Building (H-)Bridges — Part 2 —

by Peter Best

Let's add some intelligence to the H-Bridge hardware I introduced to you last time. For those of you that are just joining us, the H-Bridge hardware that was previously presented in SERVO is shown in Photo 1.

To recap for those of you that missed the introductory *SERVO* H-Bridge column, we took some off-the-shelf Microchip MOSFET drivers and fashioned them along with some glue logic to act as driver circuitry for a quad of dual-MOSFET devices. After all of the wire slinging was said and done, we ended up with a pair of H-Bridges suitable for driving small brushed and stepper motors.

When we're finished discussing the schematics, code, and photos in this month's *SERVO* offering, you'll know how to apply the services of Microchip's newest motor control oriented PIC — the PIC16HV616 — to the H-Bridge circuitry you see in Photo 1. With that, let's begin the H-Bridge driver design cycle.

The PIC16HV616

The 14-pin PDIP version of the PIC16HV616 can be seen dominating the intelligent H-Bridge driver componentry shown in Photo 2. The PIC16HV616 speaks standard PICese using only 35 assembler instructions. A precision internal oscillator provides either a 4 MHz or 8 MHz system clock for the PIC16HV616's integral subsystems. A standard crystal or ceramic oscillator can also be attached to the PIC16HV616 to provide higher or lower clock rates.

The PIC16HV616 is actually a derivative of the PIC16F616. The PIC16F616 operates over a voltage range of 2.0 V to 5.5 V. Although not stated directly in any of the PIC16HV616 datasheets, the "HV" more than likely stands for High Voltage and this is what makes the PIC16HV616 different.

The PIC16HV616's operating voltage range spans from 2.0 V to a user defined maximum. This is accomplished by applying the services of a shunt voltage regulator that is built into the PIC16HV616. The high voltage capability of the PIC16HV616 allows this PIC to be thrown into motor drive circuitry without the need for an extra voltage regulator for the PIC. Otherwise, the PIC16HV616 and PIC16F616 are logically identical.

The PIC16HV616 embodies all of the standard bells and whistles you normally see in any typical PIC. The PIC16HV616 can sleep on command, reset on brown-out conditions, and protect the code embedded in its program memory. The PIC16HV616 is also capable of high sink and source currents on most of its I/O pins. Two analog comparators with built-in, user selectable hysteresis are accompanied by a unique on-chip SR (Set Reset) latch and a programmable on-chip voltage reference.

There's also an eight-channel 10-bit analog-to-digital converter in the PIC16HV616 mix. A trio of PIC16HV616 timers — Timer0, Timer1, and Timer2 — provide a pair of eight-bit timers in Timer0 and Timer2, while Timer1 (which can also be gated with the T1G input) extends to 16 bits of resolution. Each PIC16HV616 timer can be prescaled, with Timer2 having the ability to be both prescaled and postscaled. Motor control usually implies PWM (Pulse Width Modulation). The PIC16HV616's 10-bit PWM subsystem can be configured with one, two, or four

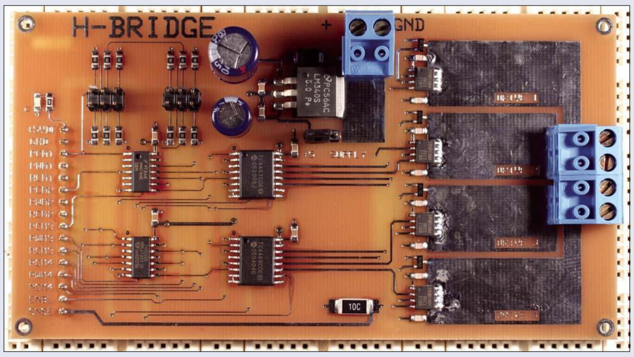


PHOTO 1. If you wish to build up your own version of the H-Bridge shown here, you can get the H-Bridge ExpressPCB layout file from the *SERVO* website (**www.servomagazine.com**). All of the H-Bridge components are off-the-shelf and can be purchased from many of the distributors that advertise in this magazine.

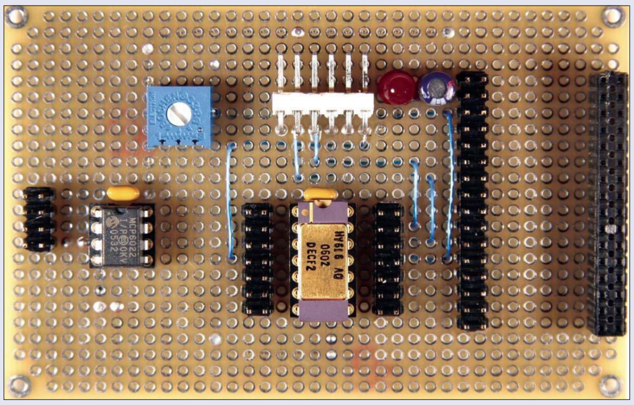


PHOTO 2. The only items that are hard-wired include the ICSP interface, the 10K pot, and the MCP6022 op-amp. Everything that attaches between the PIC16HV616 and the H-Bridge printed circuit board is open game.

