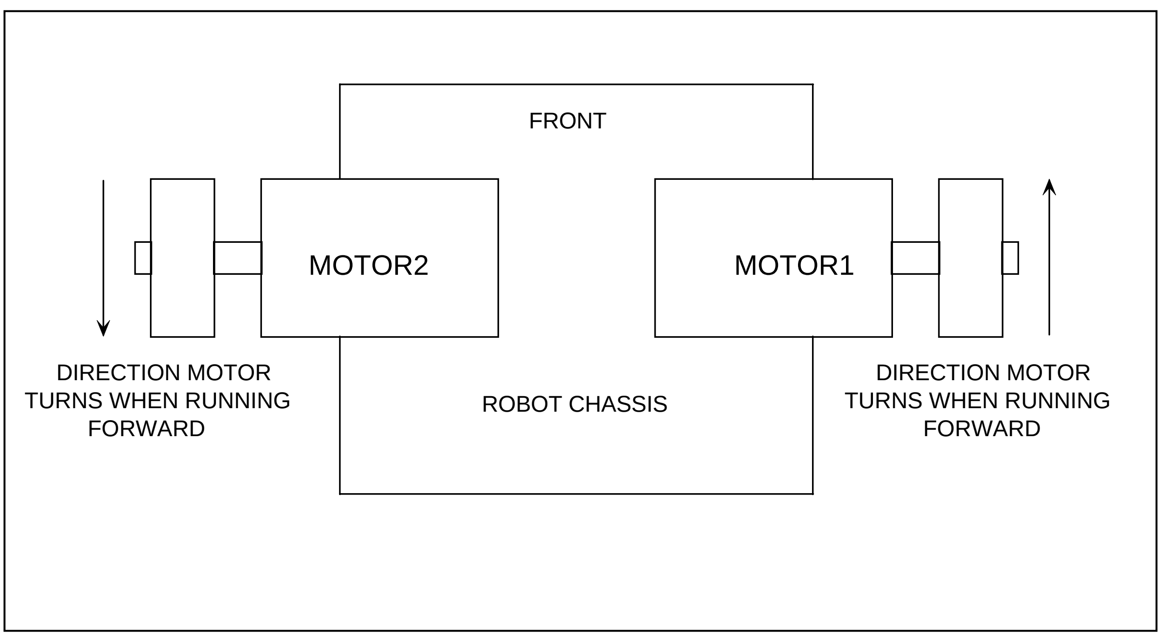
Overview:
From students doing science projects to garage-based tech warriors, many people are becoming involved in robotics. AN600 depicts a method of interfacing a BASIC Stamp 2(produced by Parallax) to two ICON Interface Modules and two ICON H-Bridges to create a two-wheeled robotic drive system. This application takes AN600 a step further by introducing radio control to the drive system. The radio interface takes the form of the standard hobbyist radio control transmitter and receiver used for R/C cars, boats, and airplanes. The radio receiver is connected to an ICON-BS2 carrier board (PN: ICON_BS2). The BASIC Stamp 2 (BS2 – sold separately) reads 2 channels from the receiver and converts them into serial data strings which are sent to the 2 ICON Interface Modules. In this design channel one is used for steering while channel two controls the speed of the drive motors. Mixing of the two channels occurs after the BS2 has converted them to numeric values and is done with a simple algorithm.

Hardware:
The hardware required for this application note is listed below.

1 – 2 channel R/C receiver and transmitter
1 – Parallax BASIC Stamp 2
1 – ICON – BS2 Carrier Board
2 – ICON Interface Modules
2 – ICON H-Bridges
1 – 12V-36V battery(s)
2 – drive motors

The system is designed so that motor 1 (controlled by ICON Interface Module #1) mounts on the right-hand side of the robot. Motor 2 has its motor setting reversed so that both motors turn in the same direction when the robot is driven forward or reversed. Figure 1 shows the direction the two motors would turn if they were both given speed values that equated to the motor turning forward. Not shown in the diagram is a pivot wheel generally used in two-wheeled robots for balance.

Figure 1: Robot Chassis Motor Mounting



This system would work best with a battery based power supply in the 12V-36V range. The ICON H-Bridge modules can handle roughly 7A without cooling in open air. With the ICON Active Cooling solution (PN: ICON_AC) the continuous current rating is closer to 12A. You should first decide on your motor voltage and battery voltage and then set the power supply selection jumpers on both ICON Interface Modules as per figure 2. In most cases the jumper setting will be at jumper 3-4 or jumper 5-6. It is possible to run motors that are powered by less than 10VDC. To do this you would connect jumper 1-2 and provide 12VDC to the points labeled VEXT1 and VEXT2 on the ICON-BS2 carrier board. This 12V supply powers all of the ICON products, while your motor can be powered off of 1-10VDC.

