Overview:

There may be instances where the 12A continuous current handling capability of the ICON H-Bridge is not adequate for a specific application. For higher current applications there are at least two options that can be implemented to allow the ICON H-Bridge to meet the needs of the design.

The first option is to increase the power dissipating capability of your system. This can most easily be accomplished by increasing the forced airflow over the ICON H-Bridge with a high-speed, high-volume, fan. But even with increased airflow the ICON H-Bridge will probably not be capable of continuously handling more than 20A of continuous current due to the size of the ICON H-Bridge PCB.

The second option is to operate multiple ICON H-Bridges in parallel. The N-channel MOSFETs that make up the H-bridges can efficiently share current, thus increasing the overall current handling capability of your system. In addition, the number of control lines required to interface to each of the MOSFETs is not increased since these control lines may also be paralleled. In this application note a single PIC16F873 (Microchip) microcontroller was used to control four ICON H-Bridge modules.

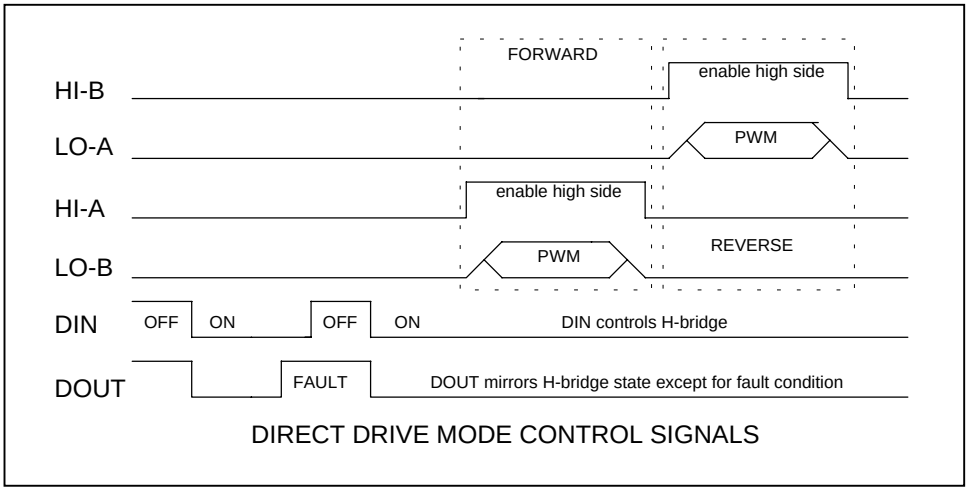
An analog-to-digital converter (ADC, internal to the PIC16F873) was used to establish a pulse-width-modulation duty-cycle (PWM, also internal to the PIC16F873) to control four ICON H-Bridge modules connected in parallel. The system was configured for unidirectional speed control and was tested at 24V with a resistive load of 0.389Ω. A custom PCB was designed to carry the four ICON H-Bridge modules and the control circuitry. In testing, this design was able to deliver 48A of continuous current to the load at a 98% duty cycle (roughly 900W) for several hours.

H-Bridge Configuration:

The ICON H-Bridge modules were configured to operate in “direct drive” mode for this application note. All other settings in the ICON H-Bridges were left in their default state (as described in ICON H-Bridge datasheet). A single ICON Interface Module was used to program the ICON H-Bridges for direct drive mode using the ICON Interface Module control software available at the www.solutions-cubed.com web site.

In direct drive mode the serial DIN line of each H-bridge becomes the /ENABLE line. Driving this line to 0V turns on the H-bridge, and enables the H-bridge control lines (HI-A, LO-B, HI-B, and LO-A) to actively control the MOSFETs on all four H-Bridges. Additionally, direct drive mode converts the DOUT line from a serial output to a STATUS line. When this line is at a logic high (+5V) the H-bridge is disabled. A logic low indicates that the H-bridge is enabled.

Figure 1: Direct Drive Mode Control Signals



It would be feasible to configure the PIC16F873 to test the ICON H-Bridges for direct drive mode of operation on power-up by toggling the DIN line and checking to see if the DOUT line follows the input. If the devices were not configured for direct drive mode they could be programmed to operate in direct drive mode via the DIN and DOUT connections of each H-bridge and the ICON H-Bridge communication protocol. This would eliminate the need for the ICON Interface Module used during the programming phase of this application note. There is also no reason that this application note could not be modified to implement the full serial communication protocol.

Hardware:

The custom PCB was designed to mount all four ICON H-Bridges in close proximity to each other. Mounting holes were included for the ICON Active Cooling fans. ¼" bolts were used as the connection points for both the load and the load supply, and were responsible for carrying the high currents required for this application note. As mentioned earlier the load supply was 24V and rated for 60A. The 12V supply required to power each ICON H-Bridge and the four cooling fans was provided by a separate 12V power supply. The 5V necessary to power the PIC16F873 was generated with a linear regulator connected to the 12V power supply.