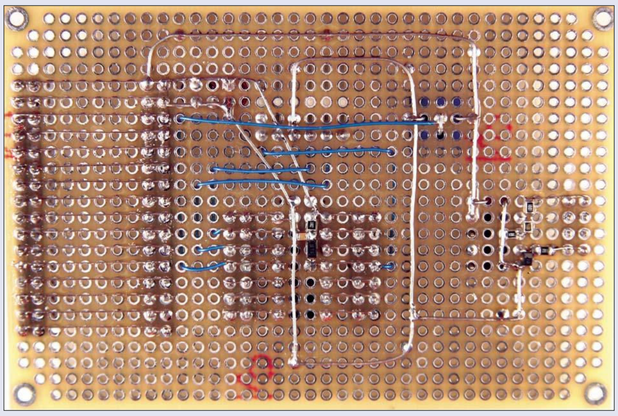


PHOTO 3. This is a shot of my wire art that makes all of the necessary connections between the PIC16HV616, the H-Bridge printed circuit board, and the H-Bridge driver's supporting components. Note my use of SMT passive components to save space and make things neater on the component side of the board.

output channels.

The PIC16HV616's ECCP (Enhanced Capture Compare PWM) module can be programmed to operate in an enhanced mode. The Enhanced PWM Mode can be used to generate a PWM signal on up to four different output pins with up to 10-bits of resolution. This is accomplished using one of four modes: Single PWM, Half-Bridge PWM, Full-Bridge PWM Forward, and Full-Bridge PWM Reverse. We'll use the Full-Bridge modes to drive a brushed DC motor in this installment.

Since the PIC16HV616 is aimed at motor control applications, the program Flash and SRAM complements are small with the PIC16HV616 supporting 2K of program Flash and 128 bytes of SRAM. Although the PIC16HV616 memory numbers may appear to be tiny, remember that many amazing things were done with the original PIC16C5X devices, which had equal or lesser amounts of memory space. In fact, I was told by a Microchip representative that the PIC16C5X parts are still one of Microchip's biggest sellers.

Wire Art

Now that you have the low-down on the PIC16HV616, let's put the part to work. Since there are multitudes of ways to connect to and drive the H-Bridge, the PIC16HV616 H-Bridge driver design must be flexible enough to provide any-to-any connection between the H-Bridge printed circuit board (PCB) and the PIC16HV616 driver board. Take a look at the H-Bridge PCB in Photo 1. There are 16 entry and exit points, which present power, ground, MOSFET driver inputs, an enable input signal, and a sense voltage output for external control and monitoring.

Now take another look at Photo 2. To keep things simple and totally flexible, the H-Bridge driver circuitry is built on a custom high quality perf board, which consists of a .1-inch center set of plated through holes. The use of a perf board implies that instead of a custom PCB, we'll use wire art to fashion our H-Bridge driver circuit.

The H-Bridge's 16-pin interface is populated with a 16-pin single-row header. The H-Bridge driver mates to the H-Bridge 16-pin header with a 32-pin double-row header socket. The header socket could also be a single-row socket but the double row configuration makes it a bit easier to orient vertically and makes for a sturdier mounting point.

Note that the PIC16HV616 is surrounded by a pair of double-row male headers in Photo 2. Also notice that a 32-pin double-row male header parallels the 32-pin double-row female header socket. The method to this header madness is revealed in Photo 3.

Forget about double-row as a physical attribute of the headers as each 32-pin double-row header is effectively a pair of 16-pin single row headers tied together electrically. Another look at Photo 3 shows that each of the row pins on every double-row header is connected to its respective row neighbor via a wire jumper. The 32-pin header and socket on the PIC16HV616 driver board are wired in parallel using wire jumpers forming four columns of 16 pin positions. Each of the 16 rows between the female and male 32-pin headers is connected electrically by a wire jumper.

The same holds true for double-row male headers that surround the PIC16HV616. A wire jumper extends from each PIC16HV616 pin across the pair of adjacent header pins. The double-row pinout arrangement allows us to use temporary wire jumpers with female terminations to connect any point of the H-Bridge 16-pin interface to any pin of the PIC16HV616. In addition, the extra pin in each row is exposed and can be used as an additional jumper point or a test point for monitoring logic levels or voltages.

