Innovation First, Inc.

Programming Reference Guide

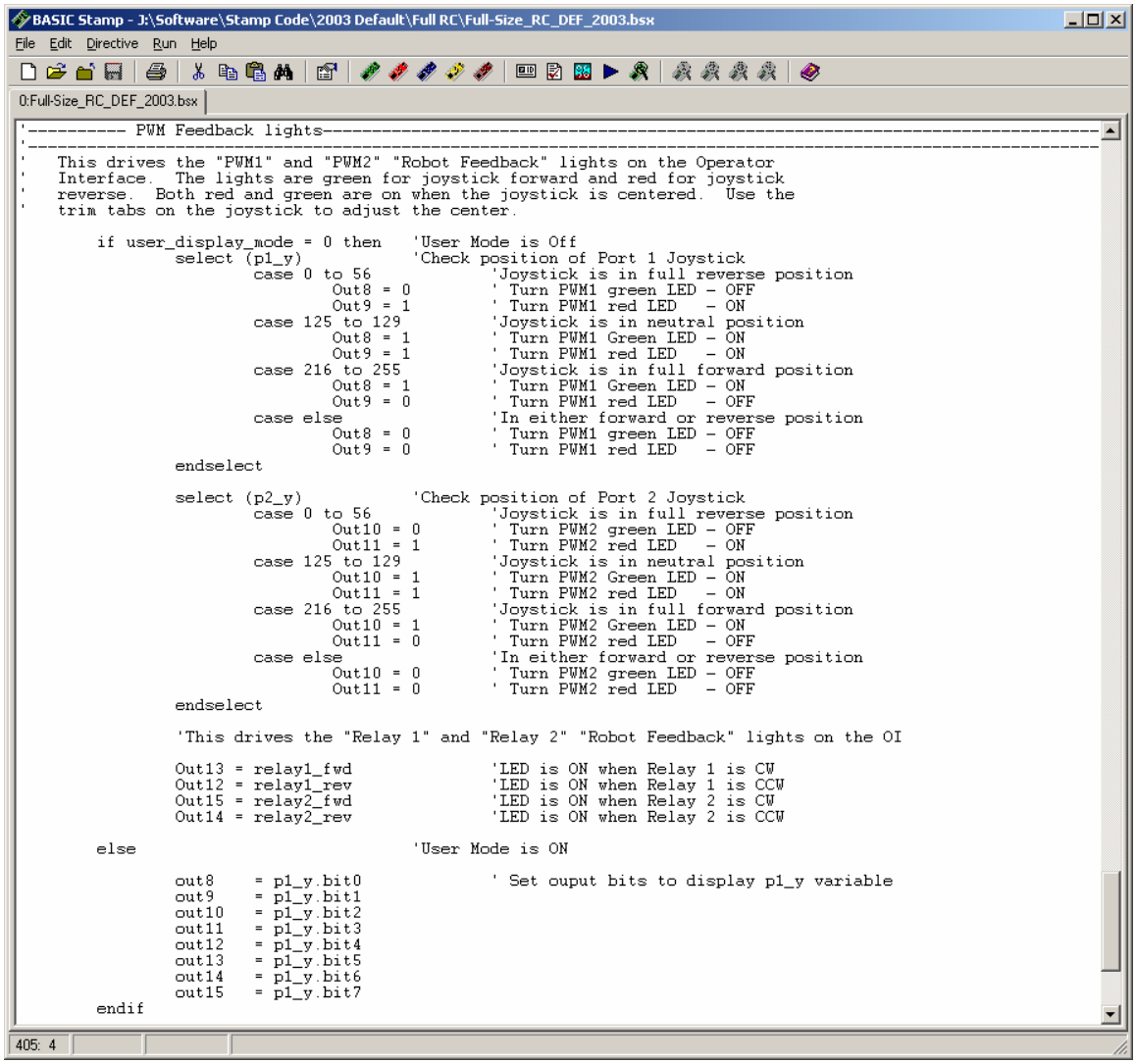
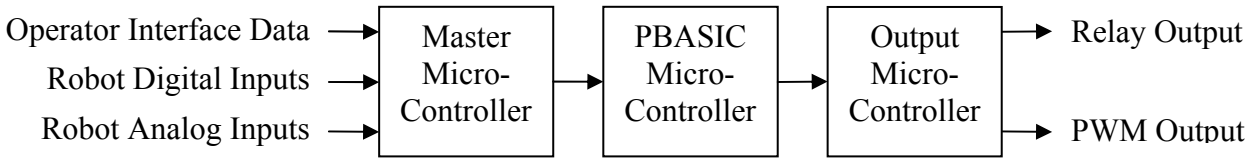


Table of Contents

1. PBASIC Processor and Memory

1.1. Processors

Each Robot Controller has three built-in microcontrollers. One of these microcontrollers, the Parallax BS2-SX 50MHz microcontroller, uses memory that is programmable by the user. This programmability allows customization of user controls, semi-automatic robot functions, and autonomous robot functions. This user processor has access to all user control and robot sensor data, as well as control over the analog and digital outputs used to control motors and relays.



1.2. PBASIC Program Memory

Each Robot Controller has one or two EEPROM memory chips used to store the PBASIC program. This memory is non-volatile; meaning that is keeps its memory after power has been removed. The full-size RC has two memory chips and a special DEF/USER jumper to select which one is being used. The EDU-RC Motherboard has one memory chip located on the EDU Motherboard. This motherboard location allows the robot program to remain in the robot while the Isaac16 EDU Robot Controller (black box) can be removed to be used in multiple robots.

The program on either Robot Controller can be changed by downloading a new program into PBASIC memory through the PROGRAM port (more details on page 14).

1.3. Default vs. Custom Programs

The Robot Controller is supplied with a "Default" program in order to help get the robot up and running quickly. When more sophisticated control of the robot is desired, a custom program can be quickly created by modifying the Default program. Default "Source Code" for the default programs are provided at www.InnovationFirst.com. Both types of Robot Controllers are programmed in the same manner, and the programs are interchangeable from one type to the other. We provide a different default program for each type of Robot Controller, due to the different uses and the different number of Inputs and Outputs available. Therefore, although the programs are interchangeable, the robots function will change when the programs are changed.

