599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com

Stamp PLC (#30064)

BASIC Stamp Programmable Logic Controller Package

Introduction

The Stamp PLC is a Program Logic Controller that is perfectly sized for automating small machines. Specified by Parallax, Inc., the Stamp PLC was designed by Lawicel Soft-&Hard of Sweden. Parallax and Lawicel have combined their expertise to deliver a PLC that represents the next stage of evolution in the small-scale PLC market.

PLCs are microcontrollers that are "pre-packaged" to withstand the hazards of an industrial environment. Specifically, their inputs and outputs are optically isolated, the outputs are fully protected, and their internal components are electrically tough and rather immune to noise typically present in industrial environments. Furthermore, the Stamp PLC is housed by a strong and sleek enclosure that offers an integral DIN rail mount.

Packing List

Verify that your Stamp PLC kit is complete in accordance with the list below:

- Stamp PLC hardware
- Serial Cable
- Documentation
- Small bag with four shunts inside

Optionally, you can purchase the MAX1270 A/D Converter 12-bit, 8 channel 4-20 mA A/D converter. This is available from Parallax web site http://www.parallax.com/detail.asp?product_id=604-00026. You will also need to provide a power supply (see below in the Power Supply and Connection section).

Demonstration and example software files used in this documentation may be downloaded from http://www.parallax.com/detail.asp?product_id=30064.

Features

- 10 Digital Inputs. Eight of these inputs are grouped together courtesy of an on-board shift register. BASIC Stamps have built-in commands to read these with ease. The remaining two inputs are read directly by the BASIC Stamp. All inputs are optically isolated.
- 8 Digital Outputs are optically isolated, electrically and thermally protected.
- 4 Analog Inputs (optional). Installing an optional A/D converter into its socket adds four analog input channels. Each channel can be independently configured as 4-20mA, 0-5 VDC, -5 to +5 VDC, -10 to +10 VDC, and has 12-bits of resolution.
- Front Panel LEDs indicate the status of all ten inputs and all eight outputs via a light-pipe array.
- Heavy-duty power supply has built-in noise protection.

- RS-232 Serial Port Once programming is completed, the on-board serial port can be used to send and receive serial data[1].
- BASIC Stamp socket accommodates any 24-pin BASIC Stamp. This means that this PLC can do virtually anything that a BASIC Stamp can perform. Logic functions, numerical computation, conditional branching, even non-volatile memory for data-logging.

Additional Features

- No proprietary software to buy; you may always download the latest BASIC Stamp editor software free of charge from www.parallax.com
- PBASIC Language: this language was designed to be straightforward and easy to use without a compiler. The PBASIC language can be learned by anyone, and can be mastered within a few of hours.
- Free Tech-Support: Call or email us with your questions.
- Resources: The BASIC Stamp has thrived for more than 13 years, and as a result, there are many, many resource websites and books dedicated to Stamps and the application thereof.
- Built-In Debugger: Connect the serial cable to a PC running the software and you can use the debugger to help you debug your program and/or report data.

Getting Started

To begin using your Stamp PLC, simply follow the instructions in the next few sections. Be sure that the Stamp PLC is de-energized while the unit is apart and for all steps requiring you to connect or disconnect wires, etc. After the connections are made, the following sections walk you through basic programming techniques used to access the Stamp PLC's inputs and outputs.

Opening the Enclosure

At first, this enclosure may seem to be a bit of an enigma. A closer examination of the bottom of the enclosure will reveal three small slots near the corners. By inserting a small, flat-bladed screwdriver into these slots, each catch in turn can be positioned to allow the two halves of the enclosure to be separated. As the third catch is released, watch for the small spring and the red latch as they will be loose and can easily become lost.

Figure 1: Opening the Stamp PLC Enclosure

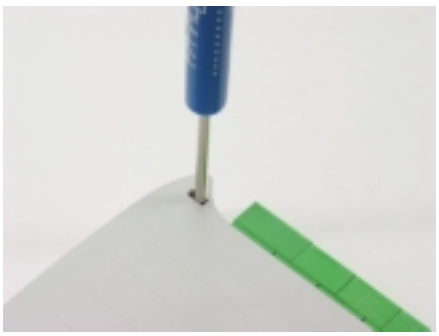
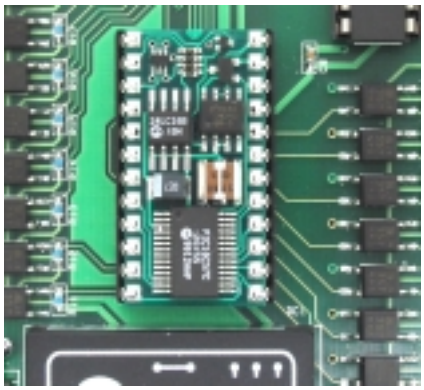


Figure 2: BASIC Stamp Plugged into the Stamp PLC

[1] The Javelin Stamp's programming port currently has limited run-time functionality. For a work-around solution, please contact Parallax Tech Support (916) 624-8333.



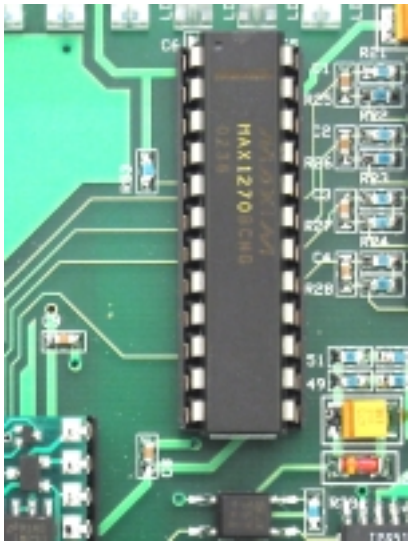
Installing the BASIC Stamp

Once the cover is removed, install the BASIC Stamp Module (sold separately) into the 24-pin IC socket provided observing the proper orientation. Be sure to get each pin into its respective socket and to not bend pins over. Please refer to Figure 2 ensure you have installed the BASIC Stamp correctly.

Installing the MAX1270 A/D Converter Chip

If you have purchased the optional MAX1270 A/D Converter, now is the best time to install it. Locate the 28-pin IC socket provide and install the MAX1270 observing the proper orientation. Please refer to the photo on the right to ensure you have installed the MAX1270 correctly.

Figure 3: Installing the Max1270 A/D

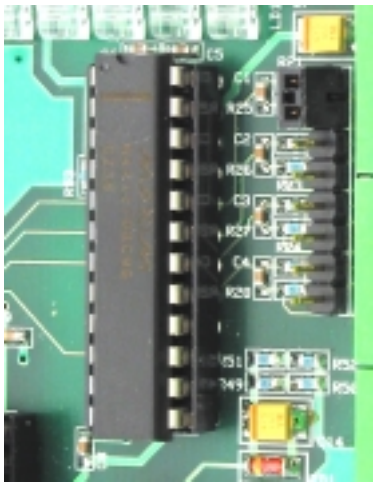


Once the BASIC Stamp IC and the optional A/D converter are properly installed re-assemble the enclosure. Be sure that the spring, red latch, and the front cover are properly aligned prior to snapping the two halves of the enclosure together.

Installing the 4-20mA Shunts

For each analog signal you wish to read that is a 4-20mA current loop you will need to install a shunt at this time. The shunt locations correspond to the analog channels they are adjacent to. Figure 4 shows the locations of the shunts relative to the MAX1270 A/D converter IC. It also shows the shunt for analog input channel 1 installed properly.

Figure 4: Oblique view of the 4-20mA Shunt Locations



Programming Port Connection

Connections Sout, Sin, ATN, and GND are for the serial port. This diagram depicts exactly how to wire these connections to a standard DB9 PC serial port. Please note the jumper wire connecting pin-6 to pin-7 on the DB9 is not necessary providing that you use BASIC Stamp Editor software version 1.2 (or later).