Figure 2: ICON Interface Module Power Supply Jumper Settings

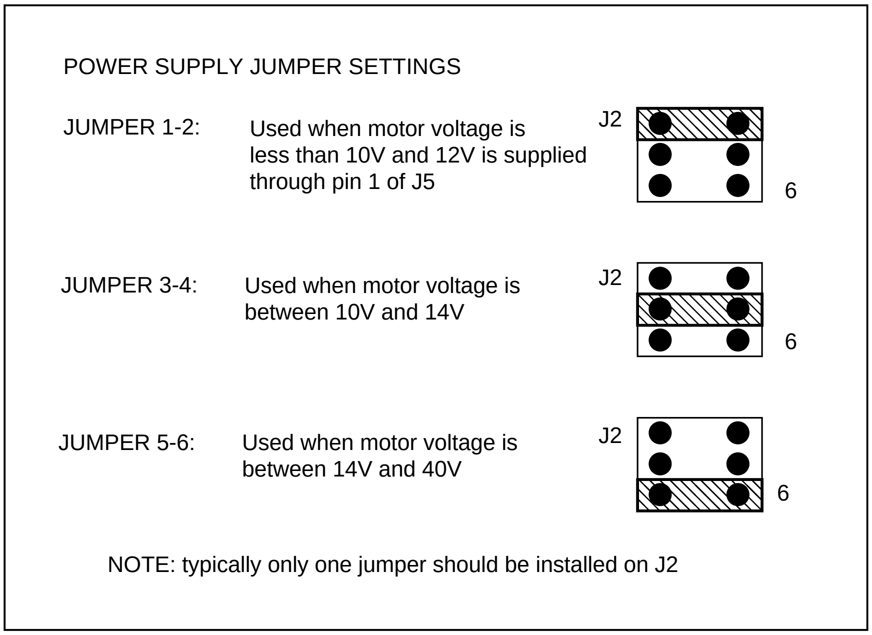
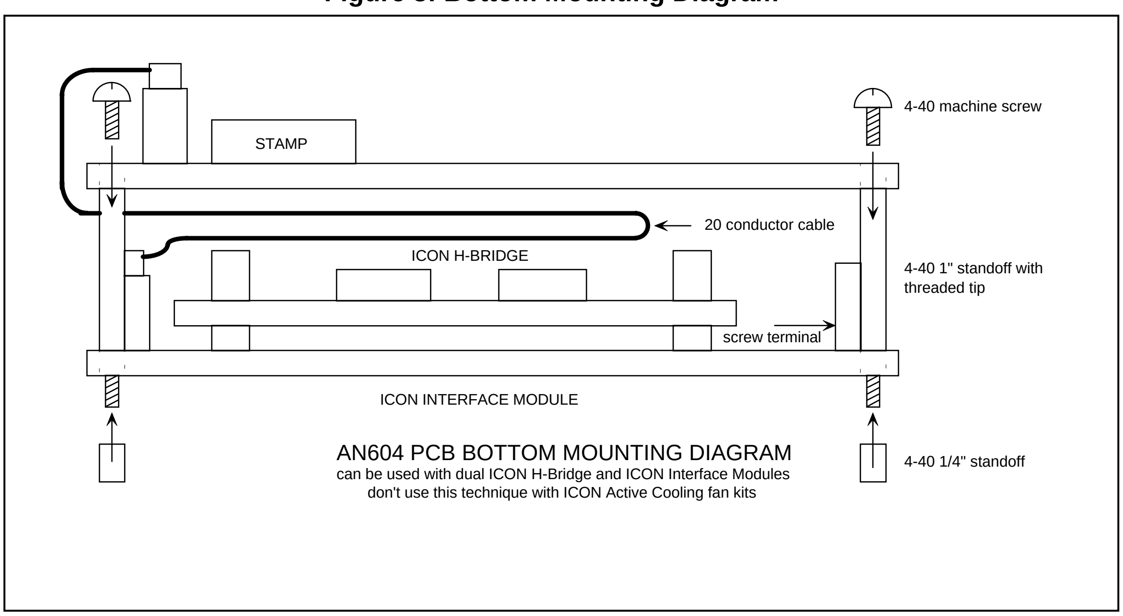