The MCP6022 op-amp is used to amplify the voltage at the H-Bridge's sense output pin. Only the op-amp's input and output pins are accessible via a header pin. All of the passive components associated with the MCP6022 circuit are SMT packages and are mounted under the MCP6022 socket on the wire side of the H-Bridge driver perf board.

I hardwired the 10K pot to the PIC16HV616's AN2 analog-to-digital converter input pin. Since this is all on perf board, you may wish to bring the pot's pins out to header connections for more connection flexibility.

The right-angle six-pin male header shown in Photo 2 is the ICSP programming interface. The PIC16HV616's MCLR reset circuitry is also made up of SMT devices. You can see the reset resistor/capacitor combination and the Vpp blocking diode nestled beneath the PIC16HV616 in Photo 3.

Supply voltage for the PIC16HV616 and the MCP6022 is provided by the H-Bridge PCB regulated power supply via the 16-pin header interface. It is a good engineering practice to include a filter capacitor at the power junction when passing supply voltages between boards using connectors. So, I placed a 10 µF

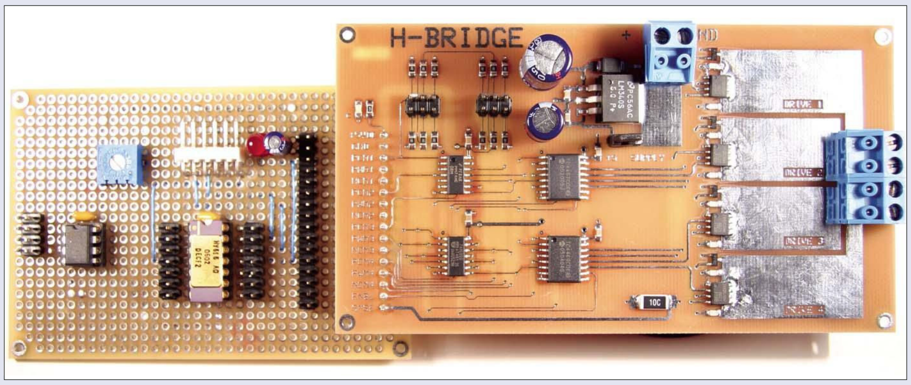


PHOTO 4. Here's a shot of the H-Bridge and the H-Bridge driver boards mated at the interface point. Wire jumpers with female terminations or wirewrap connections can be used to make connections between the PIC16HV616 and the H-Bridge MOSFET drivers.

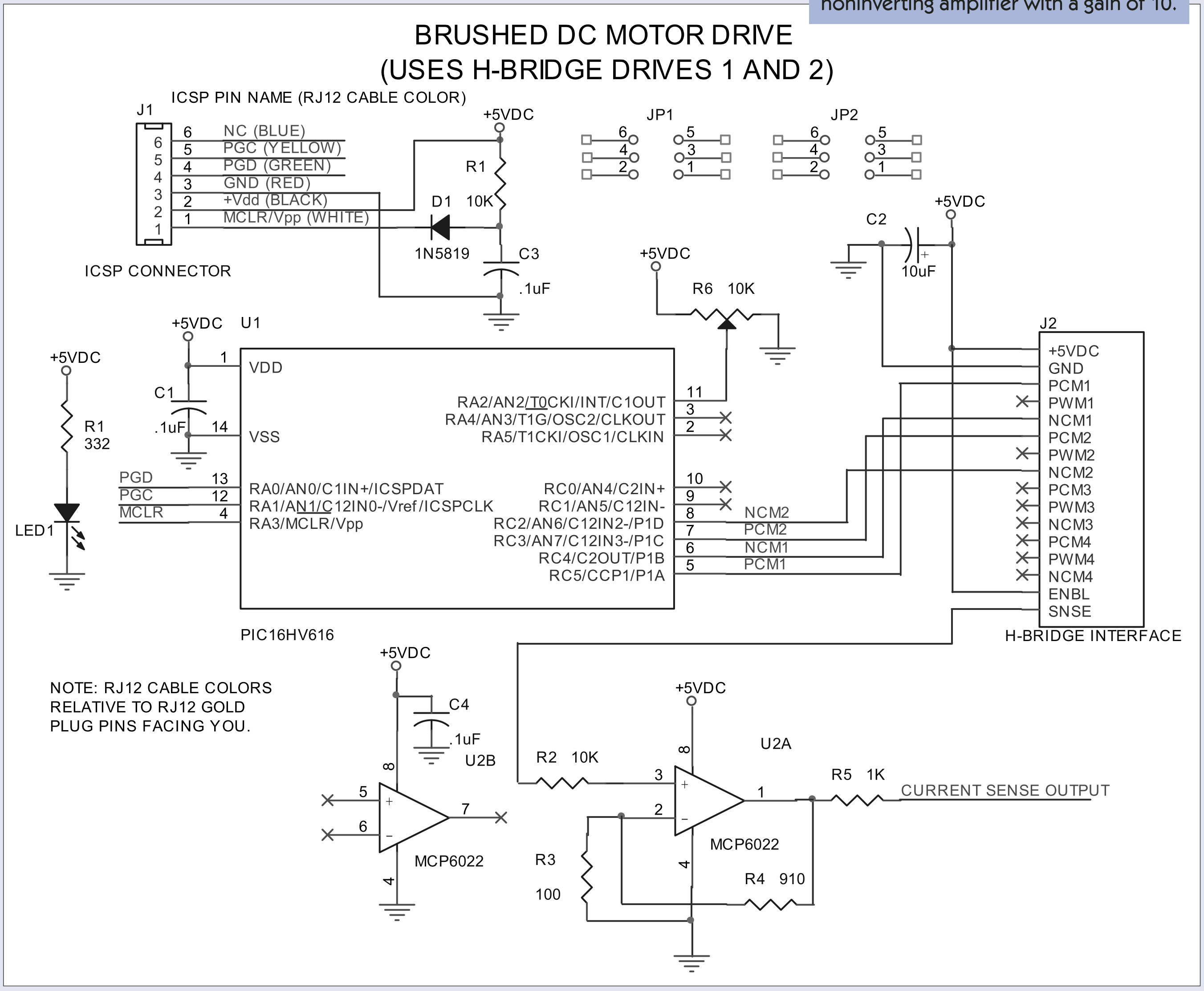
capacitor across the DC supply pins on the H-Bridge driver board.