Figure 5: Stamp PLC with Serial Port and Power Connected

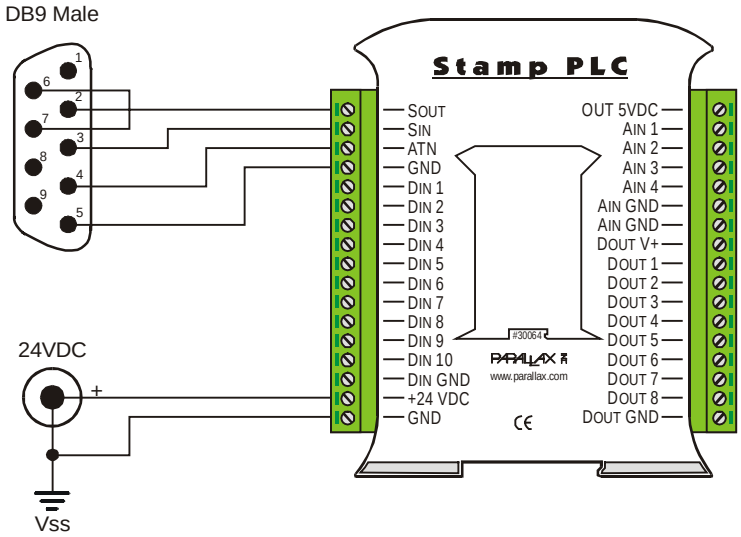
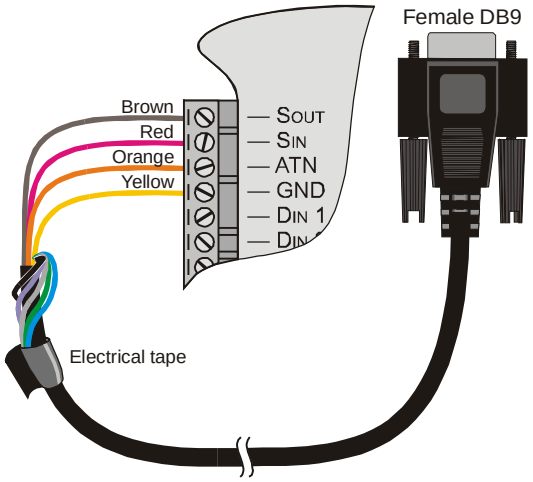


Figure 6: Serial Port Cable Detail



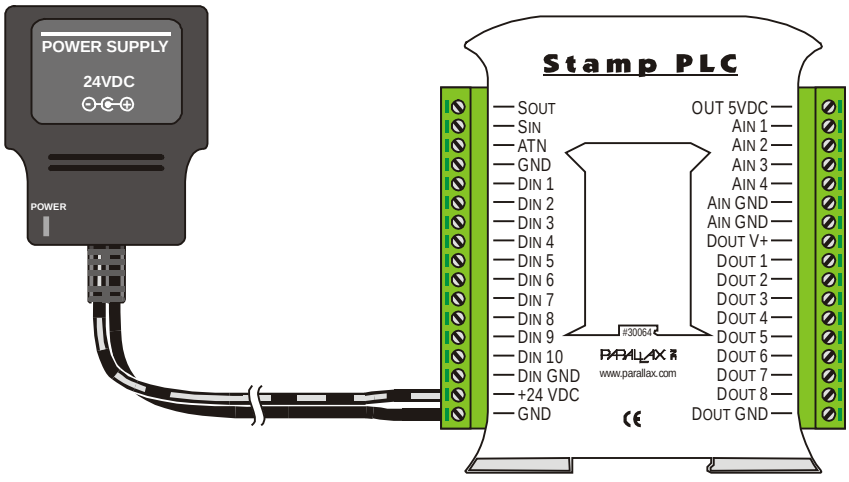
Power Supply and Connection

The Stamp PLC was designed to be powered by a 24 VDC supply. If you are simply running the Stamp PLC on the bench-top, you can use any 24VDC @ >=300mA supply. Parallax offers a small 24VDC @ 600mA supply, (**Order #750-00004**), suitable for bench-top experimentation with the Stamp PLC. If you are installing the Stamp PLC for use in the field, be sure to properly size your 24VDC supply to match the entire load, including digital outputs. The small 24 VDC supply offered by Parallax is not suitable for most field installations as it does not supply sufficient current to power most output devices.

Do not use an AC supply, doing so will damage the Stamp PLC and void the warranty.

To install the power supply issued by Parallax, remove it from its packaging and cut off the 2.1mm power connector. Be sure that the cut is very close to the power connector, and is not close to the power supply. The striped wire is the positive lead. Connect the positive lead of your power supply to the +24 VDC pin. Connect the negative lead to the adjacent terminal labeled GND. When completed, your power supply connection should resemble the image in **Figure 7**.

Figure 7: Power Supply, connected to Stamp PLC.



If you are using a different supply, follow these instructions: ensure the power supply is off, connect the positive lead of your power supply to the +24 VDC pin, connect the negative lead to the adjacent terminal labeled GND.

Test Program

Once the Editor has been installed, and you have the power and serial connections made, apply power and launch the Editor. Hit Ctrl-I to identify the BASIC Stamp. If all is well, the software will be able to locate which port the Stamp PLC is connected to and identify which type of BASIC Stamp is inside. If you have trouble getting the Editor to "find" the Stamp PLC, either email or call our technical support for help. The email address is: support@parallax.com. Our phone number is: (916) 624-8333.

Program the BASIC Stamp[2] within the Stamp PLC. Enter the following code in the Editor:

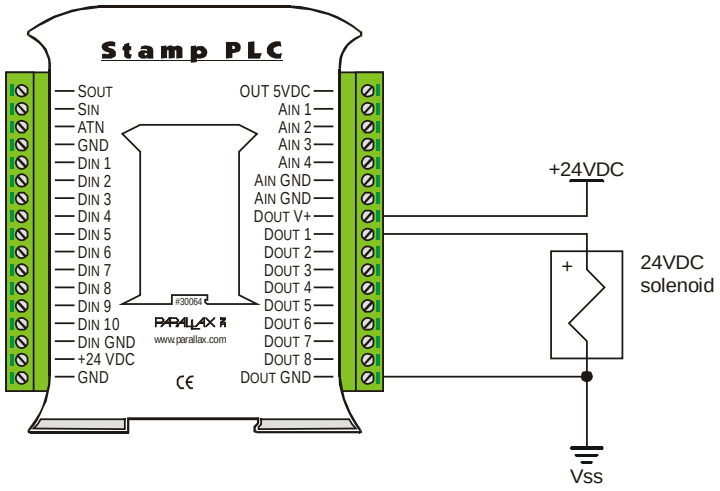
```
'{$STAMP BS2}              ' Stamp type directive
Main:                              ' Marks the start of the main program
  DEBUG "Hello World!"   ' Send "Hello World!"  to the debug window
```

Once the code above is entered, simply click on the run button, or hit Ctrl-R, or click on Run->Run, any of these will cause the software to compile, download, and run the program. Upon completing the download, a debug window will open automatically and the words "Hello World!" will appear. If you like, close the debug window, change the text within the quotes in the debug command, and run the new program. Doing this a few times can really help you get the hang of it.

Writing to Digital Outputs

Safety Notice! The digital outputs are (essentially) tied directly to the BASIC Stamp I/O pins P8 through P15. This is necessary for safety reasons. If anything "goes wrong" i.e. power interruptions, spurious resets, program errors, etc., the BASIC Stamp automatically reverts the outputs to inputs for a period of 18mS before the BASIC Stamp program can restart and take control of the outputs again. This causes the outputs to cease driving, and thereby stops motors and other output devices. When designing your system and programming it, you MUST design it to fail to a safe condition.

Figure 8: Typical Output Connection



The digital outputs are tied to the BASIC Stamp I/O pins P8 through P15 via electrically protected, high-side drivers[3]. This makes it very easy to write to the digital outputs. The following mini-program configures P14 as an output and drives it low (turns it on).