1.4. Programming Language and Windows Software

All programs running on the Robot Controller must be written in PBASIC, a dialect of the BASIC programming language. This language was selected because it is fairly easy to learn, use, and debug in

a short period of time. With the exception of the source code for the default program, the programming utility and manuals can be obtained via the Internet from Parallax, Inc. at www.parallaxinc.com.

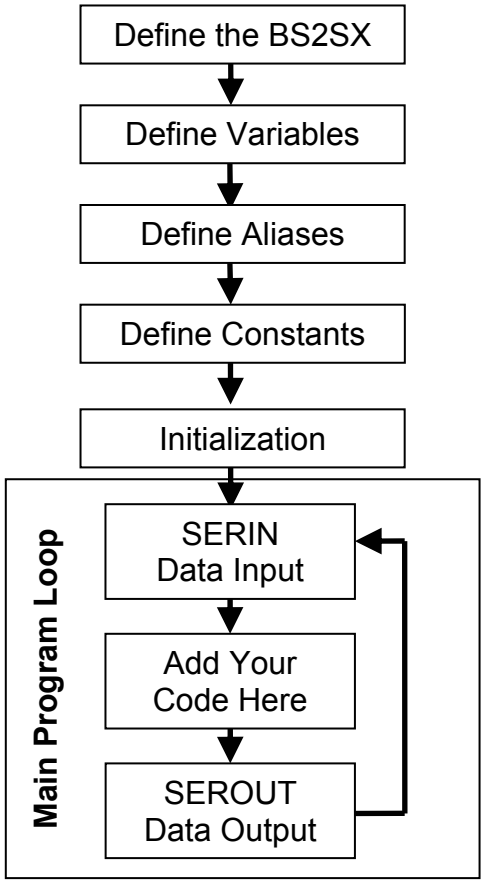
Basic Stamp is the name of the Parallax Windows program used to edit, debug, and download PBASIC programs. The latest version of the BASIC Stamp editor should be used. Check the Innovation First web site for the latest version of the BASIC Stamp editor.

2. PBASIC Software Structure

The PBASIC program used in the Innovation First Robot Controllers must adhere to a pre-defined structure. The structure does not limit the capability of your software; the structure ensures that data is correctly input and output. This structure is described in the block diagram below.

Each block has a corresponding section describing it in detail. These examples provide a simplified view of the way the Default code works.

You can refer to the Parallax BASIC Stamp Programming Manual for more information on PBASIC commands. Section 7 details the command that work within the Innovation First Control System structure.



2.1. Define the BS2-SX

The project file Definition line (below) must list the Processor Type being used. This Definition line is intentionally commented out. The Definition line is the only case in PBASIC where commented information is used by the compiler.

Commenting means that the text is for informational purposes to the reader only and is not processed by the microcontroller. Lines are commented by placing a single quote (‘) before the comment text.

Refer to the Multi-Bank Program Example in the White papers section of www.InnovationFirst.com for additional requirement for the Definition line when using Multi-Bank code.

Definition Line Example: The first line of the code below is completely commented (line starts with a single quote) and used for information purposes. The second line is the PBASIC Definition line, defining the BS2-SX as the target microcontroller for this code that will follow.

```
'---------- Define BS2-SX Project Files ----------------------------------------------------
' {$PBASIC 2.5}
' {$STAMP BS2SX}
```

2.2. Declare Variables

All variables used in the programs must be declared by name and type. The BS2-SX has 26 bytes of variable space available. See the Scratch Pad section (page 19) for additional memory space. The default program defines all the standard input and output variables. The unused variables in the Default Code are commented (unused). Just comment or un-comment the Default Code variables as needed. The names of the variables used in the default program can be changed; however, use caution since the name is used throughout the default code and must be changed at every instance. Declare any additional variables required.

Refer to the Multi-Bank Program Example in the White papers section of www.InnovationFirst.com for additional requirement for the Definition line when using Multi-Bank code.

Declare Variables Example: The first line of the code below is completely commented (line starts with a single quote) and used for information purposes. Each following line declares one variable. In this example the variables are all bytes. After each variable there are comments on the use of the variable in this application. This limited example only declares five (5) variables. The Default Code uses significantly more variables.

```
'========== DECLARE VARIABLES ==================================================================
p1_y            VAR byte            'Port 1, X-axis on Joystick
oi_swA          VAR byte            'OI Digital Switch Inputs 1 thru 8
rc_swA          VAR byte            'OI Digital Switch Inputs 1 thru 8
relayA          VAR byte            'RC Relay Output Byte, Relay Outputs 1 thru 4
Roller_Out      VAR byte            'Custom variable: the roller connected to PWM7
```

2.3. Define Aliases

Aliases provide alternate names for variables and sub-divisions of variables. Aliases don't require any additional memory. The default code uses Aliases to provide logical names for the Digital Inputs and Relay Outputs.

Define Aliases Example: Each line declares one variable. In this example the variables are all bits. After each variable there are comments on the use of the variable in this application. This limited example only declares three (3) variables. The Default Code uses significantly more variables.

```
'---------- Aliases for each RC switch input -------------------------------------------------
p1_sw_trig       VAR oi_swA.bit0       'Joystick Trigger Button,  same as Port4 pin5
rc_sw1           VAR rc_swA.bit0       'Robot Controller Switch 1 input
relay1_fwd       VAR RelayA.bit0       'Robot Controller Relay 1 Forward Command
```

2.4. Define Constants

Constants are fixed values with a name assigned to them. Constants do not require any memory or code space. Constants provide a convenient means to organize, locate, and edit fixed values in the code.

Define Constants Example: The Line below assigns one constant. The Default Code has several examples of constants.

```
'---------- Define Constants -----------------------------------------------------------------
rollerSpeed      CON 218               'PWM8 Roller Speed rc_sw1 is ON
```

2.5. Initialization

The Master micro-processor (uP) sends the data you select to the BS2SX PBASIC uP. The initialization portion of the code is used to select the input data. There is more data available than there is memory to store the data. You may select up to 26 constants, corresponding to 26 variables, from the 32 available to you. Make sure that you have variables for all the bytes received in the SERIN command (see details on page 8).

