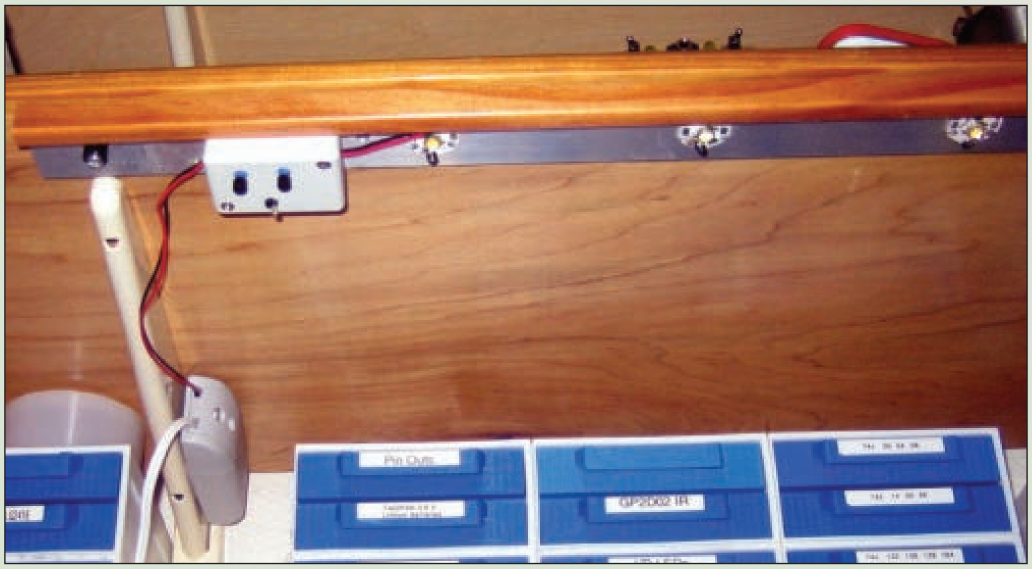


Figure 8. Workbench light with PWM dimmer.

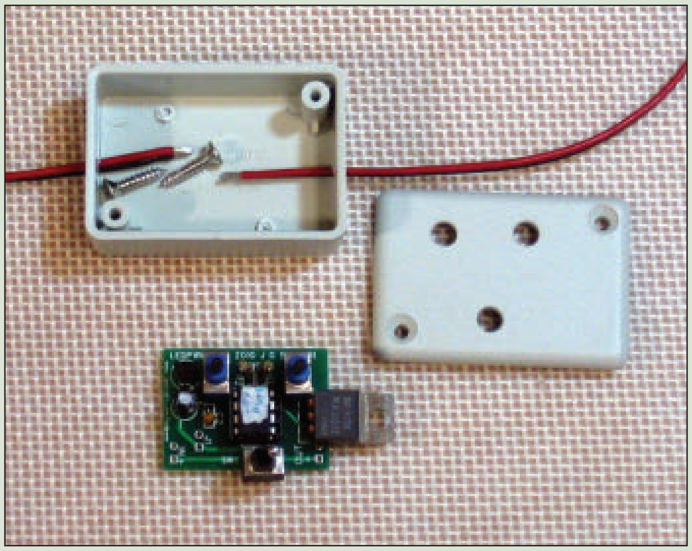


Figure 9. Light controller before assembly.

mounted on a strip of aluminum with heat-conducting epoxy. This strip is, in turn, attached with spacers to the underside of the shelf so air can circulate on all sides of it. High power LEDs require heatsinks to dissipate the heat they produce. You can see the completed circuit board and drilled box before final assembly in **Figure 9.** If you compare it with **Figure 1** and **Figure 7**, you can see that the PCB supports switches in different configurations.

Once you have worked with this basic PWM circuit, you will be ready to implement PWM in larger projects that will include such things as driving robot wheels. Unlike the brushless fan motors I have been using here, drive motors will usually require additional electronics to address direction of rotation and the reverse EMF that is generated when they get turned on and off. A PWM signal will generally be used to control the speed of the motor. **SV**

Parts List

Designator	**Component**	**Source/Part Number**
PCB	N/A	Silver Circuits, N/A
U1	PIC12F683	Mouser.com, 579-PIC12F683-I/P
U2	78L33	Mouser, 511-L78L33ACZ
Q1	MJE3055T	Mouser, 511-MJE3055T
PB1, PB2	Pushbutton switch	All Electronics.com, PB-157
R1,R2	10K 1/8 watt	
C1	1 µF 50V	
C2	.1 µF 50V	
J2	2.1 mm jack (power in)	All Electronics, DCJ-1
J3	2.1 mm plug (power out)	All Electronics, DCSID
SW1	Any suitable slide or toggle switch (optional)	

The spreadsheet, hex, and source code files are available at **www.servomagazine .com** or **www.jgscraft.com/ledpwm**. PCBs and pre-programmed chips for this project are also available.