[2] This document assumes that you are using the BASIC Stamp 2, although you may use any of the 24-pin BASIC Stamps offered by Parallax, Inc. Be aware that other types of stamps may run at different speeds. In these cases, time-critical operations will be affected and will need to have their parameters adjusted.

[3] Note: The Stamp PLC uses the IPs512G for its high current driver. To find out more about the IPs512G r please see 1st two pages of the data sheet we attached to this manual.

```
'{$STAMP BS2}             ' Stamp type directive
Main:
  LOW 14                  ' Make P14 and output and drive it low (turn on)
```

Please carefully observe the behavior of the LED on the front of the Stamp PLC. You should see it pulse off for a fraction of a second (18mS), then remain on steadily for ~2.3 seconds. This behavior is normal, considering the fact that our program is incomplete.

Every BASIC Stamp has a built-in watchdog timer. A watchdog timer monitors the program's behavior and always counts down. If the watchdog timer ever expires, it automatically resets the Stamp and the user program (your basic program) starts over from the beginning. Each time a valid Stamp instruction is completed, the watchdog timer is reset, thereby staving off a watchdog reset. Why is the reset occurring in the example above? The reason is that we have not specified what the Stamp is to do *after* completing the "LOW 14" command. Essentially, the imaginary pointer that points to "the next instruction to do" has been allowed to "fall off into the weeds". To make this program complete and to keep our program pointer on the path, it is necessary to contain Stamp programs within a loop.

```
'{$STAMP BS2}             ' Stamp type directive
'{$PBASIC 2.5}            ' Stamp expanded syntax directive
Main:
  LOW 14                  ' Make P14 and output and drive it low (turn on)
  DO : LOOP               ' Wait here until hard reset
```

Amending your program to reflect the changes made above should rectify the watchdog reset problem. In other words, the LED representing P14 will remain on steadily. It is important to understand the nature of a watchdog reset. Depending on your program, a logical error in your program could cause a watchdog reset and you may not notice it if you don't know what to look for. Please be certain that your program is functioning exactly as designed, (i.e. no resets), *before* placing it in service.

The PBASIC syntax offers many features that, if used, can greatly enhance the readability of your Stamp code. It is desirable to make your program as readable as possible. Doing so will make bugs easier to find, and perhaps three years from now, when a customer requests a function added to your program and you've forgotten how it works, a brief read through your program and it's comments reveal just how everything works. One way to enhance the readability of your program is to use descriptive aliases and comments where possible.

```
'{$STAMP BS2}             ' Stamp type directive
'{$PBASIC 2.5}            ' Stamp expanded syntax directive

Pump   PIN    14          ' P14 is connected to the pump output

Main:
  DO
    LOW  Pump             ' Turn on the Pump
    PAUSE 1000            ' let it run for 1 second
    HIGH Pump             ' Turn off the Pump
    PAUSE 10000           ' Wait for 10 seconds
  LOOP                    ' then repeat
```

As you can see, the program has been modified to switch on the pump every ten seconds for duration of one second.

The output drivers used by the Stamp PLC are fully protected high-side drivers. This means that if too much current is drawn, or if they get too hot, they shutdown automatically. Additionally, once they cool

down they will automatically start up. The BASIC Stamp does not know if this happens, so again, devote a significant amount of time to think about how to make your program and how the Stamp PLC is connected absolutely fail-safe. The diagram to the right depicts how to connect an output device, and the output power supply, to the Stamp PLC.

Reading Digital Inputs

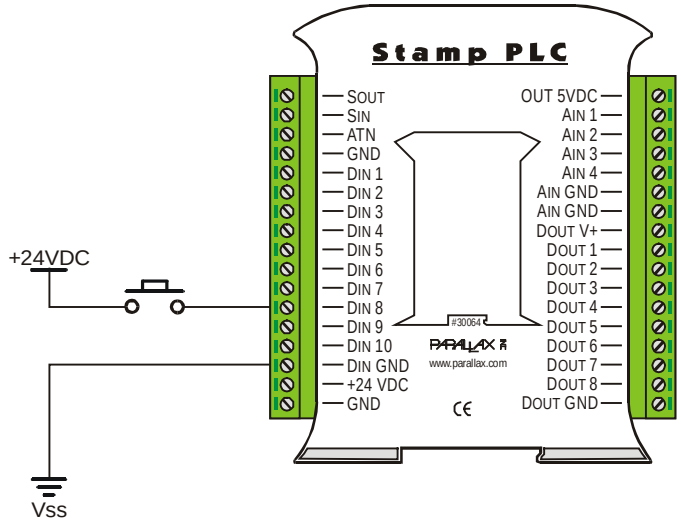
Digital inputs 1-8 are multiplexed (or grouped) into three I/O lines courtesy of a 74HC165 parallel in - serial out shift register. All Stamps have a built-in command (SHIFTIN) to assist in reading this type of device. When the 74HC165 is read, the status of the inputs will be loaded into the corresponding bits of the variable InBits. The following program will retrieve and display digital inputs 1-8.

```
'{$STAMP BS2}                ' Stamp type directive
'{$PBASIC 2.5}               ' Stamp expanded syntax directive

inBits   VAR     Byte        ' inBits contains inputs 1-8
Clk      PIN     0
Load     PIN     1
Dat      PIN     2

Main:
  DO
    HIGH Load                ' Get 8 inputs from shift-register
    SHIFTIN Dat,Clk, MSBPRE, [inBits]
    LOW Load                 ' Display 8 inputs in a binary fashion
    DEBUG "Digital Inputs:", IBIN8 inBits, CR
    PAUSE 1000               ' Wait for 1 second
  LOOP                       ' then repeat
```

Figure 8: Typical Input Connection



Inputs 9-10 are accessed directly. They are not part of the shift register, and as a result, have their own addresses. Input 9's I/O address is 6, and input 10's I/O address is 7. The following program listing shows how to access Inputs 9 and 10, and does so using conditional branching statements.

```
'{$STAMP BS2}                ' Stamp type directive
'{$PBASIC 2.5}               ' Stamp expanded syntax directive
```

```
Main:
  DO
    IF IN6 = 0 THEN DEBUG "Input  9 Pressed!",CR
    IF IN7 = 0 THEN DEBUG "Input 10 Pressed!",CR
    PAUSE 250               ' Wait for 1/4 second
  LOOP                      ' then repeat
```

Please note the conditional expression within the IF statement; it is true when the value of the inputs is 0. This is because the idle state of the optically isolated inputs is high. Therefore, when NO voltage is present at the input terminal, 5 VDC will be present on the Stamp input pin. The converse is also true: if the signal voltage (typically 12-36 VDC) is present on the input terminal, 0 VDC will be preset on the Stamp input. All digital inputs reference the ground at the terminal Din GND. The diagram to the right shows how a typical input could be connected.

Analog Inputs

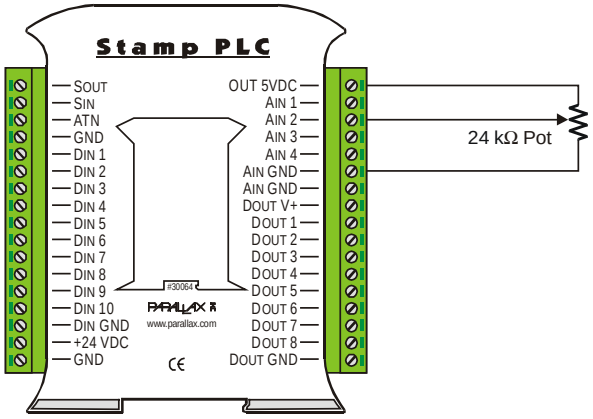
Analog inputs aren't needed for every PLC application. When they are required, the necessity for a high-quality, low-noise, full-featured Analog to Digital Converter becomes paramount. Since good ADCs aren't cheap, we designed it to be optional. That way, the core cost of the Stamp PLC is as low as possible.