Connector J2 (the 20 pin 0.1" shrouded header) was included and mirrors many of the connections of the ICON Interface Module. This allows the design to work with the ICON Adapter Board. The ICON Adapter Board was used to provide the analog input signal for PWM generation and TTL to RS232 conversion for the serial interface built into the application.

Figure 2: PIC16F873 and Support Circuitry

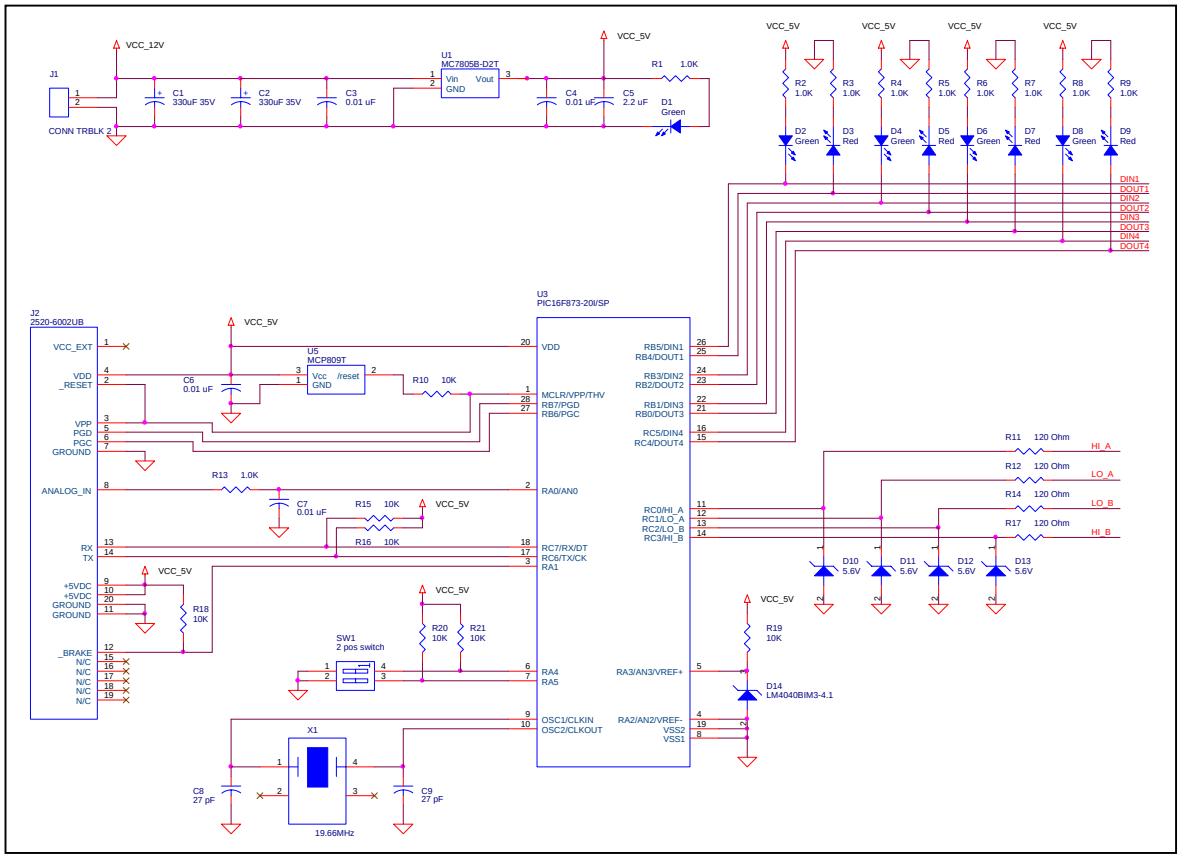
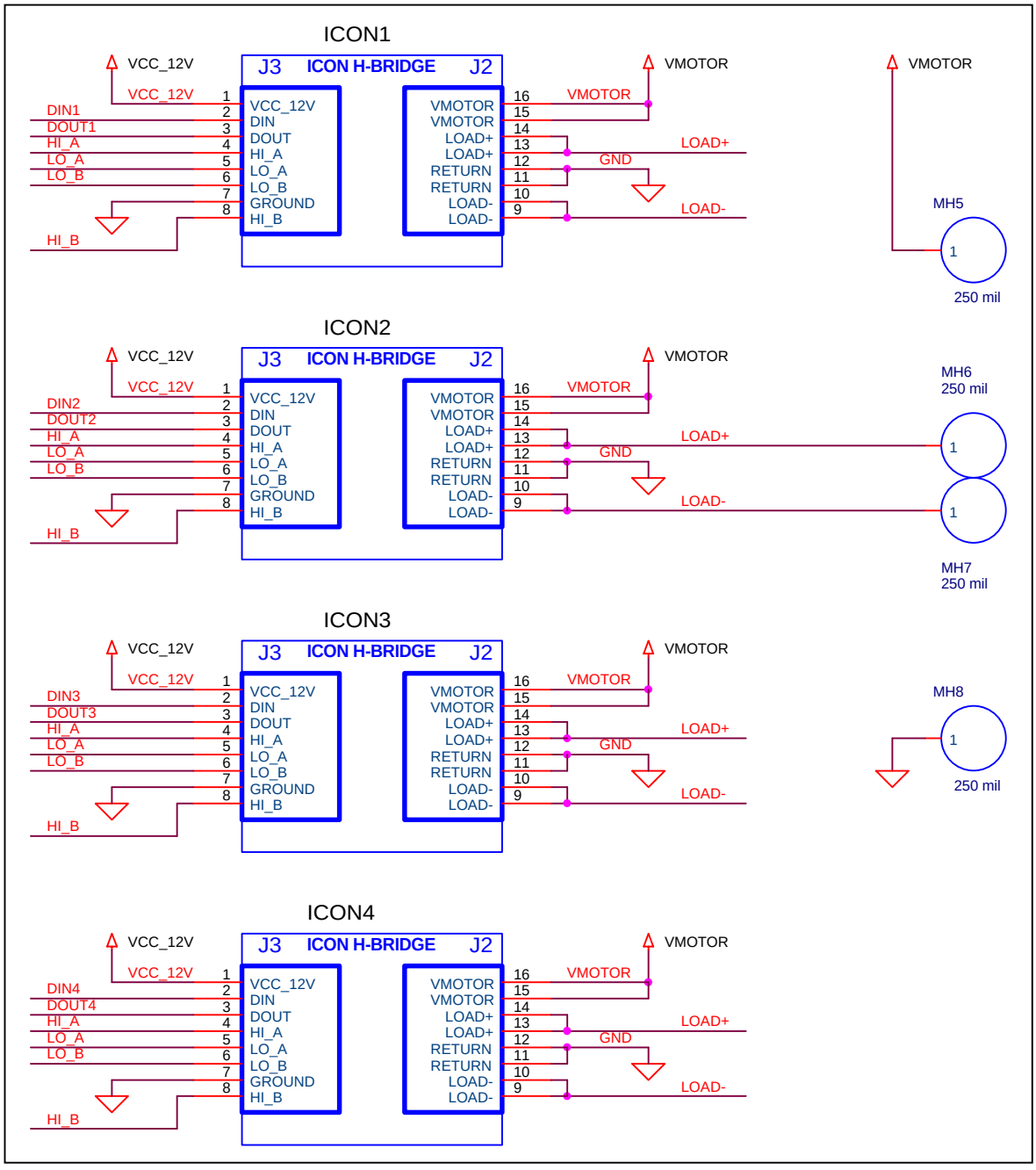