Figure 3: Bottom Mounting Diagram

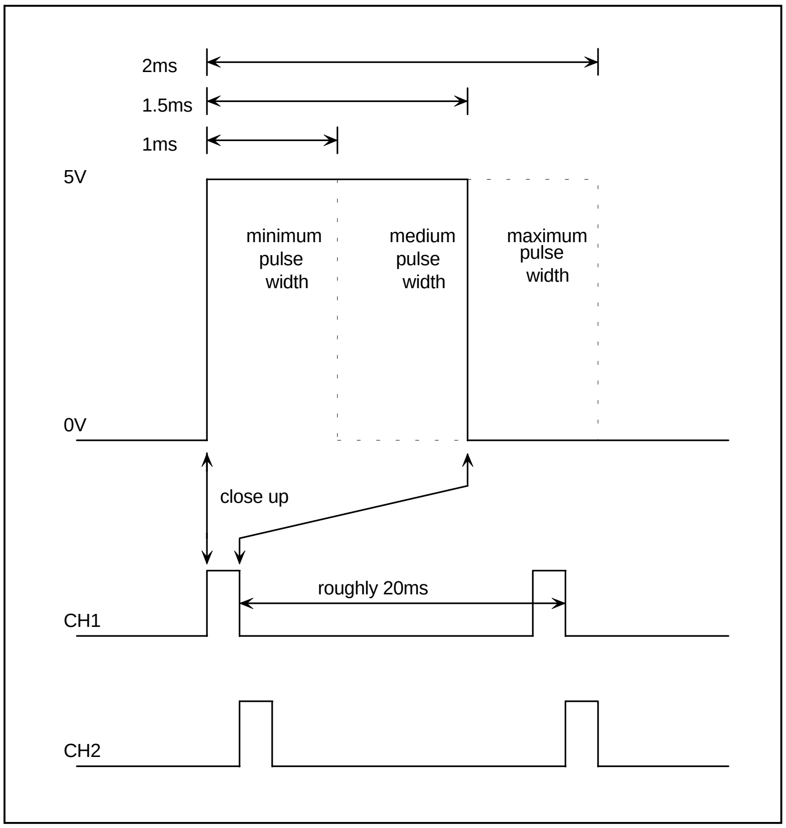


R/C Receiver Pulse Manipulation:

The BS2 performs three primary functions in this design. The first function performed by the BS2 is to read in the pulses generated by the R/C receiver. The second function is to modify the numeric values associated with the pulse lengths so that steering and speed information can be converted into serial data strings. Finally the BS2 communicates this information to the 2 ICON Interface Modules which in turn control the ICON H-Bridges driving the motors.

The BS2 measures the pulse widths using the PULSIN function. For each channel of the receiver a 16-bit value is returned. The BS2 measures the pulse width in increments of 2uS. Since a standard servo signal ranges from 1ms to 2ms, the expected values returned by the BS2 range from 500 to 1000. A value of 750 would be returned if the input pulse were 1.5ms in duration. Figure 4 displays the electrical format normally taken by an R/C receiver pulse.

Figure 4: R/C Pulse Format



The BS2 is responsible for capturing these pulses and does so by measuring the pulse at channel 1 and then channel 2 of the receiver. The pulses should be read in the order they are received (they are typically staggered with lower channels being received first). On our receiver channel 1 was associated with a joystick on the transmitter that moved left and right. Therefore it was used as the “steering” channel. This could vary by manufacturer. For the rest of this document when we refer to CH1 we will be referring to the “steering” channel while CH2 refers to the channel that is related to motor speed and direction(forward or reverse).

Initially both CH1 and CH2 are limited to values between 500 and 1000 by the BS2. The BS2 code actually allows for pulse signals from 0.5ms to 2.5ms to be used as valid signals, but limited to the range mentioned previously. Furthermore, values between 735 and 765 are forced to 750. This allows for a dead-band around the value associated with 0 speed and 0 steering. Figure 5 shows specific values and how they are associated with each motor. Note that these values are not the values sent to the ICON Interface Modules, they are just the pulse width values and the relationship they have with each motor. The values in the CH1 and CH2 columns are the raw pulse widths received. The values in the Motor1 and Motor2 columns are the values after "mixing" has occurred. Take note that if the steering pulse (CH1) is in its dead-band (750) then there is no steering effect on the speed pulse (CH2). You can also see that if CH1 is less than 750 (transmitter joystick for CH1 pushed to the left) the speed for Motor2 is reduced. Alternately, if CH1 is greater than 750 then the speed for Motor1 is reduced. In between the limits described below the effect on the motors is proportional. Finally, keep in mind that these values are not yet converted to the values needed by the ICON Interface Modules for speed control, and the motor 2 value is forced negative when the pulse width value is converted to a serial data value.