Maxim's 12-bit, multi-range, 8-channel MAX1270 A/D Converter is the ADC of choice. This ADC can accept the following inputs: 0-5, 0-10, ±5, ±10 volts DC. The range of the input is selected by writing a configuration byte to the ADC. BASIC Stamps have built-in commands that configure and read ADCs with ease. In addition to the aforementioned ranges, the user may install a shunt (jumper) within the Stamp PLC, thereby configuring that channel to receive 4-20mA current loop signal.

Reading Analog Inputs

Configuring and reading the analog inputs is straightforward. The Stamp's built-in commands, SHIFTIN and SHIFTOUT, make the whole process quite easy and shrinks the code down to a manageable size.

Figure 9: Stamp PLC to Potentiometer



The first step is to connect the necessary circuitry while the Stamp PLC and your sensor (in our example a potentiometer) are de-energized. The connection diagram to the right demonstrates one way you can connect a potentiometer to an analog input. Note that the "Out 5VDC" terminal is supplying 5 Vdc to the circuit. This output comes from the Stamp's regulator. Do not draw more than 40mA from this terminal. This 5VDC output is referenced to the Stamp's ground.

The A/D Converter is configured by sending it a control byte. The control byte is made up of configuration bits[4]. The meaning of each configuration bit is defined by the chart in Figure 10.

Figure 10: A/D Converter control byte legend

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Start	SEL2	SEL1	SEL0	RNG	BIP	PD1	PD0
1	1	W	X	Y	Z	0	0

Figure 11: A/D Converter bit table

Analog Channel	W	X
4	0	0
3	0	1
2	1	0
1	1	1

Figure 12: A/D Converter bit table

Y	Z	Range
0	0	0 VDC to +5 VDC
0	1	-5 VDC to +5 VDC
1	0	0 VDC to +10VDC
1	1	-10 VDC to +10 VDC

The entire process of configuring the ADC and retrieving the conversion results can be accomplished with just a few PBASIC commands. The following program has been written to configure the ADC for 0-5 VDC and read the analog voltage present at Ain 2.

```
'{$STAMP  BS2}
'{$PBASIC 2.5}
'
ClkAdc          PIN     0           ' A/D clock
CsAdc           PIN     3           ' Chip Select for ADC
AoutAdc         PIN     4           ' A/D Data out
AinAdc          PIN     5           ' A/D Data in
adResult        VAR     Word

Main:
  DO
    LOW CsAdc
    SHIFTOUT AoutAdc, ClkAdc, MSBFIRST, [%11100000]   'Ch2 0-5 VDC
    HIGH CsAdc
    LOW CsAdc
    SHIFTIN AinAdc, ClkAdc, MSBPRE, [adResult\12]
    HIGH CsAdc
    DEBUG "  ADC2:", SDEC adResult, CR
  LOOP
```

The LOW CsAdc command sets the Chip Select input of the ADC low, thereby enabling communications on the ADC. The SHIFTOUT command sends the configuration to the ADC. The "HIGH CSadc" signals the completion of the configuration and the start of the data conversion. The data conversion process requires at least 9.09 uSec to complete. Since the inter-instruction time for all Stamps is higher than this, no additional delay is required. The next "LOW CDadc" command enables communication for the reply. The SHIFTIN command retrieves the converted data. The following "HIGH CSadc" command ends the communications. The DEBUG command formats and displays the data for our viewing ease.

[4] You may have noticed that the MAX1270 is an 8 channel A/D converter, (yet only 4 channels are employed). The fact that there are three channel select bits is a subtle hint. Analog input channels 1:4 are mapped to A/D converter channels 7:4. It was decided to limit the number of analog input channels to accommodate the number of connections available in this small enclosure.

This is a 12-bit ADC. As a result, the two least significant bit seem to jump around quite a bit. Here are a couple of suggestions on how to deal with this.

1) If you do not need 12-bits of resolution, simply divide the number down to the desired resolution. Ex: if you require 8-bits of resolution, just: "adresult = adresult>>4". This will eliminate the lower four bits are provide a nice stable number.

2) If your application requires 12-bits of resolution, you may average or filter the result. Averaging is just that - simply take a number of samples, each time adding the sample to an accumulator, then divide the accumulator by the number of samples taken. This approach will provide a stable number and is a satisfactory approach for a control system whose numbers change relatively slowly. Filtering is similar to averaging in that several samples are taken and averaged, the difference is the way that the numbers are sampled and averaged.

The first difference is that the samples taken are the latest four samples. Each time a new sample is taken the oldest is forsaken, replaced with its next newer sample. Essentially, this is a moving average. After each new sample is taken, the average is calculated based on the new sample and the previous three. This approach works well for control systems with fast moving variables. Contact Parallax if you are interested in implementing a PID control algorithm, which is beyond the scope of this document.

Due to the protective input circuitry present on each analog input channel, the value given for each channel is attenuated by approximately 4%. For example, if you were to read an input of exactly 5 VDC, the number given would be about 3932 instead of 4095. The easiest way to account for this difference is to multiply the number read by 1.042. This is easily done within the BASIC Stamp's program. Please review the Stamp PLC Core program listed within this document.