Jürgen Schmidt can be contacted at jurgen@jgscraft.com.

References

Fairchild LM555 Datasheet
www.fairchildsemi.com/ds/ LM/LM555.pdf

PIC12F683 Datasheet
www1.microchip.com/downloads/ en/DeviceDoc/41211D_.pdf

Microchip Application Note AN594: Using the CCP Module(s)
www1.microchip.com/downloads/ en/AppNotes/00594B.pdf

Microchip Application Note AN564: Using the PWM
www1.microchip.com/downloads/ en/AppNotes/00564b.pdf

PICBASIC PRO Compiler
www.melabs.com

CCS C compiler
www.ccsinfo.com

PCB Manufacturing
www.silvercircuits.com

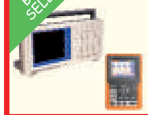
UNIQUE PRODUCTS with free lifetime support!

New Advanced Triggering

Pico - 2ch 250MHz Scope with 32 or 128 MS and single-shot sampling of 1GS/s. Adv triggering. 250MHz spec. analyzer and 125MS/s waveform gen.
PS5203 / PS5204 $1830 / $2736

BEST SELLER

Bargain LCD Scopes

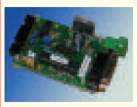


Owon - 25 or 60MHz color LCD 2ch bench scope **OR** 20MHz handheld scope/ meter. USB interface for printouts.
PDS5022S / PDS6062T ~~$315 / $499~~
HDS1022MN / HDS2062MN $593 / $699

7 in 1 USB Scope

2-ch 10-bit 2MHz scope/spectrum-analyzer, 3MHz 8-bit wfm gen; 16 x I/O; Network Analyzer; Noise Generator; PWM. Windows/Linux/ Mac compatible! USB-powered, great for toolbox, education, etc. **CGR-101 $199**

Motion Control



Motion control boards with integrated power amplifiers for DC-servo/brushless-servo/stepper-motors. Motion Controllers can even be operated as a stand-alone control system.
PIC-SERVO SC $160

Scope / Analyzers

50/100MHz 2-ch/16 logic scope and logic analyzers. 2000 wfm/s refresh rate. See wfm and logic data simultaneously. Trigger on data or signal edge.
DS1052D / DS1102D $1195 / $1480

EMC Spectrum Analyzers

EMC RF & EMF Spectrum - Analyzers 1Hz to 7GHz for measuring transmissions from radar, radio/tv towers, WLAN, WiFi, WiMAX, Bluetooth, microwave ovens, mikes, etc
from $299 / $1990

Mixed-Signal Scope

CleverScope - 100MHz Scope, Spectrum/ Logic Analyzer, & SigGen. for PCs. 4 or 8M/sample storage! Easy A-B, math, & filters! 2 ch x 10, 12 or 14-bit, 8 dig. I/P. (Opt. 0-10 MHz SigGen. **$299**)
CS328A-4 (4MS Buffer) $1259
CS328A-8 (8MS Buffer) $1474

USB Bus Analyzers

Packet-Master™ - best value USB1.1/2.0 analyzers and generators. Identify USB problems fast, fine-tune performance.
USB12 (USB1.1) $699
USB480+ (USB1.1/2.0) $1199
USB500AG (USB1.1/2.0/Gen) $1399

Thousands More!

Ethernet-Serial Cable	$89
USB to 8 Thermocouple Logger	$466
USB to I2C I/O	$89
Keyboard Simulator board	$81
uCAM-RS232 & TTL	$58
USB-Serial adapters	$19
USB Logic Analyzer	$299

FREE STARBUCKS CARD WITH $50 ORDER

New products arriving daily!
See our website for specials!

Saelig .com
unique electronics
888-7SAELIG info@saelig.com