Figure 3: ICON H-Bridge Connections



The custom PCB used in this design was a 4" x 7.1" 2-layer board with components on the top-side only. The copper cladding was a standard ½ ounce thickness, and the high current carrying traces measured roughly 1" wide. This board was not designed to continuously carry a high load current, but was designed to allow for testing of the ICON H-Bridges for several hours at their maximum current rating (48A continuous).

Boards designed for continuous loads of this magnitude should use copper cladding of 1-2 ounce thickness and should provide for some means of cooling the carrier board PCB as well as the ICON H-Bridge modules.

Figure 4: AN603 PCB Layout

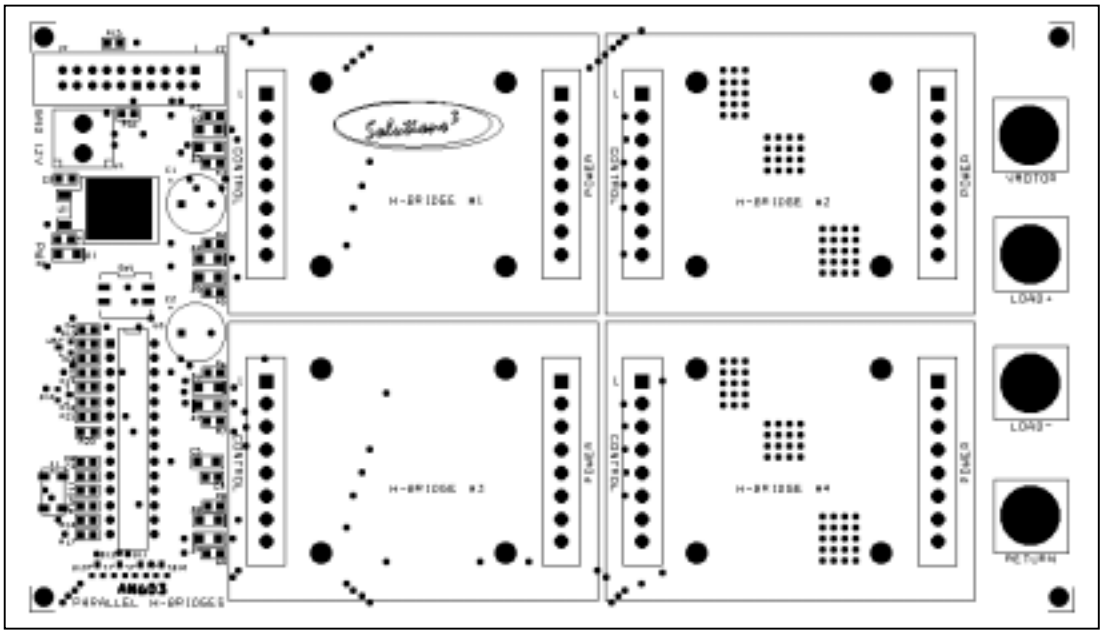


Figure 5: AN603 Bill of Materials

Item Number	Quantity	Part Reference	Description	Manufacturer	Manufacturer Part Number
1	2	C1 C2	330UF 35V RADIAL 1050MA RIP	Panasonic	EEU-FC1V331
2	4	C3 C4 C6 C7	.01UF 50V CERAMIC CAP 0805 SMD	Panasonic	ECU-V1H103KBG
3	1	C5	CAP 2.2UF 16V CERAMIC Y5V 1206	Murata	GRM42-6Y5V225Z016AK
4	2	C8 C9	27PF 50V CERAMIC CAP 0805 SMD	Panasonic	ECU-V1H270JCG
5	5	D1 D2 D4 D6 D8	SMT 1206 GREEN LED	LiteOn	LTST-C150GKT
6	4	D3 D5 D7 D9	SMT 1206 RED LED	LiteOn	LTST-C150EKT
7	4	D10 D11 D12 D13	DIODE 5.6V ZENER TO236/SOT23	Diodes Inc.	BZX84C5V6
8	1	D14	4.096V 0.2% REFERENCE	National	LM4040BIM3-4.1
9	1	J1	CONN TERM BLOCK 2POS 5.08MM	Phoenix Contact	1715721
10	1	J2	DUAL HEADER 20 VERT. 0.1"	3M	2520-6002UB
11	10	R1 R2 R3 R4 R5 R6 R7 R8 R9 R13	RES 1.0K OHM 1/10W 5% 0805 SMD	Panasonic	ERJ-6GEYJ102
12	7	R10 R15 R16 R18 R19 R20 R21	RES 10K OHM 1/10W 5% 0805 SMD	Panasonic	ERJ-6GEYJ103
13	4	R11 R12 R14 R17	RES 120 OHM 1/10W 5% 0805 SMD	Panasonic	ERJ-6GEYJ121
14	1	SW1	SWITCH 2 POS DIP TAPE SEAL SMD	CTS Corporation	219-2LPST
15	1	U1	5V LINEAR REG. D2PAK	On Semi	MC7805B-D2T
16	4	U2 U4 U6 U7	ICON H-BRIDGE REV3	Solutions Cubed	ICON_HBREV3
17	1	U3	MICROCONTROLLER	Microchip	PIC16F873-20I/SP
18	1	U5	RESET CIRCUIT	Microchip	MCP809T-300I/TT
19	1	X1	CRYSTAL 19.66MHZ SMD	Epson	MA-306-19.660M-CQ

Additional Hardware Information:

DIP Switch – This component (labeled SW1 on the PCB and schematic) is used to set some operating parameters in this application. This part is a dual surface mount DIP switch. Switch one on the dual DIP switch selects between single or quad H-bridge operation. Switch two can be used to disable or enable the H-bridges that are selected with switch one.

Figure 6: Dual DIP Switch (SW1) Settings

Switch	ON	OFF
Switch 1	Only H-bridge 1 is operational	H-bridges 1, 2, 3, and 4 are operational
Switch 2	H-bridge operation enabled	H-bridge operation disabled

If a fault condition occurs when operating in direct drive mode the only way to clear the fault is by disabling the H-bridge with a fault and then re-enabling it. This can be done with switch 2. When an H-bridge is enabled the green indicator LED associated with it will be lit. When the H-bridge is disabled the red status LED is lit. If a fault condition has occurred both LEDs will be lit.