Stamp PLC Demo Program

```
' ============================================================================
'
'   File....... StampPLC.BS2
'   Purpose.... Stamp PLC Core Routines and Framework for Apps
'   Author..... Parallax, Inc. (Copyright 2003 - All Rights Reserved)
'   E-mail..... support@parallax.com
'   Started....
'   Updated.... 21 DEC 2003
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ============================================================================


' -----[ Program Description ]----------------------------------------------
'
' This program provides a set of core routines that can be used to create
' operational programs for the Stamp PLC.  Conditional compilation is used
' so that the code can run on any 24-pin BASIC Stamp 2 module.
'
' Notes on reading ADC channels:
'
' The inputs are protected and reduce the voltage felt on the MAX1270 input
' pins.  This accounts for code to get a full-scale count of 4095.
'
' The raw value (counts) from the ADC will be returned in "adcRaw", converted
```

```
' to millivolts, and returned in "mVolts."  Be aware that in bipolar mode the
' value of "mVolts" is signed.  A "1" in BIT15 of "mVolts" indicates a negative
' value.  The BASIC Stamp does not support division or multiplication of
' negative values.


' -----[ Revision History ]------------------------------------------------
'
' 21 DEC 2003 : Updated to correctly bit-align output aliases with OUTH


' -----[ I/O Definitions ]--------------------------------------------------

Clock           PIN     0                       ' shared clock
Ld165           PIN     1                       ' 74HC165 load
Di165           PIN     2                       ' 74HC165 data in (from)
AdcCS           PIN     3                       ' ADC chip select
AdcDo           PIN     4                       ' ADC data out (to)
AdcDi           PIN     5                       ' ADC data in (from)

Di9             PIN     6                       ' direct digital inputs
Di10            PIN     7

DOuts           VAR     OUTH                    ' direct digital outputs
DOutsLo         VAR     OUTC                    ' -- Do5 - Do8
DoutsHi         VAR     OUTD                    ' -- Do1 - Do4
Do1             PIN     14                      ' updated 21-DEC-03
Do2             PIN     15
Do3             PIN     12
Do4             PIN     13
Do5             PIN     10
Do6             PIN     11
Do7             PIN      8
Do8             PIN      9

Sio             CON     16                      ' serial IO (prog port)


' -----[ Constants ]-------------------------------------------------------

IsOn            CON     1                       ' for shadow regs
IsOff           CON     0

DirectOn        CON     0                       ' for direct IO pins only
DirectOff       CON     1

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T1200       CON     813                     ' for programming port
    T2400       CON     396
    T9600       CON     84
    T19200      CON     32
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T9600       CON     240
    T19200      CON     110
#ENDSELECT

Baud            CON     T9600                   ' default (matches DEBUG)
```

```
Ain1            CON     0                       ' analog channels
Ain2            CON     1
Ain3            CON     2
Ain4            CON     3

AdcUP5          CON     0                       ' unipolar, 0 - 5 v
AdcBP5          CON     1                       ' bipolar, +/- 5 v
AdcUP10         CON     2                       ' unipolar, 0 - 10 v
AdcBP10         CON     3                       ' bipolar, +/- 10 v
Adc420          CON     4                       ' 4-20 mA input


' -----[ Variables ]----------------------------------------------------

digIns          VAR     Word                    ' shadow digital inputs
dInLo           VAR     digIns.LOWBYTE          ' Din1 - Din8
dInHi           VAR     digIns.HIGHBYTE         ' Din9 - Din10
dIn1            VAR     digIns.BIT0
dIn2            VAR     digIns.BIT1
dIn3            VAR     digIns.BIT2
dIn4            VAR     digIns.BIT3
dIn5            VAR     digIns.BIT4
dIn6            VAR     digIns.BIT5
dIn7            VAR     digIns.BIT6
dIn8            VAR     digIns.BIT7
dIn9            VAR     digIns.BIT8
dIn10           VAR     digIns.BIT9

digOuts         VAR     Byte                    ' shadow digital outputs
dOut1           VAR     digOuts.BIT0            ' use Read_DigOuts to set
dOut2           VAR     digOuts.BIT1
dOut3           VAR     digOuts.BIT2
dOut4           VAR     digOuts.BIT3
dOut5           VAR     digOuts.BIT4
dOut6           VAR     digOuts.BIT5
dOut7           VAR     digOuts.BIT6
dOut8           VAR     digOuts.BIT7

chan            VAR     Nib                     ' ADC channel (0 - 3)
mode            VAR     Nib                     ' ADC mode (0 - 4)
config          VAR     Byte                    ' configuration byte
adcRes          VAR     Nib                     ' ADC bits (1 - 12)
adcRaw          VAR     Word                    ' ADC result (raw)
mVolts          VAR     Word                    ' ADC in millivolts

bitMap          VAR     Byte                    ' for re-mapping IO bits


' -----[ EEPROM Data ]--------------------------------------------------

Project         DATA    "Stamp PLC Template", 0

AdcCfg          DATA    %11110000, %11100000, %11010000, %11000000 ' 0-5
                DATA    %11110100, %11100100, %11010100, %11000100 ' +/-5
                DATA    %11111000, %11101000, %11011000, %11001000 ' 0-10
                DATA    %11111100, %11101100, %11011100, %11001100 ' +/-10
                DATA    %11110000, %11100000, %11010000, %11000000 ' 4-20


' -----[ Initialization ]-----------------------------------------------
```

```
Setup:
  LOW Clock                                     ' preset control lines
  HIGH Ld165
  HIGH AdcCS

  DOuts = %11111111                             ' all outputs off
  DIRH = %11111111                              ' enable output drivers

  adcRes = 12                                   ' use all ADC bits


' -----[ Program Code ]----------------------------------------------------

Main:

  ' demo - replace with your code
  '
  GOSUB Read_DigIns
  DEBUG HOME, "Inputs = ", BIN10 digIns, CR, CR

  ' copy inputs to outputs
  ' -- Din9  --> Dout1
  ' -- Din10 --> Dout2
  '
  digOuts = digIns
  GOSUB Update_DigOuts
  IF (dIn9 = IsOn) THEN Do1 = DirectOn
  IF (dIn10 = IsOn) THEN Do2 = DirectOn

  ' read single-ended analog inputs
  ' -- display input as millivolts
  '
  mode = AdcUP5
  FOR chan = Ain1 TO Ain4
    GOSUB Read_ADC
    DEBUG "Ain", ("1" + chan), ".... ",
          DEC (mVolts / 1000), ".", DEC3 mVolts, CR
  NEXT

  GOTO Main
  '
  ' end of demo code

  END


' -----[ Subroutines ]------------------------------------------------------

' Scans and saves digital inputs, DIn1 - DIn10
' -- returns inputs in "digIns" (1 = input active)

Read_DigIns:
  PULSOUT Ld165, 15                             ' load inputs
  SHIFTIN Di165, Clock, MSBPRE, [dinLo]         ' shift in
  dinHi = 0                                     ' clear upper bits
  dinHi.BIT0 = ~Di9                             ' grab DIN9
  dinHi.BIT1 = ~Di10                            ' grab DIN10
  RETURN


' Refreshes digital outputs, DOut1 - DOut8
```

```
' -- uses shadow register "digOuts" (1 = output on)
'
' 21-DEC-03 Update
' ----------------
' map bits in digOuts to physical connector

Update_DigOuts:
  DOuts.BIT0 = ~digOuts.BIT6
  DOuts.BIT1 = ~digOuts.BIT7
  DOuts.BIT2 = ~digOuts.BIT4
  DOuts.BIT3 = ~digOuts.BIT5
  DOuts.BIT4 = ~digOuts.BIT2
  DOuts.BIT5 = ~digOuts.BIT3
  DOuts.BIT6 = ~digOuts.BIT0
  DOuts.BIT7 = ~digOuts.BIT1
  RETURN


' This routine can be used to refresh shadow register "digOuts" after
' direct manipulation of individual output bits.

Read_DigOuts:
  dOut1 = ~DOuts.BIT6                           ' map bits from Stamp port
  dOut2 = ~DOuts.BIT7
  dOut3 = ~DOuts.BIT4
  dOut4 = ~DOuts.BIT5
  dOut5 = ~DOuts.BIT2
  dOut6 = ~DOuts.BIT3
  dOut7 = ~DOuts.BIT0
  dOut8 = ~DOuts.BIT1
  RETURN


' Reads analog input channel (0 - 5 vdc)
' -- put channel (0 - 3) in "chan"
' -- pass mode (0 - 4) in "mode")
' -- raw value returned in "adcRaw"
' -- "adcRaw" converted to signed "mVolts"

Read_ADC:
  READ AdcCfg + (mode * 4 + chan), config       ' get config
  LOW AdcCS                                     ' select MAX1270
  SHIFTOUT AdcDo, Clock, MSBFIRST, [config]     ' send config byte
  HIGH AdcCS                                    ' deselect MAX1270
  adcRaw = 0
  LOW AdcCS
  SHIFTIN AdcDi, Clock, MSBPRE, [adcRaw\12]     ' read channel value
  HIGH AdcCS

  ' adjust ADC count for input voltage divider
  '
  adcRaw = adcRaw + (adcRaw ** $D6C) MAX 4095   ' x ~1.05243


  ' millivolts conversion
  ' -- returns signed value in bipolar modes
  ' -- uses raw (12-bit) value
  '
  SELECT mode
    CASE AdcUP5
      mVolts = adcRaw + (adcRaw ** $3880)       ' x 1.2207
```

```
  CASE AdcBP5
    IF (adcRaw < 2048) THEN
      mVolts = 2 * adcRaw + (adcRaw ** $7100) ' x 2.4414
    ELSE
      adcRaw = 4095 - adcRaw
      mVolts = -(2 * adcRaw + (adcRaw ** $7100))
    ENDIF

  CASE AdcUP10
    mVolts = 2 * adcRaw + (adcRaw ** $7100)   ' x 2.4414

  CASE AdcBP10
    IF (adcRaw < 2048) THEN
      mVolts = 4 * adcRaw + (adcRaw ** $E1FF)
    ELSE
      adcRaw = 4095 - adcRaw
      mVolts = -(4 * adcRaw + (adcRaw ** $E1FF))
    ENDIF

  CASE Adc420                                 ' -- 4000 to 20000
    mVolts = 5 * adcRaw + (adcRaw ** $1666)   ' x 5.0875

ENDSELECT

' adjust adcRaw for selected resolution
'
IF (adcRes < 12) THEN
  adcRaw = adcRaw >> (12 - adcRes)            ' reduce resolution
ENDIF

RETURN
```

Serial Communications

Once your program is loaded and running, you are free to use the serial port for "run-time" communications. Each Stamp has built-in functions, (serin and serout), that allow you to talk to your Stamp via a PC serial port, or allow the Stamp PLC to talk to other devices with RS-232 style serial ports. In fact, this port can be used for a variety of useful functions. Here are a few sample programs that show some of the possibilities.