Figure 5: Pulse Widths and Their Functions

Description	CH1	CH2	Motor1	Motor2
Stop	Don't Care	750	750	750
Full forward	750	1000	1000	1000
Full Reverse	750	500	500	500
Hard Left Full Forward	500	1000	1000	750
Hard Right Full Forward	1000	1000	750	1000
Hard Left Full Reverse	500	500	500	750
Hard Right Full Reverse	1000	500	750	500

It is easiest to describe the modifications that the BS2 makes to the pulse width values with logic statements, as these correlate well with the software used to program the BS2. The first portion of the code reads the pulse widths and then limits the pulse values to a range from 500-1000. These limits are placed on CH1 and CH2 pulses and are done as follows.

```
If RC_PULSE > 1250                  Then RC_PULSE is = 750
If RC_PULSE > 1000 and < 1251       Then RC_PULSE is = 1000
If RC_PULSE < 765 and > 735         Then RC_PULSE is = 750
If RC_PULSE < 500 and > 250         Then RC_PULSE is = 500
If RC_PULSE < 251                   Then RC_PULSE is = 750
```

This logic is applied to both CH1 and CH2. Now to use CH1 to affect steering the pulse width associated with CH1 is used to modify the CH2 pulse width. It is assumed that this logic takes effect after the pulse width from CH2 has been moved into the variables PWM_REG1, and PWM_REG2. Furthermore CH1 and CH2 are described by the RC_PULSE1 and RC_PULSE2 variables. I have called this process mixing, as it is similar to a mixing process used with R/C aircraft to control flight surfaces. Note again that the modifications made to the PWM_REGx values do not prepare them for direct transmission to the ICON Interface Modules.

```
PWM_REG1 = RC_PULSE2
PWM_REG2 = RC_PULSE2

If RC_PULSE1 > 750 and PWM_REG1 > 750 Then
                                PWM_REG1 = PWM_REG1 – (RC_PULSE1 – 750)
If RC_PULSE1 > 750 and PWM_REG1 < 750 Then
                                PWM_REG1 = PWM_REG1 – (750 – RC_PULSE1)

If RC_PULSE1 < 750 and PWM_REG2 > 750 Then
                                PWM_REG2 = PWM_REG2 – (750 – RC_PULSE1)
If RC_PULSE1 < 750 and PWM_REG2 < 750 Then
                                PWM_REG2 = PWM_REG2 – (RC_PULSE1 – 750)
```

Some additional limiting of the PWM_REGx variables is done to ensure that the values are maintained on the correct side of 750. In other words, if the value in the PWM_REGx was greater than 750 before the mixing occurs, then it is forced to a value no less than 750 after the mixing is completed.

Finally the PWM_REG1 and PWM_REG2 values must be modified to a format that works with the ICON Interface Module serial command structure. The values stored in the PWM_REGx variables after mixing will range from 500-1000. The ICON Interface Modules use negative values for reverse and positive values for forward. The values sent the ICON Interface Module should be in the range of –1023 to 1023 (if the values exceed this range the ICON Interface Modules will limit the value to -1023 or 1023). Since a pulse value of 750 is associated with the stopped, or no speed, condition we would want a value of 750 to be converted to 0 before being sent to the ICON Interface Module. Similarly a value of 500 (full reverse) should equal –1023, and a value of 1000 (full forward) should equal 1023. We therefore first want to shift the pulse width by 750.

PWM_REG1 = PWM_REG1 – 750

This gives us a new range of values from –250 to 250. To expand these values to a range from –1023 to 1023 we can multiply the result. I used a multiplier of 5 because my receiver did not actually provide 1-2ms signals. This ensured that the minimum and maximum pulse widths generated by my receiver resulted in full reverse and full forward. The end result is that the speed value for Motor1 is calculated from the equation…

PWM_REG1 = 5*(PWM_REG1 – 750)

And since the value for Motor2 should always be reversed from Motor1 the speed value for Motor2 is calculated with the equation…

PWM_REG2 = -5*(PWM_REG2 – 750)

These values are now ready for transmission to the ICON Interface Modules.