Just to make sure power was indeed present on the H-Bridge driver pins, I added an LED indicator. Photo 4 is a composite shot of the H-Bridge and H-Bridge driver board mated at their respective 16-position interfaces.

Driving a Brushed DC Motor

With the addition of the PIC16HV616-based H-Bridge driver, the H-Bridge hardware build for our application is complete. Schematic 1 is a graphical depiction of the configuration we will use to attach and drive a brushed DC motor. Note that the H-Bridge segments are not coupled in full-bridge configuration as there are no jumpers on the JP1 and JP2 pins. Only one pair of the H-Bridge's quad of half-bridge drives is needed to drive the brushed motor. The actual motor leads are

SCHEMATIC 1. A five-jumper hookup is all that's needed to drive a brushed DC motor. The MCP6022 is configured as a noninverting amplifier with a gain of 10.



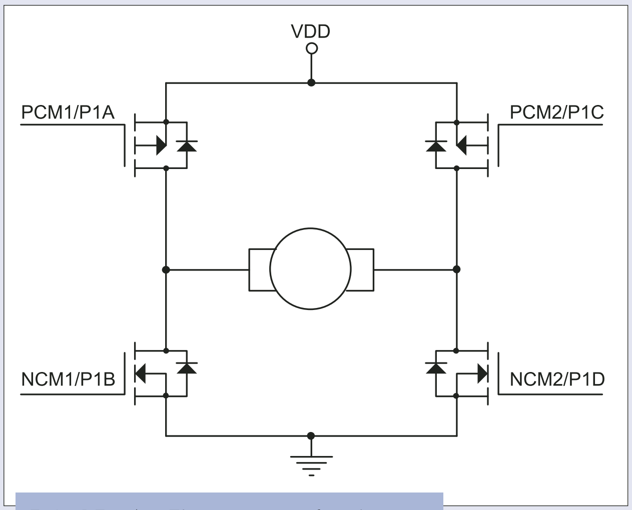


FIGURE 1. The motor leads are attached to each half-bridge segment of the H-Bridge in this manner. Basically, PCM1 and NCM2 are on to drive the motor in one direction and PCM2 and NCM1 work together to drive the motor in the opposite direction. You can get more details by examining the original H-Bridge schematics, which I've provided as part of this month's download package.

connected to half-bridge Drives 1 and 2 of the H-Bridge. A much simplified half-bridge connection diagram is shown in Figure 1.

Only five jumper connections (ENBL, PCM1 to P1A, PCM2 to P1C, NCM1 to P1B, and NCM2 to P1D) are necessary to configure the H-Bridge to operate in ECCP Full-Bridge mode. An optional sixth connection from the H-Bridge SNSE (current sense) output to the MCP6022 op-amp's input can be attached if you want to monitor the motor's current consumption.