Status Reporting

```
' =============================================================================
'
'   File....... Stamp PLC_Report.BS2
'   Purpose.... Typical PLC application whereby this unit listens for
'               serial commands from a host controller.
'   Author..... Parallax, Inc.
'   E-mail..... support@parallax.com
'   Started.... 26 JUN 2003
'   Updated.... 26 JUN 2003
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================
```

```
' -----[ Program Description ]---------------------------------------------
'   This program demonstrates how to periodically "listen" for a serial
'   command while performing another task.  When a serial command is
'   received, it is parsed and acted upon.  If no command is received
'   within 5 seconds, the serial routine times out and returns to the
'   user task.

' -----[ Revision History ]--------------------------------------------------


' -----[ I/O Definitions ]---------------------------------------------------


' -----[ Constants ]---------------------------------------------------------

T1200           CON     813
T2400           CON     396
T9600           CON     84
T19200          CON     32

Baud            CON     T9600

DegSym          CON     $B0                     ' degrees symbol


' -----[ Variables ]---------------------------------------------------------

waxTemp         VAR     Byte
keyIn           VAR     Byte                    ' terminal input


' -----[ EEPROM Data ]-------------------------------------------------------
' -----[ Initialization ]----------------------------------------------------

Startup:
  waxTemp = 105
  GOSUB Initialize


' -----[ Program Code ]------------------------------------------------------
Main:
  DO
    SERIN 16, Baud, 50, Run_Task, [keyIn]       ' Run_Task if no input
  LOOP UNTIL (keyIn = "W")                      ' wait for "W"

Ready_Prompt:
  SEROUT 16, Baud, [CR, LF, "Ready!", CR]

Do_Command:
  DO
    SERIN 16, Baud, 5000, Back_To_Work, [keyIn] ' wait for command

    SELECT keyIn                                ' process command
      CASE "?"
        SEROUT 16, Baud,
               [CR, LF, "Wax Temperature = ",
                DEC WaxTemp, DegSym, "F", CR]

      CASE ELSE
        IF (keyIn <> "X") THEN
          SEROUT 16, Baud, [BELL]               ' bell for invalid input
```

```
        ENDIF
    ENDSELECT

  LOOP UNTIL (keyIn = "X")

Back_To_Work:
  SEROUT 16, Baud, [CR, LF, "Returning to work"]
  GOSUB Initialize

Run_Task:                                       ' User task code goes here
  SEROUT 16, Baud, ["."]                        ' -- progress indicator
  GOTO Main

' -----[ Subroutines ]------------------------------------------------------

Initialize:                                     ' setup for user task goes
  SEROUT 16, Baud, [CR, LF, "Working"]          '   goes here
  RETURN
```

Sometimes it is necessary to query a PLC for a particular piece of information. The example above shows how to create two modes within one program. The default mode is the working mode, and it splits its time into two functions: running the user program (which has been omitted for clarity), and checks the serial port to see if there is an input from the master, (you). The second mode is invoked if the proper wake up character is received from the master. Once in this mode, data will be reported if the master sends a "?" command. If no command is received for a period of 5 seconds or more, the program will automatically revert to the first mode (running the user program and monitoring the serial line).

Password Protection

PLCs typically control machinery. Quite often, these machines and/or the products that they make are worth quite a bit of money. Given this, security is an issue. To safeguard the program, and any variables that should not be changed, it is sometimes necessary to control access to the PLC. A common way to do this is to utilize a password.

```
' =============================================================================
'
'   File....... Stamp PLC_Password.BS2
'   Purpose.... Typical PLC application whereby this unit listens for a
'               password on the serial port before allowing commands to be
'               parsed.
'   Author..... Parallax, Inc.
'   E-mail..... support@parallax.com
'   Started.... 25 JUN 2003
'   Updated.... 25 JUN 2003
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'   This program demonstrates how to periodically "listen" for a password
'   message on the serial port while performing another task. Once the
'   correct password is given, the password may be altered or cleared.
'   The password is stored in eeprom so it is non-volatile.

' -----[ Revision History ]------------------------------------------------
```

```
' -----[ I/O Definitions ]-------------------------------------------------


' -----[ Constants ]-------------------------------------------------------

T1200           CON     813
T2400           CON     396
T9600           CON     84
T19200          CON     32

Baud            CON     T9600


' -----[ Variables ]-------------------------------------------------------

keyIn           VAR     Byte                    ' terminal input
serString       VAR     Byte(4)


' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]---------------------------------------------------

Startup:
  GOSUB Read_Password                           ' Read password
  IF keyIn = $FF THEN GOSUB Set_Password        ' If not set, set it
                                                ' else, start main code

' -----[ Program Code ]-----------------------------------------------------

Main:
  SERIN 16, Baud, 500, Run_Program, [WAITSTR serString\4]
  DEBUG CR, "Password accepted!"
  DO                                            ' Once password received
    DEBUG CR, "C:> "
    SERIN 16, Baud, [keyIn]                     ' get a command
    SELECT keyIn                                ' process command
      CASE "X"
        GOTO Resume_Program
      CASE "N"
        GOSUB Set_Password
      CASE "C"
        GOSUB Clear_Password
      CASE ELSE
        DEBUG " : Invalid command",BELL
     ENDSELECT
  LOOP

Resume_Program:
  DEBUG CR                                      ' otherwise, run the

Run_Program:                                    ' user's program
  DEBUG "*"                                     ' user program goes here
  GOTO Main

' -----[ Subroutines ]-----------------------------------------------------

Read_Password:                                  ' Read password from eeprom
```

```
  FOR keyIn = 0 TO 3
    READ keyIn, serString(keyIn)
  NEXT
  keyIn = serString(0)
  RETURN

Set_Password:                                   ' User interface to enter
  DEBUG CR, "Enter a 4-character password", CR  ' password
  SERIN 16, Baud, [STR serString\4]
  DEBUG CR, "Confirm password: "
  SERIN 16, Baud, [WAITSTR serString\4]
  FOR keyIn = 0 TO 3                            ' Write password to eeprom
    WRITE keyIn,serString(keyIn)
  NEXT
  DEBUG CR, "Password set", CR, "Working!", CR
  keyIn = "N"
  RETURN

Clear_Password:                                 ' Clear password in RAM
  FOR keyIn = 0 TO 3                            ' to $FF's and in eeprom
    WRITE keyIn, $FF                            ' to "0"'s
    serString(keyIn) = $30
  NEXT
  DEBUG CR, "Password cleared", CR
  keyIn = "C"
  RETURN
```

Remote Telemetry Unit

Data loggers are devices that record data to be retrieved at some later point in time. Another name for a similar device is a Remote Telemetry Unit, or RTU. An RTU differs from a Data Logger in that it has a limited amount of control ability. A good example of this type of RTU is a Dual Pump Controller. The Dual Pump Controller is responsible for maintaining a parameter, like fluid level within a tank, by controlling and monitoring two pumps.

- Before the advent of small microcontrollers, a simple float switch would have performed this task. The Dual Pump Controller can do this as well as several other useful features:
- Rotate duty between two pumps – evening the wear.
- Use only one pump if the other is out of commission.
- Use both pumps if the level is very low.
- Record periodic tank levels and other parameters at periodic intervals.
- Record the amount of run-time of each pump.
- Send a notification if there is a fault with a pump.
- Transmit the data recorded when called for.