The constants below have a "c_" prefix, as compared to the variables that they will represent.
Step 1: Set the Constants below to 1 for each data byte you want to receive.
Step 2: Set the Constants below to 0 for the unneeded data bytes.
Step 3: Change the SERIN command to match the variables selected

Initialization Example: In this example, only p1_y, oi_swA, and rc_swA have been selected as input data. The code below lists all 32 data available, and the selected data ahs been commented. The Default Code uses significantly more Input Data.

```
'---------- Set the Initialization constants you want to read --------------------------------
c_p1_y           CON                   1            '←- Selected for Input
c_p2_y           CON                   0
c_p3_y           CON                   0
```

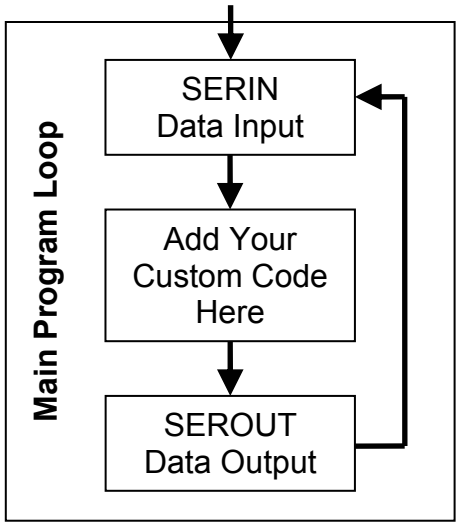
```
c_p4_y          CON             0
c_p1_x          CON             0
c_p2_x          CON             0
c_p3_x          CON             0
c_p4_x          CON             0
c_p1_wheel      CON             0
c_p2_wheel      CON             0
c_p3_wheel      CON             0
c_p4_wheel      CON             0
c_p1_aux        CON             0
c_p2_aux        CON             0
c_p3_aux        CON             0
c_p4_aux        CON             0
c_oi_swA        CON             1           '<- Selected for Input
c_oi_swB        CON             0
c_sensor1       CON             0
c_sensor2       CON             0
c_sensor3       CON             0
c_sensor4       CON             0
c_sensor5       CON             0
c_sensor6       CON             0
c_sensor7       CON             0
c_batt_volt     CON             0
c_rc_swA        CON             1           '<- Selected for Input
c_rc_swB        CON             0
c_delta_t       CON             0
c_PB_mode       CON             0
c_packet_num    CON             0
c_res01         CON             0
```

Caution: Do not change the names of these initialization constants. Do not change any other portion of the initialization code.

2.6. Main Loop

The core function of the PBASIC program is to run in a continuous loop that performs the following:

1. Read In the Input Data from the driver and on-board robot sensors
2. Perform specific functions based on the Input Data
3. Send Out the results to perform physical movement of the robots motors and valves.
4. Goto Step 1 (repeat).



This loop structure ensures that 1) new inputs (joysticks, etc.) are continuously read in, and 2) the outputs (motors, victors, spikes, etc.) continuously receive new commands. This loop repeats at a fixed maximum speed of 38.168Hz (38.168 loops per second). This corresponds to 26.2 mSec (0.0262 seconds). The actual loop speed may be 1, 2, 3, or 4 times longer if extensively long and/or slow code is written. Intermittent operation will result if the code takes longer than 5x to execute. See the Delta_T section on page 20 for information on checking your software speed.

2.7. Main Loop - SERIN Data Input

The first part of the Main Loop is the SERIN command. The SERIN command reads a serial stream of bytes from the Master Microprocessor.

Construct the SERIN command using the following rules:

1. There must be one variable for every input defined in the Initialization section.
2. The order must match the order in the example SERIN command below.
3. The total number of all variables may not exceed 26.
4. Only use one SERIN command.
5. The SERIN command must occupy only one line.

If you see a BASIC INIT Error on the Robot Controller after programming and pressing RESET, then there is a problem with the SERIN command line. Check the number of variables. A BASIC INIT Error will not occur if you have the variables in the wrong order, however your code will not work correctly.

SERIN Order Example: The SERIN line below defines the order that variables are read in. This sample line violates the 26 variable limit. This sample line also violates the "must be on one line" rule. This sample line is located in the Default Code for convenience.

```
Serin COMA\COMB, INBAUD, [oi_swA,oi_swB,rc_swA,rc_swB,p2_x,p1_x,p4_x,p3_x,PB_mode,packet_num,
sensor1,sensor2,p2_y,p1_y,sensor3,sensor4,p4_y,p3_y,sensor5,sensor6,p2_wheel,p1_wheel,
sensor7,sensor8,p4_wheel,p3_wheel,p2_aux,p1_aux,p4_aux,p3_aux,delta_t,res01]
```

Initialization Example: In the example below, only p1_y, oi_swA,and rc_swA have been selected in the initialization, therefore we must input those three variables in the same order as the SERIN Order Example above. The Default Code uses significantly more Input Data. The MainLoop: line marks the beginning of the Main Loop.

```
MainLoop:

'---------- Serin Command - Get Data from Master uP ----------------------------------------
Serin COMA\COMB, INBAUD, [oi_swA ,rc_swA, p1_y]
```

2.8. Main Loop - Perform Operations Here

This section of the code is where all the custom programming occurs. This is where you make your robot act the way you want, respond to your joystick and button commands, and react to the sensor inputs.

Perform Operations Example: The main loop below is just an example. Refer to the Default Code for a more complete example. The code below performs the following functions:
1. Port 1 Y axis linked to PWM1. There is no code needed for this. See the SEROUT line below.
2. Port 1 Trigger activates Relay 1 in the Forward Direction. See the PWM Outputs on page 13.
3. Robot sw1 automatically activates the roller forward at the speed set by constant rollerSpeed.

```
'========== PERFORM OPERATIONS ======================================================================

'---------- Button to Relay -----------------------------------------------------------------------
relay1_fwd = p1_sw_trig                 'Port 1 Trigger = Relay 1 Forward
                                        'relay1_fwd is an Alias to the relayA byte

'---------- Roller Code ---------------------------------------------------------------------------
if rc_sw1 = 1 then turn_roller_on:      'sw1 = 1 when activated, so goto turn_roller_on
       Roller_Out = 127                 'sw1 must be 0, so turn Roller_Out Off (127 is Off)
Goto exit_roller

turn_roller_on:
       Roller_Out = rollerSpeed
exit_roller:
```

2.9. Main Loop - SEROUT Data Output

The final part of the Main Loop is the SEROUT command. The SEROUT command sends a serial stream of bytes to the Output Microcontroller. The Output Microcontroller passes this to each PWM and Relay output. The Output Microcontroller will not output data if there is no communication with the Operator Interface or if the Competition Mode is Disabled. Do not delete any elements from the SEROUT command line. Set unused PWM outputs to 127. Set unused relay outputs to 0.