Serial Communication Link - A serial communication link was used in this application note to send speed and H-bridge status data to an ASCII terminal. The hardware is connected for bi-directional communication but this application note only supports sending data. The serial data is configured for 38.4KBPS, 8N1, TTL level, true data. An ICON Adapter Board is connected to component J2 via a ribbon cable and is used to provide the analog control signal and to convert the TTL serial data to RS232 levels.

LED Indicators – As previously mentioned each ICON H-Bridge module has a set of indicator LEDs associated with it. When an H-bridge is enabled the green indicator LED will be lit. When disabled the red LED is lit. If an over-current or over-temperature fault occurs then both LEDs will be lit. There is one hardware consideration associated with the red status LEDs. The ICON H-Bridge modules have a pseudo open-collector output. When the ICON H-Bridge outputs 5V it is actually configuring its output pin as an input, and the line is pulled to 5V with a 2.2KΩ resistor. This has the effect of creating a voltage divider between the 2.2KΩ resistor on the H-bridge, the status LED, and the 1.0KΩ resistor associated with the LED. The output voltage due to this relationship is closer to 3V than 5V. In systems where this could have an adverse effect the status LED and 1.0KΩ resistor should be omitted.

Firmware:

The firmware used in this application note was generated using the CCS, Inc. PIC C compiler. The code is pretty straightforward and provides analog control of a unidirectional DC motor.

Essentially the PIC16F873 regularly checks a DIP switch with two switches. These switch settings determine first if any ICON H-Bridge modules should be enabled. Secondly, they are used to determine whether all H-bridges (1,2,3 and 4) should be enabled or whether only H-bridge1 should be enabled. The function that checks the state of the switches is also responsible for reading the status inputs from each H-bridge to determine if a fault has occurred. If a fault has occurred a flag is set for the H-bridge experiencing the fault condition.

After the status of the H-bridges are determined the program will read the 10 bit ADC and move that value to the PWM register used to drive the low side MOSFET (LO-B). If the ADC reads 0 then the high side MOSFET (HI-A) is turned off. The analog input should range from 0-4.096V.

Finally, a serial data interface (38.4KBPS, 8N1, TTL level) sends the H-bridge status information and the PWM settings (as a percentage). This information can be read with a terminal program once the TTL level data is converted to RS232 levels.