Our Dual Pump Controller sample program is not as full-featured as it could be, but is complete enough to get you pointed in the right direction. Since it is a comparatively long listing, we will break it into sequential segments and discuss each block as it comes. There's not a lot to discuss with the first block, but reading the program description and the note are good first steps.

```
' =============================================================================
'
'   File....... Stamp PLC_PumpController.BS2
'   Purpose.... Dual Pump Controller.  Example program.
''   Author..... Parallax, Inc.
'   E-mail..... support@parallax.com
'   Started.... 27 JUN 2003
```

```
'   Updated.... 27 JUN 2003
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ================================================================================
'
' -----[ Program Description ]---------------------------------------------
'   This program controls two pumps to maintain tank level.  When the level
'   falls below setpoint #1, the pump with the fewest number of hours is
'   started.  If the level should continue falling to setpoint#2, the other
'   pump is started as well.  During pump startup, the pump's aux contact
'   and flow sensor are monitored TO determine whether or not the pump
'   actually started.  If a failure does occur, that pump is immediately
'   shutdown and flagged as out-of-commission, and the other pump starts.
'   While all of this is going on, data is logged on pump runtimes, tank
'   level AND the serial port is monitored for commands and queries.
'
'   Note: To more easily see this program work, you can change the constant
'         in "Timer_Logic" from 3600 to 3 or so.  This will effectively
'         speed time up so you can watch all aspects of this program work.
'         Also, you may wish to enable the DEBUG statements too.
'
' -----[ Revision History ]------------------------------------------------
```

This next section defines the Stamp I/O pin connections, the variable sizes and names, and the size of the record array used to store data collected. Note that some variables as declared are actually parts of other variables.

```
' -----[ I/O Definitions ]-------------------------------------------------
AdcClk          PIN     0         ' A/D clock input
SrClk           PIN     0         ' HC165 clock input
Load            PIN     1         ' Latch input for HC165
SrDin           PIN     2         ' HC165 data input
AdcCs           PIN     3         ' A/D chip select input
AdcDout         PIN     4         ' A/D Data out
AdcDin          PIN     5         ' A/D Data in
PumpA           PIN     8         ' 1 turns on Pump A
PumpB           PIN     9         ' 1 turnd on Pump B

' -----[ Constants ]-------------------------------------------------------
T1200           CON     813
T2400           CON     396
T9600           CON     84
T19200          CON     32
Baud            CON     T9600

' -----[ Variables ]-------------------------------------------------------
tankLevel       VAR     Word        ' Actual level of the tank
timer           VAR     Word        ' Sets the periodicity
runTimeA        VAR     Byte        ' # hours that Motor A has run
runTimeB        VAR     Byte        ' # hours that Motor B has run
inputs          VAR     Byte        ' Inputs 1 - 8 from the HC165
tmp             VAR     Byte        ' Temporary work variable
ptr             VAR     Byte        ' Points to the current record
keyIn           VAR     Byte        ' Variable for keyboard input
pumpStatus      VAR     Byte        ' Status of both pumps
setPoint1       VAR     Byte        ' If level <, start 1 pump
setPoint2       VAR     Byte        ' If level <, start 2 pumps
startCode       VAR     Nib         ' Desired pump configuration
```

```
                                        ' Inputs via HC165
flowA           VAR     Inputs.BIT0     '1 = Flow  OK, 0 = No Flow
flowB           VAR     Inputs.BIT1
auxContactA     VAR     Inputs.BIT2     '1 = Motor ON, 0 = Motor off
auxContactB     VAR     Inputs.BIT3
                                        ' Bit variables
pumpAStatus     VAR     pumpStatus.LOWNIB
pumpBStatus     VAR     pumpStatus.HIGHNIB

' -----[ EEPROM Data ]-----------------------------------------------------
Records    DATA  0(256)                 ' Data table for records.  Note! when
                                        ' the pointer > 64, the old data will
                                        ' be overwritten by new data because
                                        ' each record = 4 bytes.
' -----[ Notes ]-----------------------------------------------------------
'                       Pump Status Definition
'                   0     OK Off           8         FAILED no flow
'                   1     OK On            9         FAILED no aux contact
'                  2-7    undefined       A-F        undefined
'
'                       Start Codes defined
'                        1   PumpA desired
'                        2   PumpB desired
'                        3   Both pumps desired
```

The first section of code handles the initialization. The first line, in this case, is not necessary because the Stamp clears all variables to zero automatically. The program consists of a large loop that is executed over, and over again. Within the main loop, the serial port is examined. If the host has sent a character, that character is received, parsed and executed. If no characters are received within one second, or once the received character has been parsed the monitor and control section of code is executed.

```
' -----[ Initialization ]-------------------------------------------------

  ptr = 0                               ' Set record pointer to the start
  setPoint1 = 100                       ' First low level setpoint
  setPoint2 =  50                       ' Second low level setpoint

' -----[ Main Program ]---------------------------------------------------

Main:
  SERIN 16, Baud, 1000, Control_Level, [keyIn]
  SELECT keyIn
    CASE "A"
      SEROUT 16, Baud, [CR, "Runtime A:", DEC runTimeA, CR]
    CASE "B"
      SEROUT 16, Baud, [CR, "Runtime B:", DEC runTimeB, CR]
    CASE "a"
      runTimeA = 0
      SEROUT 16, Baud, [CR, "Runtime A cleared", CR]
    CASE "b"
      runTimeB = 0
      SEROUT 16, Baud, [CR, "Runtime B cleared", CR]
    CASE "D"
      IF ptr > 0 THEN
        SEROUT 16, Baud, [CR, "Tank Level Report", CR, CR]
        FOR tmp = 0 TO (ptr-4 MIN 1) STEP 4
          READ tmp+3,keyIn
          SEROUT 16, Baud, ["Hour: ", DEC3 tmp/4,
                            "  Tank Level: ", DEC3 keyIn, CR]
        NEXT
        READ tmp-3,keyIn
```

```
        READ tmp-2,tmp
        SEROUT 16, Baud, ["Pump A Runtime: ", DEC3 keyIn,
                          " Pump B Runtime: ", DEC3 tmp, CR, CR]
      ELSE
        SEROUT 16, Baud, [CR, "No data yet!", CR, CR]
      ENDIF
    CASE ELSE
      SEROUT 16, Baud, [" : Invalid command", BELL, CR, CR]
   ENDSELECT

Control_Level:
  GOSUB Read_Tank_Level
  GOSUB Control_Pumps
'  GOSUB Debug_Data
  GOSUB Error_Handler
  GOSUB Timer_Logic
  GOTO Main
```

As described by the names of the subroutines, the tank level is read, pumps are controlled, errors are handled, and the timer is maintained before returning to the top of the program where this sequence of events proceeds forever.

Each main loop iteration takes one second. The Timer_Logic subroutine increments a register called "Timer" each time through. After 3600 interations, (one hours time), data is recorded and the runtime of each pump is updated. The Debug_Data subroutine is normally commented out, but can be enabled at any time to "see" what's going on.

```
' -----[ Subroutines ]------------------------------------------------------

Timer_Logic:
  timer = timer + 1
  DEBUG ?timer
  IF (timer > 3600) THEN
'    DEBUG CR, "Data Recorded!  Pointer = ",HEX2 ptr, CR
    GOSUB Record_Data
    timer = 0
    IF (pumpAStatus = 1) THEN runTimeA = runTimeA + 1
    IF (pumpBStatus = 1) THEN runTimeB = runTimeB + 1
  ENDIF
  RETURN

Debug_Data:
  DEBUG "Status: ", HEX2 pumpStatus, CR
  RETURN

Record_Data:
  WRITE ptr+0,pumpStatus
  WRITE ptr+1,runTimeA
  WRITE ptr+2,runTimeB
  WRITE ptr+3,tankLevel.LOWBYTE
  ptr = ptr + 4
  RETURN

Stop_Pumps:
  HIGH PumpA: pumpAStatus = 0
  HIGH PumpB: pumpBStatus = 0
  RETURN

Error_Handler:
```

```
  SELECT  startCode
    CASE  1
      IF (PumpAStatus > 7) THEN GOSUB StartB
    CASE  2
      IF (PumpBStatus > 7) THEN GOSUB StartA
  ENDSELECT
  RETURN
```

Since the output drivers for the Stamp PLC are "high-side" drivers, you must issue a LOW command to energize an output, and likewise, you must issue a HIGH command to de-energize an output. The Error_Handler subroutine checks the start code of each pump before starting it. If the pump is out of commission, the other pump is started instead.