The last byte of the SEROUT command also passes a special Oscillator command to the Output Microcontroller for the EDU Robot Controller only (see the Oscillator Output section on page 14).

SEROUT Order Definition: The SEROUT line below defines the order that variables must be sent out. This sample line is located in the Default Code for convenience.

```
Serout USERCPU, OUTBAUD, [255,255,(PWM1),relayA,(PWM2),relayB,(PWM3),(PWM4),(PWM5),(PWM6),
(PWM7),(PWM8),(PWM9),(PWM10),(PWM11),(PWM12),(PWM13),(PWM14),(PWM15),(PWM16), (OSC)]
```

SEROUT Example: In the example below, we want to control PWM1, PWM7, and Relay 1(part of the RelayA byte). The variables p1_y, relayA, and Roller_out have been placed in the SEROUT line in the correct order. The Default Code uses significantly more Output Data. Notice the p1_y is in the PWM1 slot, relayA is in the first relay slot, and Roller_Out is in the PWM7 slot. The location of these variables is directly related to the robot wiring. In this case, the Roller is wired to a Speed Controller that is connected to PWM7.

```
'========== OUTPUT DATA ==============================================================================
Serout USERCPU, OUTBAUD, [255,255,p1_y,relayA,127,0,127,127,127,127,Roller_Out]
```

2.10.Main Loop – Return to SERIN

The Main Loop is now complete. The `Goto` line below sends the program back to just before the `MainLoop:` marker just before the SERIN line so it can start again.

Return to SERIN Example: This line must be located after the SEROUT command.

```
Goto MainLoop:
```

3. Program Inputs

The following sections describe the types of data input to the PBASIC program. The PBASIC program receives Input data from the Master Processor inside the Robot Controller. This Master Processor collects all the data from the Operator Interface via the RS-422 Radio Modem and all the Robot Controller sensor Inputs. The Master Processor then transfers the desired data to the PBASIC processor through a serial input (SERIN) command. See the Initialization section (page 6) for details on selecting the input data you want. Each piece of data is read into variables. See the SERIN section (page 8) for more details.

3.1. Digital Inputs

Digital Inputs are defined as inputs that are either ON or OFF. Buttons and switches are the most common form of digital inputs. In software a digital input is either a zero (0 = OFF) or a one (1 = ON). If a digital input is open or not connected, the digital input is OFF. If a digital input is connected is ground, the input is ON. Only connect digital inputs to ground and not to any positive voltage. The buttons on the joysticks are an example of digital inputs to the Operator Interface.

Operator Interface Digital Inputs

Refer to the Operator Interface Reference Manual for details on the Digital Inputs. The sixteen (16) Digital Inputs from the Operator Interface are grouped into two bytes, `OI_SWA` and `OI_SWB`. These input variables are aliased into more meaningful names such as `p1_sw_trig` which refers to the Joystick Trigger connected to Port 1.

Digital Inputs: The Operator Interface Digital Input variables are found in the Default Code.

```
'---------- Operator Interface - Digital Inputs ----------------------------------------------
oi_swA          VAR byte              'OI Digital Switch Inputs 1 thru 8
oi_swB          VAR byte              'OI Digital Switch Inputs 9 thru 16

'---------- Aliases for each OI switch input -------------------------------------------------
p1_sw_trig      VAR oi_swA.bit0       'Joystick Trigger Button, same as Port4 pin5
p1_sw_top       VAR oi_swA.bit1       'Joystick Top Button, same as Port4 pin8
p1_sw_aux1      VAR oi_swA.bit2       'Aux input, same as Port4 pin9
p1_sw_aux2      VAR oi_swA.bit3       'Aux input, same as Port4 pin15

p3_sw_trig      VAR oi_swA.bit4       'Joystick Trigger Button,  same as Port2 pin5
p3_sw_top       VAR oi_swA.bit5       'Joystick Top Button, same as Port2 pin8
p3_sw_aux1      VAR oi_swA.bit6       'Aux input, same as Port2 pin9
```

```
p3_sw_aux2       VAR oi_swA.bit7        'Aux input, same as Port2 pin15

p2_sw_trig       VAR oi_swB.bit0        'Joystick Trigger Button
p2_sw_top        VAR oi_swB.bit1        'Joystick Top Button
p2_sw_aux1       VAR oi_swB.bit2        'Aux input
p2_sw_aux2       VAR oi_swB.bit3        'Aux input

p4_sw_trig       VAR oi_swB.bit4        'Joystick Trigger Button
p4_sw_top        VAR oi_swB.bit5        'Joystick Top Button
p4_sw_aux1       VAR oi_swB.bit6        'Aux input
p4_sw_aux2       VAR oi_swB.bit7        'Aux input
```

Robot Controller Digital Inputs

Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the Digital Inputs. The Digital Inputs from the Robot Controller are grouped into two bytes, `RC_SWA` and `RC_SWB`. These input variables are aliased into more meaningful names such as `rc_sw2` which refers to the Switch 2 on the Robot Controller.

Digital Inputs: The Robot Controller Digital Input variables are found in the Default Code.

```
'---------- Robot Controller - Digital Inputs ------------------------------------------------
rc_swA           VAR byte               'RC Digital Inputs 1 thru 8
rc_swB           VAR byte               'RC Digital Inputs 9 thru 16 (full-size RC only)

'---------- Aliases for each RC switch input -------------------------------------------------
rc_sw1           VAR rc_swA.bit0
rc_sw2           VAR rc_swA.bit1
rc_sw3           VAR rc_swA.bit2
rc_sw4           VAR rc_swA.bit3
rc_sw5           VAR rc_swA.bit4
rc_sw6           VAR rc_swA.bit5
rc_sw7           VAR rc_swA.bit6
rc_sw8           VAR rc_swA.bit7
rc_sw9           VAR rc_swB.bit0        '(full-size RC only)
rc_sw10          VAR rc_swB.bit1        '(full-size RC only)
rc_sw11          VAR rc_swB.bit2        '(full-size RC only)
rc_sw12          VAR rc_swB.bit3        '(full-size RC only)
rc_sw13          VAR rc_swB.bit4        '(full-size RC only)
rc_sw14          VAR rc_swB.bit5        '(full-size RC only)
rc_sw15          VAR rc_swB.bit6        '(full-size RC only)
rc_sw16          VAR rc_swB.bit7        '(full-size RC only)
```

3.2. Analog Inputs

Analog Inputs are defined as inputs that are continuously changing, not just ON or OFF. Joystick axis are the most common form of analog inputs. Potentiometers and Gryo sensors are also Analog inputs. The CH FlightStick joystick has three analog inputs per joystick, the X axis, the Y axis, and the Wheel.