Figure 7: AN603.C Flow Chart

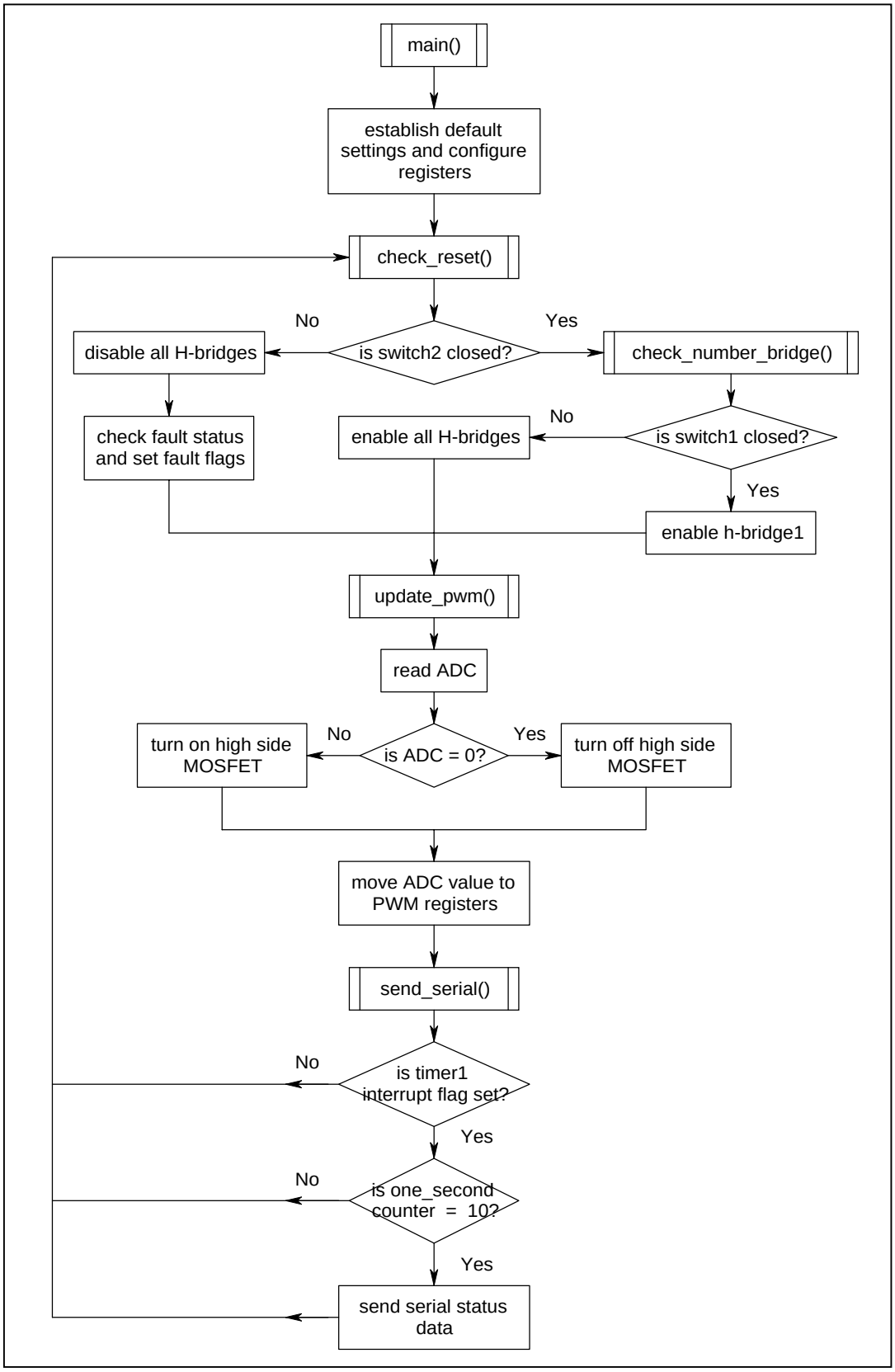


Figure 8: AN603 Code Listing

```
////////////////////////////////////////////////
////////////////////////////////////////////////
//        Application Note 603 (AN603)
//        Parallel ICON H-Bridges
//        Date 04-02-02
//        Description:  This application note allows a single
//        PIC16F873 to control up to 4 ICON H-Bridge modules.
//        These modules are connected in parallel and have
//        been pre-programmed to operate in direct drive mode
//        with an ICON Interface Module before being placed
//        into this system.
//
//        The firmware was generated with the CCS C compiler
//        (www.ccsinfo.com) designed for use with Microchip's
//        (www.microchip.com) line of PIC microcontrollers.
//
//        An ICON Adapter Board was used to introduce a control
//        voltage and to translate TTL level data to RS232
//        levels for the serial output.

//Pin definitions, include files, and compiler directives
#include <16F873.h>
#use delay(clock=19660000)
#fuses HS,NOWDT
#define PIN_ADC_IN   40
#define PIN_BRAKE    41
#define PIN_VREFN    42
#define PIN_VREFP    43
#define PIN_SW1      44
#define PIN_SW2      45
#define PIN_DO3      48
#define PIN_DI3      49
#define PIN_DO2      50
#define PIN_DI2      51
#define PIN_DO1      52
#define PIN_DI1      53
#define PIN_HIA      56
#define PIN_LOA      57
#define PIN_LOB      58
#define PIN_HIB      59
#define PIN_DO4      60
#define PIN_DI4      61

#use rs232(baud=38400,parity=N,xmit=PIN_C6,rcv=PIN_C7)
#zero_ram                                    //clear ram
#use    fast_io(A)                           //Ensure port direction is not changed
#use    fast_io(B)                           //  unless specified by code
#use    fast_io(C)
```

```
//PIC RAM register and bit definitions
#byte   PortA      = 0x05
#byte   PortB      = 0x06
#byte   PortC      = 0x07
#byte   adresh     = 0x1E
#byte   adresl     = 0x9E
#byte    status     = 0x03
#byte    indf       = 0x00
#bit     T1if       = 0x0C.0                  //Define timer 1 interrupt flag

//variable definitions
float    speed;
long    pwmvalue;                             //define global pwm value
int      HB_State;
int      one_second;

#bit     HB1_Enable = HB_State.0               //define flags detailing which H-bridges
#bit     HB2_Enable = HB_State.1               //  should be enabled
#bit     HB3_Enable = HB_State.2
#bit     HB4_Enable = HB_State.3

#bit     HB1_Fault  = HB_State.4               //define flags detailing which H-bridges
#bit     HB2_Fault  = HB_State.5               //  have a fault
#bit     HB3_Fault  = HB_State.6
#bit     HB4_Fault  = HB_State.7

////////////////////////////////////////////////
////////////////////////////////////////////////
// Functions
//
////////////////////////////////////////////////
// check_number_bridges: This function reads the
//         dual DIP switch connected to port A and
//         determines whether the system should be
//         configured for 1 or 4 ICON H-Bridge
//         Modules.
////////////////////////////////////////////////
void check_number_bridges()
  {
  byte  dip_value;                             //define variable for dip switch value
  dip_value = PortA & 0b00010000;              //capture port A dip switch settings
  dip_value = dip_value >> 4;                  //rotate value so it equals 0 or 1

  switch(dip_value)                            //determine case based on dip switch
         {
         case 0:
           HB1_Enable = True;                  //case 0 only one H-bridge in use
           HB2_Enable = False;
           HB3_Enable = False;
           HB4_Enable = False;
           break;
```

```
        case 1:
          HB1_Enable = True;                    //case 1 enables all 4 H-bridges
          HB2_Enable = True;
          HB3_Enable = True;
          HB4_Enable = True;
          break;
        }
  }

////////////////////////////////////////////////
// check_reset:  This function checks the DIP
//        switch connected to port A to see if the
//        ICON H-Bridges can be enabled or if they
//        should be turned off.  If the H-bridges
//        can be turned on then the check_number_bridges
//        function is called.  This routine will also
//        update the status fault flag associated
//        with each H-bridge.
////////////////////////////////////////////////
void check_reset()
  {
  byte  dip_value;                          //define variable for dip switch value
  dip_value = PortA & 0b00100000;           //capture port A,5 dip switch setting
  dip_value = dip_value >> 5;               //rotate value so it equals 0 or 1