The voltage output at the MCP6022's output is equal to the motor's current consumption. For instance, if the voltage at the output of the op-amp is one volt, the current consumption is one amp. That's in a perfect world as you must consider even with 1% components all around the op-amp, there will be some small percentage of deviation in the voltage measurement.

The fifth jumper connection involves activating the H-Bridge's ENBL (Enable) input pin. The H-Bridge ENBL pin must be logically high to allow the MOSFETs to be driven. The ENBL line can be simply tied high or tied to a PIC16HV616 pin for firmware control. For the sake of simplicity, I chose to tie the ENBL line high with a jumper from the ENBL pin to the +5V pin on the H-Bridge interface.

Once the PCMx and NCMx jumpers are in place, the H-Bridge driver code is used to invoke the ECCP Full-Bridge mode. You may wish to consult Figure 1 again as I describe the ECCP Full-Bridge Forward and Reverse modes. In ECCP Full-Bridge Forward mode, the PIC16HV616's P1A pin is driven active and P1D is modulated via PWM. The active state of the PIC16HV616's ECCP I/O pins is determined in firmware.

In this application, the active state programmed as a logical high. The PIC16HV616's P1B and P1C pins are held inactive in ECCP Full-Bridge Forward mode. To drive the motor in the reverse direction, the ECCP Full-Bridge Reverse mode is used. In ECCP Full-Bridge Reverse mode, the PIC16HV616's P1C

Listing 1

```
void main()
{
        TRISC = 0b11000011;        //P1A-P1D outputs
        ANSEL = 0b00000100;        //select AN2
        ADCON1 = 0b01010000;       //ADC conversion clock Fosc/16
        PR2 = 0x3F;                //set PWM period 320uS@8MHz or PWM freq of 31.25KHz
        OPTION = 0b00000111;       //set TMR0 prescaler 1:256

        ADCON0 = 0b00001001;       //enable ADC
        CM1CON0 = 0x07;            //disable comparators

        TMR2ON = 1;                //turn on PWM
        master timer = 0;          //initialize clock
        secs = 0;
        mins = 0;
        hours = 0;
        T0IE = 1;                  //enable TMR0 interrupt
        PEIE = 1;                  //enable peripheral interrupts
        GIE = 1;                   //enable global interrupt

        PORTC = 0x00;

        while(1)
        {
                timer = 0;
                GODONE = 1;
                motor(rev);
                while(timer == 0);

                timer = 0;
                GODONE = 1;
                motor(fwd);
                while(timer == 0);

        }
}
```

LISTING 1. What you don't see here are the motor function and the clock interrupt service routine. Don't worry. The complete volume of source code is included with the H-Bridge driver download package.

pin is driven active and the PWM modulation is provided by the P1B pin. ECCP pins P1A and P1D are held inactive in ECCP Full-Bridge Reverse mode.

The code needed to drive a simple brushed motor is rather simple and is shown partially in Listing 1. The brushed motor driver firmware configures the PIC16HV616's P1A, P1B, P1C, and P1D pins as outputs, which is required if ECCP mode is to be used. The 10K pot — whose wiper is attached directly to the PIC16HV616's AN2 analog-to-digital converter input pin — will be used in this application to control the speed of the motor. The PIC16HV616's analog-to-digital converter reads the voltage at the 10K pot's wiper and the measured voltage value is then loaded in as the PWM duty cycle value via the CCP1CON and CCPR1L registers.

The forward and reverse motion of the brushed motor is determined by bits within the CCP1CON register. The PIC16HV616's internal clock is set for 8 MHz operation. I've also implemented a 32.768 ms-per-tick clock by prescaling Timer0 with a 1:256 ratio. The Timer0 clock is used strictly for delay timing in this application. The brushed motor direction delay is set for 30 32.768 ms ticks, which is approximately one second.

Thus, here's how the brushed motor firmware flows:

• The 32.768 ms tick timer is reset to zero.

• A duty cycle analog-to-digital converter conversion is started.

• The motor function is called with a fwd argument.