In software, an analog input is represented as a number from 0 to 254. The table below shows how an Analog Input varies as the joystick is moved. The Analog Inputs correspond directly to the Analog Outputs without modification (see Analog Outputs on page 13).

Joystick Function	Position and Analog Input Value		
Y Axis	Full Forward = 254	Neutral = 127	Full Back = 0 – 25*
X Axis	Full Left = 254	Neutral = 127	Full Right = 0 – 25*
Wheel	Full Forward = 254	Neutral = 127	Full Back = 0 – 25*

* Note: The joystick axis rarely go all the way to zero. This is normal. See the PWM section on page 13.

Operator Interface Analog Inputs

Refer to the Operator Interface Reference Manual for details on the Analog Inputs. The sixteen (16) Analog Inputs from the Operator Interface are each stored in bytes, such as `p1_x` and `p1_y` corresponding to the X and Y Joystick axis connected to Port 1.

Analog Inputs: The Operator Interface Digital Input variables are found in the Default Code.

```
'---------- Operator Interface (OI) - Analog Inputs ----------------------------------------
p1_x            VAR byte              'Port 1, X-axis on Joystick
p2_x            VAR byte              'Port 2, X-axis on Joystick
p3_x            VAR byte              'Port 3, X-axis on Joystick
p4_x            VAR byte              'Port 4, X-axis on Joystick

p1_y            VAR byte              'Port 1, Y-axis on Joystick
p2_y            VAR byte              'Port 2, Y-axis on Joystick
p3_y            VAR byte              'Port 3, Y-axis on Joystick
p4_y            VAR byte              'Port 4, Y-axis on Joystick

p1_wheel        VAR byte              'Port 1, Wheel on Joystick
p2_wheel        VAR byte              'Port 2, Wheel on Joystick
p3_wheel        VAR byte              'Port 3, Wheel on Joystick
p4_wheel        VAR byte              'Port 4, Wheel on Joystick

'p1_aux         VAR byte              'Port 1, Aux (not available on CH Joystick)
'p2_aux         VAR byte              'Port 2, Aux (not available on CH Joystick)
'p3_aux         VAR byte              'Port 3, Aux (not available on CH Joystick)
'p4_aux         VAR byte              'Port 4, Aux (not available on CH Joystick)
```

Robot Controller Analog Inputs

Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the Analog Inputs. The Analog Inputs from the Robot Controller are each stored in bytes, such as `sensor2` corresponding to pin 16 on the Robot Controller's Analog Input Port.

Analog Inputs: The Robot Controller Digital Input variables are found in the Default Code.

```
'---------- Robot Controller (RC) - Analog Inputs ------------------------------------------
'sensor1        VAR byte              'RC Analog Input 1, connector pin 2
'sensor2        VAR byte              'RC Analog Input 2, connector pin 16
'sensor3        VAR byte              'RC Analog Input 3, connector pin 5
'sensor4        VAR byte              'RC Analog Input 4, connector pin 19
'sensor5        VAR byte              'RC Analog Input 5, connector pin 8
'sensor6        VAR byte              'RC Analog Input 6, connector pin 22
'sensor7        VAR byte              'RC Analog Input 7, connector pin 11
'bat_volt       VAR byte              'RC Analog Input 8, hardwired to the Battery
```

4. Program Outputs

The following sections describe the types of data output from the PBASIC program. The PBASIC program sends Output data to the Output Processor inside the Robot Controller. This Output Processor collects all the data from the PBASIC Program and sends corresponding electrical commands to the PWM and RELAY ports. The PBASIC Program transfers the desired data to the Output Processor through a serial output (SEROUT) command. See the SEROUT section (page 9) for more details.

4.1. PWM Analog Outputs

The PWM Output Ports on the Robot Controllers are all controlled by PBASIC software. Each PWM output provides variable speed control for a motor in both Forward and Reverse.

PWM stands for Pulse Width Modulation and is the type of signal commonly used to control Speed Controllers such as the Victor 883. This PWM signal is the standard R/C type used on model airplane controls. This PWM is not the square wave type output by most microcontrollers. The signal is created by the Output processor in the Robot Controller. The PBASIC processor commands the Output processor to create the signal based on a byte (0-254) of data in the SEROUT command (SEROUT details on page 9).

The most common method of controlling a PWM output is via the joystick Analog Input. By sending a joystick Analog Input to a PWM output, a motor connected to the PWM port will act as follow:

Action	Function and Analog Value		
PWM Speed Controller	Full Forward = 254-227	Neutral = 136-123	Full Reverse = 0 – 37
Joystick Y Axis	Full Forward = 254	Neutral = 127	Full Back = 0 – 25
Joystick X Axis	Full Left = 254	Neutral = 127	Full Right = 0 – 25
Joystick Wheel	Full Forward = 254	Neutral = 127	Full Back = 0 – 25

Refer to the appropriate Robot Controller or Operator Interface Reference Manual for details on the number and location of the PWM Outputs.

Example (using the default software): Move the Y axis on the Port 1 Joystick forward to make a motor connected to PWM1 turn. The further you push the joystick, the faster the motor turns. From a neutral position move the Y axis on the Port 1 Joystick backwards to make a motor connected to PWM1 turn the other direction. Again, the further you push the joystick, the faster the motor turns.

4.2. Relay Digital Outputs

The Relay ports on the Robot Controllers are all controlled by PBASIC software. The Relay outputs provide ON or OFF control for motors and other devices. Each relay port has 2 digital outputs. These 2 digital outputs provide Full Forward, OFF, and Full Reverse control only. Refer to the appropriate Robot Controller or Operator Interface Reference Manual for details on the number and location of the Relay Outputs.