System Configuration:

The hardware in this system is pretty straightforward to set up. Figure 2 displays the possible jumper settings for the power supply on the ICON Interface Modules. Figure 3 depicts a possible mounting method for connecting the ICON-BS2 carrier board to the ICON Interface Modules. If the ICON Active Cooling Solution is used with the ICON Interface Module then the ICON Interface Modules should be mounted on top of the ICON-BS2 carrier board. The ICON-BS2 datasheet has a mounting diagram for the top-mounting technique.

The receiver is powered from the BS2 5V supply (VCC on the PCB, VCC_STAMP on the schematic), which is generated on the ICON Interface Module1 PCB. BS2 pins P8 and P9 are used with the PULSIN command to read the R/C receiver's pulse outputs. CH1 is connected to P8 and CH2 is connected to P9. The R/C receiver is also powered off of the 5V supply.

The system connections are shown on figure 6. The 2 ICON H-Bridge modules connected to each ICON Interface Module are not shown in the schematic.

Figure 6: AN604 System Schematic

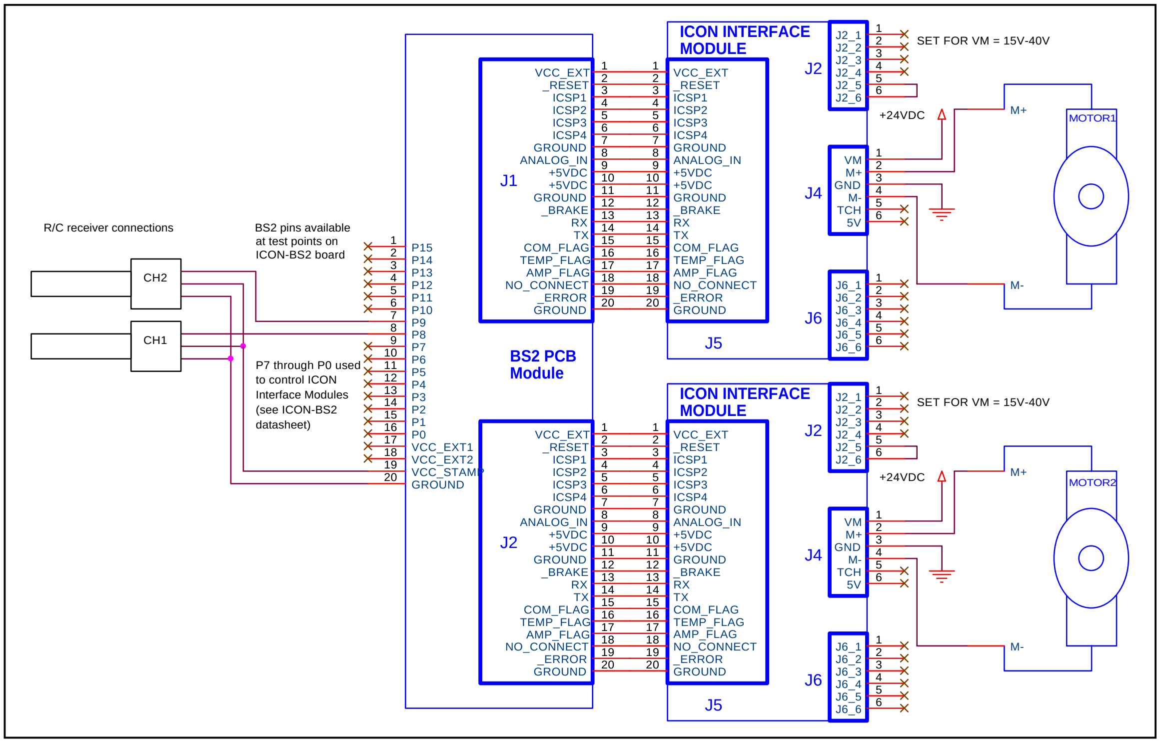


Figure 7: AN604.BS2 Software Listing

```
'AN604 An R/C Interface for Robot Control
'
'This application note is based on the ICON - BS2 interface board,
'2 ICON Interface Modules, and 2 ICON H-Bridges. The BS2 reads
'2 channels from an R/C receiver and uses them to generate speed
'and steering settings.  These settings are converted to serial
'data strings and sent to the two ICON Interface Modules used
'to drive the ICON H-Bridges.

'Communication string variables
        CMMD            VAR     BYTE            'Command byte storage
        ADDR            VAR     BYTE            'Address byte storage
        LENG            VAR     BYTE            'Length byte storage
        CKSUM           VAR     BYTE            'Checksum byte storage
        DAT1            VAR     BYTE            'Data byte registers
        DAT2            VAR     BYTE
        DAT3            VAR     BYTE
        DAT4            VAR     BYTE
        LOOP            VAR     WORD            'For next loop word
        BADCOMM         VAR     BYTE            'Storage for bad communication counter
        RC_PULSE1       VAR     WORD            'Input pulse storage register channel 1
        RC_PULSE2       VAR     WORD            'Input pulse storage register channel 2

'PWM storage registers
        PWM_REG1        VAR     WORD            'PWM storage register for ICON Module 1
                P1HI    VAR     PWM_REG1.HIGHBYTE
                P1LO    VAR     PWM_REG1.LOWBYTE
        PWM_REG2        VAR     WORD            'PWM storage register for ICON Module 2
                P2HI    VAR     PWM_REG2.HIGHBYTE
                P2LO    VAR     PWM_REG2.LOWBYTE

'Program constants
        BAUD    CON     6                       'Use BAUD = 6 for BS2
```

```
'ICON Interface Module #1 control lines
        RESET1  CON     0               'Causes hardware reset when pulled low
        BRAKE1  CON     1               'Implements braking function when pulled low
        DOUT1   CON     2               'TTL serial data to ICON Interface Module
        DIN1    CON     3               'TTL serial data from ICON Interface Module

'ICON Interface Module #2 control lines
        RESET2  CON     4               'Causes hardware reset when pulled low
        BRAKE2  CON     5               'Implements braking function when pulled low
        DOUT2   CON     6               'TTL serial data to ICON Interface Module
        DIN2    CON     7               'TTL serial data from ICON Interface Module