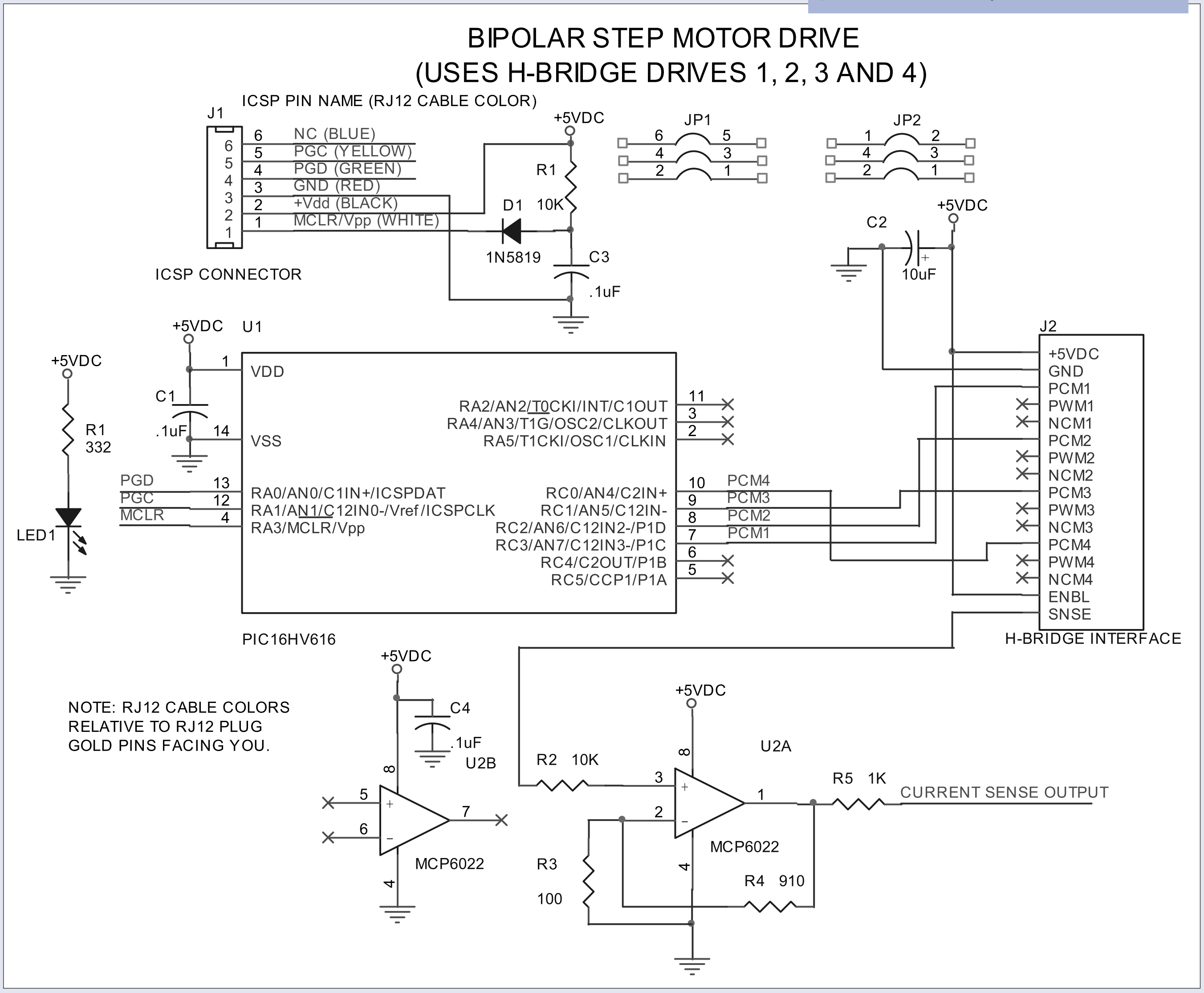
• Duty cycle and direction information is loaded.