Example (using the default software): Press the Trigger button on the Port 1 Joystick to make a motor connected to RLY1 rotate. Press the Top button on the Port 1 Joystick to make a motor connected to RLY1 rotate in the opposite direction.

4.3. Oscillator Output (EDU-RC only)

The oscillator output is set by the default code to output a 40 KHz signal at about a 50% duty cycle. This frequency can be controlled by the 17th PWM byte in the SEROUT command in the PBASIC code. As the 17th byte varies from 2 – 254, the OSC Output frequency will vary from 19.6 KHz to 1.66 MHz. The 17th byte only needs to be sent out 1 time to set the frequency. The 17th byte does not need to be sent out again until the frequency needs to be changed.

The formula for the frequency is:

1/((PWM value + 1) * 4 * 0.00000005) Note: Valid PWM range is 2 – 254.

5. Downloading to the Robot Controller

5.1. Programming Steps

The Robot Controller programming steps are as follows:

1. Power ON the Robot Controller.
2. Connect a DB9 Male-to-Female Pin-to-Pin cable (maximum length 6 ft.) from the PROGRAM port on the Robot Controller to PC's serial port.
3. Set the Program Jumper to USER or DEF as desired (full-size RC only) (details below)
4. Be sure the Write Protect Jumper is installed (EDU-RC only) (details below)
5. Run Parallax, Inc. BASIC Stamp Version 1.33 or higher.
6. Open the desired program with the BASIC Stamp program.
7. Press CTRL-R to download the program.
8. Some older full-size Robot Controllers require the user to press RESET after downloading a program.

Note: Before the PBASIC program will start executing, the Robot Controller must have a Valid RX (link) to an OI or the RC must be in Autonomous Mode.

5.2. Write Protect Jumper (EDU-RC only)

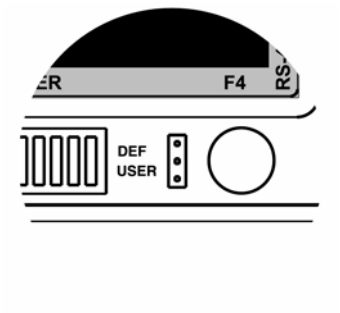
The Write Protect jumper on the Isaac16 EDU Motherboard is used to allow or disallow writing to the PBASIC program memory chip. The PBASIC program memory chip is located on the bottom side of the Motherboard. The full-size Robot Controller does not have a user accessible write protect jumper.

- The PBASIC program can be changed when the jumper is installed (Write Protect OFF).
- The PBASIC program **cannot** be changed when the jumper is **not** installed (Write Protect ON).

5.3. DEF/USER Jumper (full-size RC only)

The full-size Robot Controller has two memory chips, USER and DEFAULT. A Program Jumper on the Robot Controller is used to select which control program (default or user) is running. Both program chips contain the Default Code when shipped from the factory. The user can change the program in one or both memory chips. When starting out, it is best to keep the Default Code in the DEFAULT location to aid in troubleshooting.

- Install a jumper across the upper two pins to run and/or download to the DEFAULT program.
- Install a jumper across the lower two pins to run and/or download to the USER program.
- After changing the jumper, press RESET or Power Cycle the Robot Controller.



5.4. Basic Run (Green LED)

After programming the Robot Controller, after pressing RESET on the Robot Controller if needed, and after the Robot Controller is communicating with an Operator Interface, you should see the BASIC RUN LED flashing. This BASIC RUN LED indicates that the new program is running. The PBASIC program has controls of this light, and therefore, may not blink if the programmer changes that portion of the code.

5.5. Basic Run Error (Red LED)

If after programming and resetting the Robot Controller, the BASIC RUN ERR LED is ON, then the basic code has no output. This means that the code is not running properly. Check for errors in the code. The most common problems are “infinite loops” or “pause” commands. The BASIC RUN ERR light is controlled by the Output microprocessor. Debug statements (page 16) and PB_Mode (page 18) may be helpful in locating the problem.

5.6. Basic Init Error (Red LED)

If after programming and resetting the Robot Controller, the BASIC INIT ERR LED is ON, then the basic code did not properly initialize the data packet structure with the master microprocessor. Check the initialization part of the code for errors. A common mistake is having a different number of variables in the SERIN command, as compared to the requested data setup in the “Set the Initialization

constants you want to read" section of the code. The BASIC INIT ERR light is controlled by the Master microprocessor.

6. Additional PBASIC techniques

6.1. Debug Statements

Debug provides a convenient way for your PBASIC program to send messages to the PC screen while running. The name "debug" suggests its most popular use; debugging programs by showing you the value of a variable or expression, or by indicating what portion of a program is currently executing.

Debug will display information on the PC screen within the BASIC Stamp editor program. This command can be used to display text or numbers in various formats on the PC screen in order to follow program flow (called debugging). If you close your debug window, quit and reboot the BASIC Stamp Editor.

There are many more ways to use the Debug command. Reference the BASIC Stamp Programming Manual 2.0c for more information. Below are a few examples to get you started.

Syntax: **DEBUG** *OutputData {,OutputData}*

OutputData is a variable/constant/expression (0 - 65535) that specifies the information to output. Valid data can be ASCII characters (text strings and control characters), decimal numbers (0 - 65535), hexadecimal numbers ($0000 - $FFFF) or binary numbers (up to %1111111111111111).

Debug Example #1: The following example demonstrates using the debug command to send the text string message "Innovation First!".

```
'---------- Somewhere in the Main Loop ----------------------------------------------------
debug "Innovation First!",CR                  'Test Message
```

After you add this one line somewhere in the main loop, and after downloading, the BASIC Stamp Editor will open a Debug Terminal on your PC screen and wait for a response from the BASIC Stamp. A moment later, the phrase " Innovation First!" will appear. The CR symbol will cause the Debug Terminal to start a new line (carriage return). The line will repeat every loop of the code. Note that if you close the Debug Terminal, your program keeps executing, but you can't see the debug data anymore.