'Set BS2SX i/o direction and level
        DIRS            =%0000000001110111      'Set P0,P1,P2,P4,P5,P6 as outputs all others inputs
        OUTS            =%1111111111111111      'Set all outputs high

        LOW             RESET1          'Start program by resetting ICON Interface Modules
        LOW             RESET2
        PAUSE           5
        HIGH            RESET1
        HIGH            RESET2
        PAUSE           750             'Wait 750ms for ICON Interface Modules to power up
        DEBUG           CLS             'Clear debug screen

        GOSUB           INIT_IM         'Initialize the ICON Interface Modules

START:

        GOSUB           GET_PULSES
        GOSUB           MIX_PULSES
        GOSUB           CONVERT_TO_SERIAL
        GOSUB           SETDC_IM1
        GOSUB           SETDC_IM2

        GOTO            START           'Return to start of program

'*********************************** Subroutines *************************************

'*****************************************************************************************
'INIT_IM:       This subroutine initializes the ICON Interface Modules by ensuring that the
'               ADDRESS registers are programmed to "1" and programming the IM_FUNCTION
'               registers to binary %1010000.  The IM_FUNCTION value enables dynamic
'               braking and amps retry settings (see the communication protocol for
'               more information on these functions).  The ADDRESS and IM_FUNCTION register
'               are written to using the universal address of "0", therefore no ACK will be
'               be received from the ICON Interface Modules.  The BS2 then sends the STORE
'               command to each ICON Interface Module and waits for the ACK.  This process
'               occurs every time the circuit is powered up, but in reality it only needs to
'               occur once since the settings are stored in EEPROM with the STORE command.
'*****************************************************************************************
INIT_IM:
        CMMD            = $D2           'WRITE command
        ADDR            = $00           'Universal address
        LENG            = $04           'Message length
        DAT1            = $0E           'ADDRESS register index value
        DAT2            = $01           'Write "1" to ADDRESS register
        DAT3            = $0F           'IM_FUNCTION register index value
        DAT4            = $A0           'Write %1010000 to IM_FUNCTION register
        CKSUM           = CMMD+ADDR+LENG+DAT1+DAT2+DAT3+DAT4
        SEROUT DOUT1,BAUD,[CMMD,ADDR,LENG,DAT1,DAT2,DAT3,DAT4,CKSUM]
        SEROUT DOUT2,BAUD,[CMMD,ADDR,LENG,DAT1,DAT2,DAT3,DAT4,CKSUM]

        PAUSE           20
```

```
        CMMD            = $D3                   'STORE command
        ADDR            = $01                   'ICON Interface Module address of "1"
        LENG            = $00                   'Length of 0, no data in command
        CKSUM           = CMMD+ADDR+LENG
        SEROUT DOUT1,BAUD,[CMMD,ADDR,LENG,CKSUM]
        SERIN   DIN1,BAUD,150,NA_INIT1,[DAT1]
        IF DAT1 = $6 THEN A_INIT1               'Wait 150ms for an ACK from module 1
NA_INIT1:
        DEBUG           "NO ACK INIT1",CR
A_INIT1:
        SEROUT DOUT2,BAUD,[CMMD,ADDR,LENG,CKSUM]
        SERIN   DIN2,BAUD,150,NA_INIT2,[DAT1]
        IF DAT1 = $6 THEN A_INIT2               'Wait 150ms for an ACK from module 2
NA_INIT2:
        DEBUG           "NO ACK INIT2",CR
A_INIT2:
        RETURN

'*****************************************************************************************