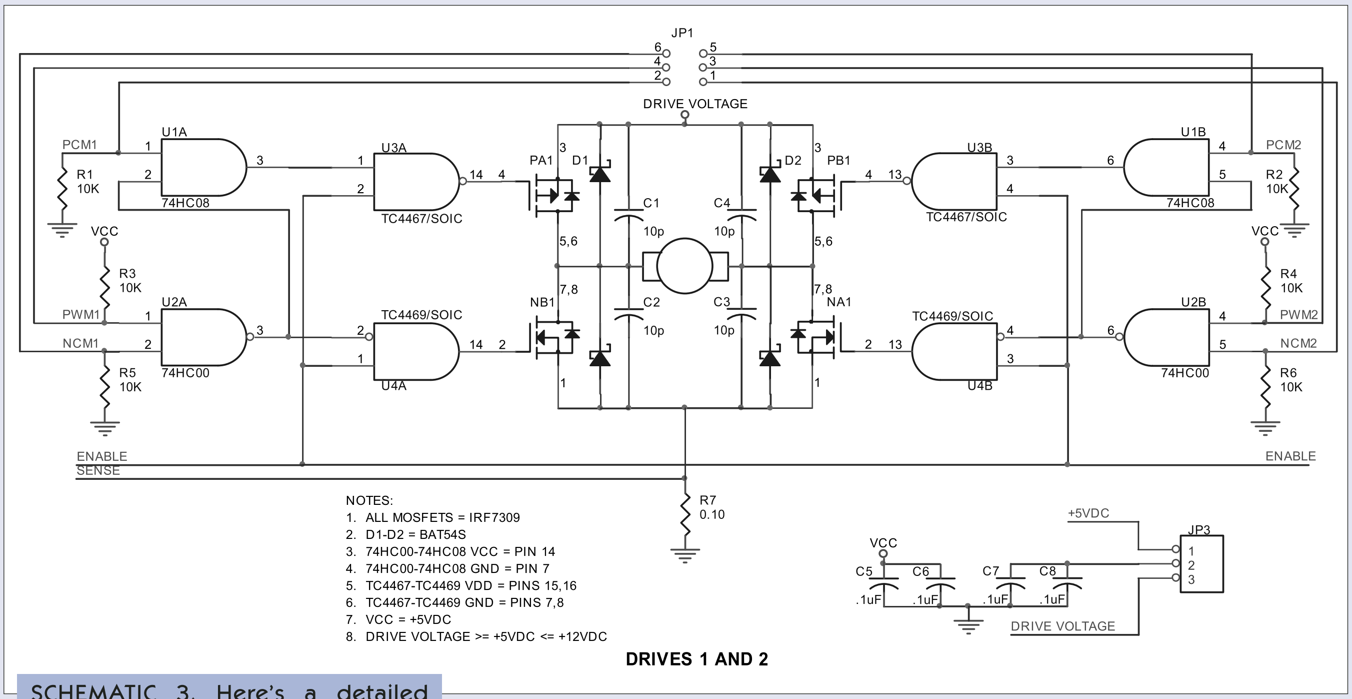
• P1A is driven active and P1D emits PWM at measured duty cycle.

• Motor shaft turns in a forward direction for one second.

• The 32.768 ms tick timer is reset

SCHEMATIC 2. This is a single-step stepper motor implementation. Note that there is no PWM involved in the stepping motion. PCM1 through PMC4 simply walk through the bipolar step pattern continually.





SCHEMATIC 3. Here's a detailed look at a pair of H-Bridge drivers. Placing jumpers on JP1 makes things interesting.

to zero.

• A duty cycle analog-to-digital converter conversion is started.

• The motor function is called with a rev argument.

• Duty cycle and direction information is loaded.

• P1C is driven active and P1B emits PWM at measured duty cycle.

• Motor shaft turns in a reverse direction for one second.

• This entire cycle repeats from the beginning.

That's all there is to driving a brushed DC motor. Now, let's turn our configuration and coding efforts towards driving a simple bipolar stepper motor.

Driving a Stepper Motor

Before we can attach a bipolar stepper motor to the H-Bridge Drives, we must do some physical reconfiguring. Note that in Schematic 2, JP1 and JP2 are populated with jumpers. The presence of these jumpers combines the quad of half-bridges into a pair of full-bridges. Let's use Schematic 3 to step through what happens when the JP1 jumpers are in place. Keep in mind as we're walking

Listing 2

```
void main(void)
{
        TRISC = 0b00000000;             // PORTC = Output
        OPTION = 0b00000111;            // TMR0 Prescaled 1:256
        master timer = 0;               // Clear RTC regs
        secs = 0;
        mins = 0;
        hours = 0;
        T0IE = 1;                       // Enable TMR0 Interrupt
        PEIE = 1;                       // Enable Peripheral Interrupts
        GIE = 1;                        // Enable Global Interrupt

        while(1)                        // Loop Forever
        {
                timer = 0;              // Clear timer count
                PORTC = PCM1;           // Energize winding
                while(timer == 0);      // Wait for timer to expire
                timer = 0;
                PORTC = PCM4;
                while(timer == 0);
                timer = 0;
                PORTC = PCM2;
                while(timer == 0);
                timer = 0;
                PORTC = PCM3;
                while(timer == 0);
        }
}
```

LISTING 2. The H-Bridge circuitry takes the complexity out of this code. All we have to do to move the stepper motor shaft is ollow the coil activation/deactivation pattern for a bipolar stepper motor. Operation of bipolar stepper motors is described very well in Microchip's AN907 app note.

through this process that the half-bridges coupled by JP2 behave exactly the same way.