  switch(dip_value)                         //determine case based on dip switch
        {
        case 0:
          check_number_bridges();           //case 0 turn on the H-bridges
          break;
        case 1:
          HB1_Enable = False;               //disable H-bridges is off
          HB2_Enable = False;
          HB3_Enable = False;
          HB4_Enable = False;
          break;
        }

  if (input(PIN_DO1) == True)               //Test each H-bridge status flag to see if a
        HB1_Fault = True;                   //  fault condition is present
  else
        HB1_Fault = False;

  if (input(PIN_DO2) == True)
        HB2_Fault = True;
  else
        HB2_Fault = False;

  if (input(PIN_DO3) == True)
        HB3_Fault = True;
  else
        HB3_Fault = False;
```

```
if (input(PIN_DO4) == True)
          HB4_Fault = True;
  else
          HB4_Fault = False;

  }

////////////////////////////////////////////////
// enable_HB:  This function checks the HBx_Enable
//         flag.  If the flag is true then the H-bridge
//         is enabled.
////////////////////////////////////////////////
void enable_HB()
  {
         if (HB1_Enable == True)              //test H-bridge enable flag
           output_low(PIN_DI1);               //  if true then enable H-bridge
         else
           output_high(PIN_DI1);              //  otherwise diable H-bridge

         if (HB2_Enable == True)              //repeat for all four H-bridges
           output_low(PIN_DI2);
         else
           output_high(PIN_DI2);

         if (HB3_Enable == True)
           output_low(PIN_DI3);
         else
           output_high(PIN_DI3);

         if (HB4_Enable == True)
           output_low(PIN_DI4);
         else
           output_high(PIN_DI4);
  }

////////////////////////////////////////////////
// update_pwm:  This function reads the 10-bit analog
//         value present at RA0.  The ADC is set up for
//         0-4.096VDC input (4mV per bit).  The C compiler
//         used in this application note should have
//         returned a 10 bit value for direct movement
//         to the PWM register associated with the PIC's
//         PWM generation hardware.  This was not the
//         case and may be due to a coding error on our
//         part or an error in the version of the C
//         compiler used.  In either case the ADC registers
//         were manipulated in this routine to provide
//         10-bit resolution.
////////////////////////////////////////////////
void update_PWM()
  {
```

```
  pwmvalue = read_adc();                    //read ADC at RA0
  pwmvalue = pwmvalue << 2;                 //rotate adresh value  for 2 LSBs in adresl
  bit_set(status,5);                        //select bank 1 RAM
  indf = adresl >> 6;                       //move LSBs to and store in indf register
  bit_clear(status,5);                      //select bank 0 RAM
  pwmvalue = pwmvalue + indf;               //add indf to pwm storage to get 10 bit result
  if (pwmvalue == 0)                        //if value is equal to 0 then turn off high side
        output_low(PIN_HIA);                //  driver via control pin
  else
        output_high(PIN_HIA);               //if ADC not equal zero turn on high side
                                            //  driver via control pin
  set_pwm1_duty(pwmvalue);                  //use ADC to set CCP1 PWM value
  }

////////////////////////////////////////////////
// send_serial:  This function sends the PWM value
//         as a percentage (0-100%) and is labeled
//         "Speed".  A message associated with each
//         H-bridge is sent detailing the state of
//         each H-bridge (enabled or disabled) and if
//         the H-bridge is enabled its status is also
//         sent (OK or FAULT).  Data is sent in ASCII
//         format at 38.4KBPS, 8N1.  This data can be
//         displayed on any ASCII terminal if the data
//         is converted to RS232 format.
////////////////////////////////////////////////
void send_serial()                          //this routine sends the duty cycle value
  {                                         //  and the status of each H-bridge
  if (T1if == True)                         //test for timer overflow which occurs
                                            //  every 100ms
        {
        SET_TIMER1(0);                      //clear timer 1
        T1if = False;                       //clear timer 1 overflow flag
        if (one_second == 10)               //test 1 second timer counter
          {
          one_second = 0;                   //reset 1 second timer counter

          speed = pwmvalue                  //store PWM in 32 bit variable