'SETDC_IM1:     This routine sends speed and direction data to the ICON Interface Module
'               number one located on the right hand side of the robot.  The speed and
'               direction data are stored in the PWM_REG1 register.  If an ACK is not
'               received within 150ms then the BS2 will attempt to send the command again.
'               This retry will be attempted up to 5 times.
'*****************************************************************************************

SETDC_IM1:
'       DEBUG           "PWM1 = ",ISHEX4 PWM_REG1,TAB
        CMMD            = $D0                   'SETDC command
        ADDR            = $01                   'ICON Interface Module address of "1"
        LENG            = $02                   'Length of SETDC is 2
        CKSUM           = CMMD+ADDR+LENG+P1LO+P1HI
        SEROUT DOUT1,BAUD,[CMMD,ADDR,LENG,P1HI,P1LO,CKSUM]
        SERIN   DIN1,BAUD,150,NA_SDC1,[DAT1]
        IF DAT1 <> $6 THEN NA_SDC1
        BADCOMM = 0
        RETURN
NA_SDC1:
        BADCOMM = BADCOMM+1
        IF BADCOMM < 5 THEN SETDC_IM1
        BADCOMM = 0
        RETURN

'*****************************************************************************************
'SETDC_IM2:     This routine sends speed and direction data to the ICON Interface Module
'               number two located on the left hand side of the robot.  The speed and
'               direction data are stored in the PWM_REG2 register.  If an ACK is not
'               received within 150ms then the BS2 will attempt to send the command again.
'               This retry will be attempted up to 5 times.
'*****************************************************************************************

SETDC_IM2:
'       DEBUG           "PWM2 = ",ISHEX4 PWM_REG2,CR
        CMMD            = $D0                   'SETDC command
        ADDR            = $01                   'ICON Interface Module address of "1"
        LENG            = $02                   'Length of SETDC is 2
        CKSUM           = CMMD+ADDR+LENG+P2LO+P2HI
        SEROUT DOUT2,BAUD,[CMMD,ADDR,LENG,P2HI,P2LO,CKSUM]
        SERIN   DIN2,BAUD,150,NA_SDC2,[DAT1]
        IF DAT1 <> $6 THEN NA_SDC2
        BADCOMM = 0
        RETURN
NA_SDC2:
        BADCOMM = BADCOMM+1
        IF BADCOMM < 5 THEN SETDC_IM2
        BADCOMM = 0
        RETURN
```

```
'*****************************************************************************************
'GET_PULSES:     This subroutine reads the two R/C pulse inputs the data is then stored
'                in the RC_PULSE1 and RC_PULSE2 registers.  The pulse values are limited
'                to a range of 500 to 1000.  750 is a stop condition, 500 is full reverse
'                for channel 2 and hard-left for channel 1.  Likewise, 1000 is full forward
'                for channel 2 and hard-right for channel 1.  A dead band is included
'                between 740-760 (1.47-1.53ms).  The values below (particularly the dead
'                band area) may need to be adjusted based on the output of your R/C
'                transmitter and receiver.
'*****************************************************************************************

GET_PULSES:
        PULSIN  8,1,RC_PULSE1
        PULSIN  9,1,RC_PULSE2

        IF RC_PULSE1 > 1250        THEN RC1_750
        IF RC_PULSE1 > 1000        THEN RC1_1000
        IF RC_PULSE1 > 760         THEN RC1_PROP
        IF RC_PULSE1 > 740         THEN RC1_750
        IF RC_PULSE1 > 500         THEN RC1_PROP
        IF RC_PULSE1 > 250         THEN RC1_500
RC1_750:
        RC_PULSE1 = 750
        GOTO RC1_PROP
RC1_1000:
        RC_PULSE1 = 1000
        GOTO RC1_PROP
RC1_500:
        RC_PULSE1 = 500
        GOTO RC1_PROP
RC1_PROP:

        IF RC_PULSE2 > 1250        THEN RC2_750
        IF RC_PULSE2 > 1000        THEN RC2_1000
        IF RC_PULSE2 > 765         THEN RC2_PROP
        IF RC_PULSE2 > 735         THEN RC2_750
        IF RC_PULSE2 > 500         THEN RC2_PROP
        IF RC_PULSE2 > 250         THEN RC2_500
RC2_750:
        RC_PULSE2 = 750
        GOTO RC2_PROP
RC2_1000:
        RC_PULSE2 = 1000
        GOTO RC2_PROP
RC2_500:
        RC_PULSE2 = 500
        GOTO RC2_PROP
RC2_PROP:

        PWM_REG1 = RC_PULSE2
        PWM_REG2 = RC_PULSE2

        RETURN

'*****************************************************************************************
'MIX_PULSES:     This subroutine takes the values in the RC_PULSEx registers and modifies
'                them so that R/C pulse 1 (stored in RC_PULSE1 register) can be used as a
'                steering signal.
'*****************************************************************************************

MIX_PULSES:
        IF RC_PULSE1 > 750 THEN ADJUST_PWM1
        IF RC_PULSE1 < 750 THEN ADJUST_PWM2
        GOTO    MIX_END

ADJUST_PWM2:
        IF PWM_REG1 > 750 THEN SUB_PWM2
        IF PWM_REG1 < 750 THEN ADD_PWM2
        GOTO    MIX_END
```

```
SUB_PWM2:
	PWM_REG2 = PWM_REG2 - (750 - RC_PULSE1)
	IF PWM_REG2 > 750 THEN EX_SUB2
	PWM_REG2  = 750
EX_SUB2:
	GOTO MIX_END
ADD_PWM2:
	PWM_REG2 = PWM_REG2 - (RC_PULSE1 - 750)
	IF PWM_REG2 < 750 THEN EX_ADD2
	PWM_REG2 = 750
EX_ADD2:
	GOTO MIX_END


ADJUST_PWM1:
	IF PWM_REG1 > 750 THEN SUB_PWM1
	IF PWM_REG1 < 750 THEN ADD_PWM1
	GOTO MIX_END

SUB_PWM1:
	PWM_REG1  = PWM_REG1 - (RC_PULSE1 - 750)
	IF PWM_REG1 > 750 THEN EX_SUB1
	PWM_REG1 = 750
EX_SUB1:
	GOTO MIX_END
ADD_PWM1:
	PWM_REG1  = PWM_REG1 - (750 - RC_PULSE1)
	IF PWM_REG1 < 750 THEN EX_ADD1
	PWM_REG1 = 750
EX_ADD1:
	GOTO MIX_END

MIX_END:
	RETURN

'*****************************************************************************************
'CONVERT_TO_SERIAL: This subroutine takes the PWM values generated from the R/C input
'			pulses and translates them into a format that is compatible the ICON
'			Interface Module communication protocol.  The left hand motor controlled
'			by ICON Interface Module#2 has the PWM_REG2 inverted so that it will run in
'			the opposite direction of the left hand motor.  The multiplier "5" may be
'			reduced or increased based on the actual output pulses from your receiver.
'			The R/C receiver used in our testing had pulse outputs from 1.2-1.8ms.  A
'			smaller range would require a larger multiplier, while a larger range would
'			require a smaller multiplier.
'*****************************************************************************************

CONVERT_TO_SERIAL:
	PWM_REG1 = 5*(PWM_REG1 - 750)
	PWM_REG2 = -5*(PWM_REG2 - 750)
	RETURN

END:
```

Summary:
This application note provides and easy to follow method for converting R/C pulse signals from a standard R/C receiver to speed and steering controls for a two-wheeled robot. The R/C pulses are read, mixed, and converted to serial data strings for use with two ICON Interface Modules carrying two ICON H-Bridge modules. This method is straightforward and requires limited soldering and no external circuitry. The end result is a dependable robotic drive system with medium current handling capability.

Some users may need to modify the application note to meet the particular nature of their R/C receiver. But the software methodology is described in enough detail that this should be a simple matter for individuals who understand the processes described herein.