Let's begin by placing an imaginary jumper across JP1 pins 1 and 2 only. When a logical high is presented to the PCM1 pin, the imaginary jumper we just installed on Schematic 3 routes the high-going signal applied to PCM1 over to NCM2, which just happens to be the PA1's electrically complementary MOSFET. NA1 activates and current flows through PA1, the motor winding, and NA1. Now let's add another imaginary jumper across JP1 pins 3 and 4.

When a logical high PWM signal is applied to PWM1, that same signal is applied to PWM2 by way of our newly installed imaginary jumper at JP1 pins 3 and 4. The logical high PWM signal we just applied has no effect upon the output of NAND gate U2A as its input levels did not change. PA1 is still active at this point. At NAND gate U2B, there is also no change in output as PWM2's and NCM2's inputs did not change state either.

When the PWM signal swings to a low logical level at PWM1, it also swings low at PWM2. U2A's output level does not change as both of its inputs are low, resulting in a high output, which keeps PA1 energized. The resulting low-going PWM pulse results in a pair of low-level inputs at NAND gate U2B. U2B's output shifts from logically low to logically high. U4B's inverting input sees the U2B high output as a low input and turns off NA1. PCM2 and NCM1 are kept out of the picture by their pulled down inputs.

When we install that last imaginary jumper across JP1 pins 5 and 6, PB1, NB1, PCM2, PWM2, and NCM1 are drawn into the mix and respond to logic level stimulus exactly like their counterparts PA1, NA1, PCM1, PWM1, and NCM2.

Okay, while we're in imaginary mode, replace the motor in Schematic 3 with a coil from a bipolar stepper motor. Then create another circuit just like the one in Schematic 3 for the second coil of a bipolar stepper motor. What you end up with is an H-Bridge for each bipolar stepper motor coil, or our H-Bridge hardware.

The code in Listing 2 energizes the bipolar motor coils in a pattern that flips the rotor of your stepper motor round and round. To reverse the direction of rotation, simply reverse the order of the writes to PORTC in Listing 2. Speed up the rotation by shortening the delays between steps. Conversely, slow down the rotation by increasing the delays between steps. I used the same 32.768 mS tick timer code from the brushed motor application in the stepper code to make delay generation easy.

If you need to know more about how bipolar stepper motors work, I suggest getting a copy of Microchip's AN907, *Stepping Motor Fundamentals*. There you will find a bipolar stepper motor truth table that you can directly correlate to the stepper motor code and schematics I've provided for you.

Things to Play With

Our H-Bridge and motor discussion is complete. You now have everything hardware and firmware you need to spin small- to medium-sized stepper and brushed DC motors. Here are some things you can tinker with once you get your motors spinning. If you need over-current shutdown capability, you can use the current sense circuitry to feed an analog-to-digital converter input or comparator input on the PIC16HV616 and kill the Enable line when a preset current level is exceeded. Motor RPM can easily be obtained and controlled by monitoring the motor shaft rotation optically and counting the incoming pulses over a predefined time period.

The H-Bridge design used in this series was designed with a development board mentality. You can greatly decrease the physical footprint of the H-Bridge and H-Bridge driver circuits described within these pages to fit a medium-powered programmable motor controller into the tightest of spaces. **SV**

SOURCE

PIC16HV616
MCP6022
Microchip Technologies
www.microchip.com

Peter Best can be contacted via email at peterbest@cfl.rr.com

"THERE IS NO ROYAL ROAD TO DESIGN" .....EUCLID (c 300 bc)
"but, "THERE IS A CHEAP TOLL ROAD"" .....CLARK (c 2000 ad)
UML or BNF?
SEE_HERE: ; (THIS IS A COMMENT)
; Teach your kid that American Design needs to be much better than
; "Good enough for government work." Perhaps the result will be
; firms and engineers that can design a SUV, a LEVEE, or a Robot.
; BNF simple example
: Begin
IV-KITS ::= BREADBOARDS
& ¦ _ PARTS (& ¦ _ IS AND OR NOT)
& ¦ _ CHIPS
& ¦ _ CODE
& ¦ _ CONSTRAINTS
where BREADBOARDS ::= BB2; BB3; BB4; BB5; ...
-- LOW_COST AND SHORT_SCHEDULE
: end
ORDER: "Using BNF for Design" ....(120 pages softcover) $9.00 MO only, postpaid USA
USE_BNF as a design language for systems, hardware, software, firmware, planning, packaging, debugging, testing, documentation, analysis, synthesis, verification, and validation. It is a top_down and bottom_up definitional methodology. In general BNF is not logic but a consistent rational and irrational methodology for words, numbers, and events, as well as IDEAS, OBJECTS, ATTRIBUTES, and ELEMENTS.
IV, POB 245, WALDWICK, NJ, 07463-0245