Debug Example #2: Multiple pieces of data can be sent with one debug command by separating the data with commas (,). The following example produces exactly the same results as the example above.

```
'---------- Somewhere in the Main Loop ----------------------------------------------------
debug "Innovation ", "First!", cr             'Test Message
```

After you add this one line somewhere in the main loop, and after downloading, the BASIC Stamp Editor will open a Debug Terminal on your PC screen and wait for a response from the BASIC Stamp. A moment later, the phrase " Innovation First!" will appear. The CR symbol will cause the Debug Terminal to start a new line (carriage return). The line will repeat every loop of the code. Note that if you close the Debug Terminal, your program keeps executing, but you can't see the debug data anymore.

Debug Example #3: Debug can also print and format numbers (values) from both constants and variables. The (?) formatter always displays data in the form "symbol = value" (followed by a carriage return). In addition, it defaults to displaying in decimal. It's important to note that the "symbol" it displays is taken directly from what appears to the right of the "?".

```
'---------- Somewhere in the Main Loop -------------------------------------------------
Debug  ?p1_y                                  'display Port 1 "Y" variable
```

The above example will display the port 1 "Y" axis variable in the debug window, when using Default Code, in the format "p1_y = 127". Move the Port 1 joystick and the value on the screen will change.

6.2. Robot Feedback and User Mode

The Robot Controller has the ability to send data back to the Operator interface. The Robot Controller can send one byte (8 bits) every loop of the code. The data is available on the Operator Interface in three ways: 1) On the eight Robot Feedback LEDs, 2) on the multi-segment display when in user mode, and 3) on a PC connected to the Dashboard Port. See the PB_Mode section (page 18) for writing code that can display numbers on the multi-segment display.

The Robot Feedback byte is passed from the PBASIC microcontroller to the Master microcontroller eight bits at a time. The Master microcontroller then sends the data to the Operator interface via the Radio Modems. You change the Feedback bits one at a time.

Robot Feedback Example: The Default Code assigns the eight output pins on the PBASIC microcontroller as outputs (shown below). The user can change these bits directly (also shown below).

```
'---------- Input & Output Declarations -------------------------------------
Output      8 'define Robot Feedback LED => out8      => PWM1 Green
Output      9 'define Robot Feedback LED => out9      => PWM1 Red
Output      10      'define Robot Feedback LED => out10     => PWM2 Green
Output      11      'define Robot Feedback LED => out11 => PWM2 Red
Output      12      'define Robot Feedback LED => out12 => Relay1 Red
Output      13      'define Robot Feedback LED => out13 => Relay1 Green
Output      14      'define Robot Feedback LED => out14 => Relay2 Red
Output      15      'define Robot Feedback LED => out15 => Relay2 Green


'========== PERFORM OPERATIONS section ==========================================================
Out13 = relay1_fwd         'The Relay1 Green LED is ON when Relay 1 is Forward (1)
out11 = my_Bit_Variable    'The PWM2 Red LED is ON when my_Bit_Variable is 1
```

6.3. PB_Mode – Competition, Autonomous, and User

PB_Mode is a byte accessible through the SERIN command. The Competition Mode, Autonomous Mode, and User Mode are all available in this byte. The Default Code already contains all the code needed to use these modes.

Competition Mode
Bit 7 of the PB_mode byte (aliased as comp_mode below) indicates the status of the Competition Control, either Enabled or Disabled. This indicates the starting and stopping of rounds at the competitions. Comp_mode is indicated by a solid "Disabled" LED on the Operator Interface.
Comp_mode = 1 for Enabled, 0 for Disabled.

Autonomous Mode
Bit 6 of the PB_mode byte (aliased as auton_mode below) indicates the status of the Autonomous Mode, either Autonomous or Normal. This indicates when the robot must run on its own programming. When in Autonomous Mode, all OI analog inputs are set to 127 and all OI switch inputs are set to 0 (zero). Auton_mode is indicated by a blinking "Disabled" LED on the Operator Interface.
Auton_mode = 1 for Autonomous, 0 for Normal.

Autonomous Mode can be turned ON by setting the RC to Team 0 (zero) and power cycling the RC unit.

User Mode
Bit 5 of the PB_mode byte (aliased as user_display_mode below) indicates when the user selects the "User Mode" on the OI. PB_mode.bit5 is set to 1 in "User Mode". When the user selects channel, team number, or voltage, PB_mode.bit5 is set to 0. When in "User Mode", the eight Robot Feedback LED are turned OFF and the Out 8 – Out 15 of the PBACIS processor are sent to the OI multi-segment display.

Note: "User Mode" is identified by the letter “u” in the left digit (for 4 digit OI's)
Note: "User Mode" is identified by decimal places on the right two digits (for 3 digit OI's)
The Default Code sends Port 1 Y-Axis to the display when in User Mode.

Autonomous _Mode Example: This example shows portions of the code that reference PB_Mode. The Perform Operations Section below uses aliases based on PB_Mode bits to execute different sections of code.

```
'---------- Declare Miscellaneous Variables -------------------------------------------------
PB_mode      VAR byte

'---------- Aliases for the Pbasic Mode Byte (PB_mode) ---------------------------------------
comp_mode           VAR PB_mode.bit7
auton_mode          VAR PB_mode.bit6
user_display_mode   VAR PB_mode.bit5


'========== PERFORM OPERATIONS section ============================================================
if auton_mode = 1 then do_autonomous_stuff
      'Insert your normal driver mode code here
goto end_of_autonomous

do_autonomous_stuff:
      'Insert your autonomous mode code here
end_of_autonomous:
```

User Mode Example: This example shows the Default Code that changes the data sent to the Operator Interface when the user sets the Operator Interface to User Mode.

```
'---------- Feedback LEDs -------------------------------------
if user_display_mode = 1 then user_mode
       Out8  =   p1_y/216         'LED is ON when Victor883 full forward (default CAL)
       Out9  = ~(p1_y/56  max 1)  'LED is ON when Victor883 full reverse (default CAL)
       Out10 =   p2_y/216         'LED is ON when Victor883 full forward (default CAL)
       Out11 = ~(p2_y/56  max 1)  'LED is ON when Victor883 full reverse (default CAL)
       Out13 = relay1_fwd         'LED is ON when Relay 1 is Forward
       Out12 = relay1_rev         'LED is ON when Relay 1 is Reverse
       Out15 = relay2_fwd         'LED is ON when Relay 2 is Forward
       Out14 = relay2_rev         'LED is ON when Relay 2 is Reverse
goto display_done

user_mode:                        'Send p1_y to multi-segment display when in User mode
       out8  = p1_y.bit0
       out9  = p1_y.bit1
       out10 = p1_y.bit2
       out11 = p1_y.bit3
       out12 = p1_y.bit4
       out13 = p1_y.bit5
       out14 = p1_y.bit6
       out15 = p1_y.bit7
display_done:
```

6.4. Scratch Pad

Scratch Pad RAM is useful for additional workspace and for passing data to programs in other program slots. It is different than regular RAM in that symbol names cannot be assigned directly to locations and each location is always configured as a byte only. The Scratch Pad RAM is accessed using the PUT and GET command.