                                            //send speed value as XXX.X% setting
          printf("\n\rSpeed = %3.1f%%\n\r",(speed*100/1023));

          if (HB1_Enable == True)           //determine state of H-bridge
                {
                printf("H-Bridge1 Enabled"); //display H-bridge enabled message
                if (HB1_Fault == True)
                      printf("  FAULT\n\r"); //display fault message
                else
                      printf("  OK\n\r");   //display OK message
                }
          else                              //display disabled message
                printf("H-Bridge1 Disabled\n\r");
```

```
            if (HB2_Enable == True)                 //repeat for all H-bridges
                    {
                    printf("H-Bridge2 Enabled");
                    if (HB2_Fault == True)
                            printf("  FAULT\n\r");
                    else
                            printf("  OK\n\r");
                    }
            else
                    printf("H-Bridge2 Disabled\n\r");

            if (HB3_Enable == True)
                    {
                    printf("H-Bridge3 Enabled");
                    if (HB3_Fault == True)
                            printf("  FAULT\n\r");
                    else
                            printf("  OK\n\r");
                    }
            else
                    printf("H-Bridge3 Disabled\n\r");

            if (HB4_Enable == True)
                    {
                    printf("H-Bridge4 Enabled");
                    if (HB4_Fault == True)
                            printf("  FAULT\n\r");
                    else
                            printf("  OK\n\r");
                    }
            else
                    printf("H-Bridge4 Disabled\n\r");
          }
           else
             one_second = one_second + 1;
      }
    }

////////////////////////////////////////////////
//  main:  The main function sets up many of the
//         registers used by this application note
//         and calls four functions.
////////////////////////////////////////////////
void main()
{
  setup_adc_ports(RA0_ANALOG_RA3_RA2_REF);   //configure ADC for RA0 vref at RA2,3
  setup_adc(ADC_CLOCK_DIV_32);    //ADC clock setting
  setup_spi(FALSE);               //disable SPI port
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);    //timer0 is Fosc/2
  setup_timer_1(T1_DIV_BY_8|T1_INTERNAL);      //timer1 divide by 8
  setup_timer_2(T2_DIV_BY_1,255,1);            //update PWM every period
  setup_ccp1(CCP_PWM);            //CCP1 set for PWM function
  setup_ccp2(CCP_PWM);            //CCP2 set for PWM function
  set_uart_speed(38400);          //set serial data speed to 38400
  set_tris_a(0xFF);               //establish I/O settings for ports
  set_tris_b(0xD5);
```

```
  set_tris_c(0x90);

  set_adc_channel(0);              //select RA0 for ADC input
  delay_us(10);                    //delay 10us for ADC setting

  output_low(PIN_LOA);             //initialize H-Bridge control lines to
  output_low(PIN_HIB);             //  a disabled state
  output_low(PIN_LOB);
  output_low(PIN_HIA);
  output_high(PIN_DI1);
  output_high(PIN_DI2);
  output_high(PIN_DI3);
  output_high(PIN_DI4);
  set_pwm2_duty(0);                //initialize PWM output to 0
  set_pwm1_duty(0);
  delay_ms(1000);                  //wait a second before running main loop

while( True )                      //This is the main program loop
  {
  check_reset();                   //check dip switch for user reset
  enable_HB();                     //enable h-bridges based on HBx_Enable flags
  update_PWM();                    //read ADC at RA0 and use value for PWM setting
  send_serial();                   //send program data to PC serial port
  }
}
```

Summary:

This application note provides one method that may be implemented to allow a single device to control multiple ICON H-Bridge modules. The hardware and components used in this design are flexible enough to be used for a much more complex control system. The modular nature of the ICON H-Bridge and its simplified “direct-drive” mode of control were exploited in this design to provide the components of a medium current motor controller. Additionally, the fault protections built into the ICON H-Bridge provided excellent protection for both the H-bridges and the load connected to them. The status output flags were also useful in ensuring that the controlling device was aware of fault conditions.

In this application note the ICON H-Bridge was easily designed into a system that otherwise would require significant time to engineer. For low or medium volume production runs a design utilizing parallel ICON H-Bridge modules is a superior solution.