Scratch Pad RAM locations are from 0 – 63 and most are available for general use. The highest location, 63 is a special, read-only, location that always contains the number of the currently running program slot. The Scratch Pad is organized as bytes only.

Scratch Pad Example: The following code defines a Variable Byte called temp, writes the value 100 to location 25 with PUT, reads it back out with GET, stores it in a variable called temp and finally displays it with a Debug statement.

```
'---------- Declare Miscellaneous Variables -------------------------------------------------
temp                VAR    Byte

'---------- Define Constants ----------------------------------------------------------------
my_eeprom_data = 25

'========== PERFORM OPERATIONS section ========================================================
Main:
       PUT my_eeprom_data, 100
       GET my_eeprom_data, temp
       DEBUG ?temp
       END
```

6.5. Delta_T

The Delta_T variable in the Default Code can be used to determine if the user/custom code you have written is running slowly. Your User program should be updated with new packets of data every time the Main Loop executes, about every 26 ms. If your custom program has too many lines of code that must be executed for each loop, you will begin to miss every other new packet of data (or worse). If you miss more than 5 packets of data in a row, the BASIC RUN ERR LED will be illuminated on the Robot Controller. The Delta_t variable will indicate if your code is missing packets of data. Missing every other packet is acceptable, but you will begin to notice "sluggish" control if you miss several packets.

Do not use the Debug command to monitor this variable because the added execution time to perform the Debug command itself, can cause the Detla_t value to show missed packets that would not have otherwise occured. The Delta_t value will equal 0 when there are no missed data packets. When data packets are being missed, Delta-t will increment and show the total number of data packets that are being missed. The best way to look at the Delta_t value is using the USER MODE on the Operator Interface displays.

Delta_t Example: The corresponding changes to the Default Code must be made to enable Delta_t to be displayed on the Operator Interface. Also, the Operator Interface must be in User Mode.

```
'---------- Declare Miscellaneous Variables -------------------------------------------------
delta_t          VAR byte


'---------- Set the Initialization constants you want to read -------------------------------
c_delta_t        CON  1


MainLoop:
'---------- Serin Command - Get Data from Master uP -----------------------------------------
Serin COMA\COMB, INBAUD, [ . . . ,delta_t, . . .]


'========== PERFORM OPERATIONS section ================================================

'---------- Feedback LEDs -------------------------------------
if user_display_mode = 1 then user_mode
      (normal display code here)
goto display_done

user_mode:                          'Send delta_t to multi-segment display when in User mode
      out8  = delta_t.bit0          'Add these lines of code just before the OUTPUT DATA
      out9  = delta_t.bit1          'section of code.
      out10 = delta_t.bit2
      out11 = delta_t.bit3
      out12 = delta_t.bit4
      out13 = detla_t.bit5
      out14 = delta_t.bit6
      out15 = delta_t.bit7
display_done:
```

6.6. Multiple Program Banks

See the example code at www.InnovationFirst.com.

7. PBASIC Commands

Not all PBASIC commands can be used with the Innovation First Control System. The valid PBASIC Commands for the Innovation First Control System are shown below. In addition to these commands, all the Unary Operators and Binary Operators will work with the Innovation First Control Systems. These include the basic logic operations, i.e. ADD (+), SUBSTRACT (-), MULTIPLY (*), etc. Reference the BASIC Stamp Programming Manual 2.0c for more information.

BRANCHING / PROGRAM CONTROL

IF …THEN	IF Condition THEN Address
BRANCH	BRANCH Offset,[Address1, Address2, ...AddressN]
GOTO	GOTO Address
GOSUB	GOSUB Address
RETURN	RETURN
RUN	RUN ProgramSlot

IF-THEN-ELSE	IF Condition THEN
	Statement(s)
	ELSE
	Statement(s)
	ENDIF

DO-LOOP	DO Statement(s) LOOP
SELECT-CASE	SELECT Expression CASE Condition Statement(s) CASE Condition Statement(s) CASE ELSE Condition Statement(s) ENDSELECT

LOOPING

FOR…NEXT	FOR Counter = StartValue TO EndValue {STEP StepValue} ... NEXT

EEPROM DATA

DATA	{Symbol} DATA DataItem {, DataItem, ...}
READ	READ Location, Variable
WRITE	WRITE Location, DateItem **CAUTION:** Do Not WRITE to EEPROM in every loop of your code. This will cause the EEPROM to fail. The EEPROM cycle life is only about 100K.

RAM ACCESS

GET	GET Location, Variable
PUT	PUT Location, Value

NUMERICS

LOOKUP	LOOKUP Index, [Value0, Value1, ...ValueN], Variable
LOOKDOWN	LOOKDOWN Target, {ComparisonOp} [Value0, Value1, ...ValueN], Variable
RANDOM	RANDOM Variable

DIGITAL I/O

INPUT	INPUT Pin
OUTPUT	OUTPUT Pin
LOW	LOW Pin

	HIGH	HIGH Pin
	TOGGLE	TOGGLE Pin

ASYNCHRONOUS I/0

	SERIN	SERIN Rpin {\Fpin},Baudmode,{Plabel,}{Timeout, Tlabel,} [InputData]
	SEROUT	SEROUT Tpin {\Fpin},Baudmode,{Pace,} {Timeout, Tlabel,} [OutputData]

TIME

	PAUSE	PAUSE Period **CAUTION:** A PAUSE that creates a long delay can cause the robots movement to become intermittent.

PROGRAM DEBUGGING

	DEBUG	DEBUG (OutputData {, OutputData})