Each start routine functions the same way. If the pump is OFF, it attempts to start it. If the aux-contact responds and, within 2 seconds, the flow responds, then the pump is considered to be running properly. If either the flow indication or aux-contactor input fails, then the pump is flagged as out-of-commission and de-energized.

```
StartA:
  IF (pumpAStatus = 0) THEN
    LOW PumpA
    GOSUB GetDigitalInputs
    IF auxContactA = 0 THEN
      PAUSE 2000
      GOSUB GetDigitalInputs
      IF FlowA = 0 THEN
        pumpAStatus = 1
      ELSE
        pumpAStatus = 8
        HIGH PumpA
      ENDIF
    ELSE
      pumpAStatus = 9
      HIGH PumpA
    ENDIF
  ENDIF
  GOSUB Error_Handler
  RETURN

StartB:
  IF (pumpBStatus = 0) THEN
    LOW PumpB
    GOSUB GetDigitalInputs
    IF auxContactB = 0 THEN
      PAUSE 2000
      GOSUB GetDigitalInputs
      IF (FlowB = 0) THEN
        pumpBStatus = 1
      ELSE
        pumpBStatus = 8
        HIGH PumpB
      ENDIF
    ELSE
      pumpBStatus = 9
      HIGH PumpB
    ENDIF
  ENDIF
  GOSUB Error_Handler
  RETURN
```

The subroutine "Control_Pumps" decides which pump to start based on how much run-time is on each pump. The idea here is to even the wear.

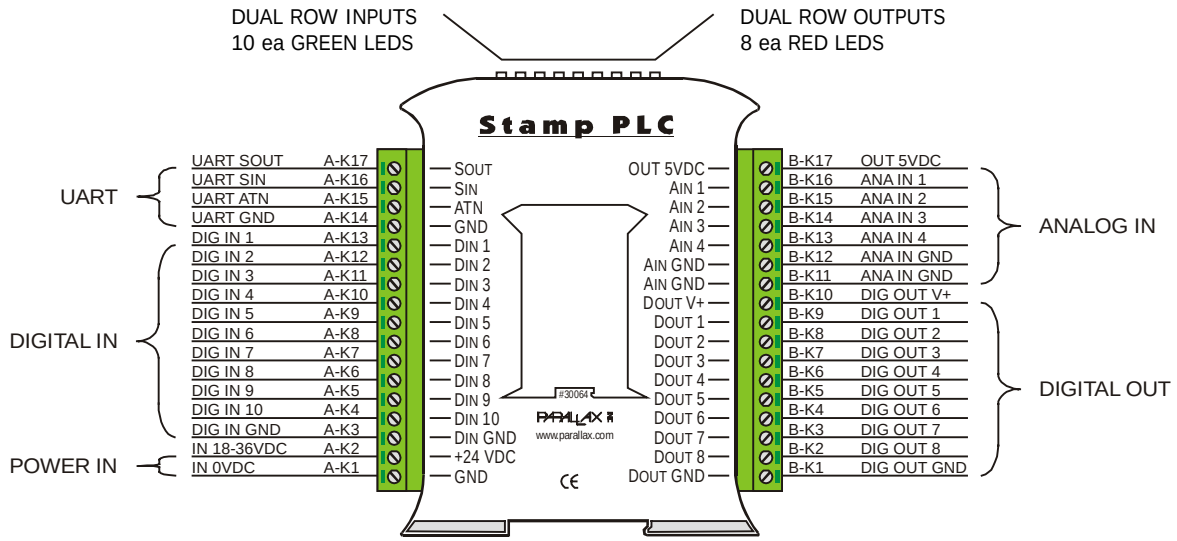
```
Control_Pumps:
  SELECT tankLevel
    CASE < setPoint2
      startCode = 3
      GOSUB StartA
      GOSUB StartB
    CASE < setPoint1
      IF (pumpAStatus <> 1 AND pumpBStatus <>1) THEN
        IF RunTimeA < RunTimeB THEN
          startCode = 1
          GOSUB StartA
        ELSE
          startCode = 2
          GOSUB StartB
        ENDIF
      ENDIF
    CASE ELSE
      GOSUB Stop_Pumps
  ENDSELECT
  RETURN

GetDigitalInputs:
  LOW load
  PAUSE 1
  HIGH load
  PAUSE 1
  SHIFTIN SrDin,SrClk, LSBPRE, [tmp]
  RETURN

Read_Tank_Level:                                'Reads tank level
  LOW AdcCs
  PAUSE 1
  SHIFTOUT AdcDout, AdcClk, MSBFIRST, [240]   'Tank Level
  HIGH AdcCs
  PAUSE 1
  LOW AdcCs
  PAUSE 1
  SHIFTIN AdcDin, AdcClk, MSBPRE, [tankLevel\12]
  PAUSE 1
  HIGH AdcCs
  tankLevel = tankLevel>>4                      'Divide it by 8
'  DEBUG "Tank Level:", DEC3 tankLevel, " inches  "
  RETURN
```

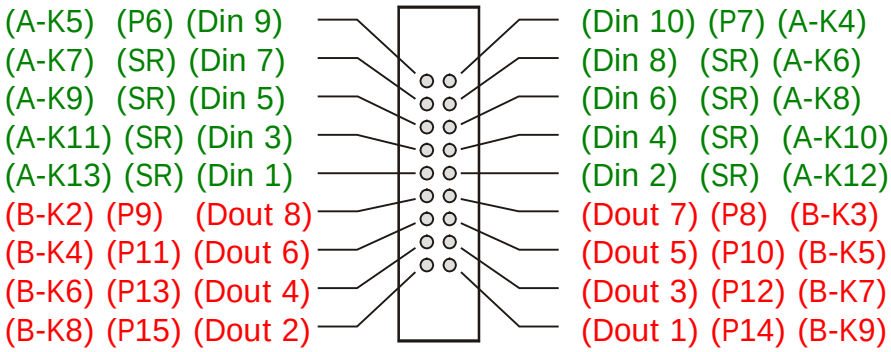
The other two subroutines on this section of code are straight forward, but for the trick we play with the "tankLevel" variable. The A/D converter is a 12-bit device. Our example program is only concerned with the upper 8 bits of data, so we shift the data four positions to the right. This is a fast way to divide by 16 within a binary digital system. If you wanted more precision for you're application simply forego this step, but beware that you need to ensure the memory you use must accommodate the extra bits.

Figure 11: Connection Diagram



This diagram identifies each connection on the Stamp PLC by name and by number. Please note that this is a top view. Throughout this manual, the Stamp PLC connections may be referred to by either their names or their numbers. Please refer to this diagram when making or changing connections. Each connection can accept 18 to 12 gauge wire.

Figure 12: Front Panel LEDs



Note: SR = connected to 74HC165 parallel in - serial out shift register

Safety Notes and Liability Disclaimer

Please bear in mind that these sample programs are just that - samples. They have been written to show how certain functions *could* be implemented. Depending on you're application, the implementation shown may not be the way it *should* be implemented. When designing, implementing, and programming PLCs that control equipment, use extreme care to ensure that your design always: starts in a predictable fashion, performs within limits, is mechanically and electrically interlocked where applicable, and fails safe.

Neither Parallax, Inc. nor Lawicel are responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products.

Absolute Maximum Ratings

Absolute maximum ratings indicate sustained limits beyond which damage to the device may occur. All voltage parameters are referenced to GROUND lead. (T_j = 25°C unless otherwise specified).

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{out}	Maximum output voltage	V_{cc}-50	V_{cc}+0.3	V	
V_{offset}	Maximum logic ground to load ground offset	V_{cc}-50	V_{cc}+0.3		
V_{in}	Maximum Input voltage	-0.3	5.5		
$I_{in, max}$	Maximum IN current	-5	10	mA	
V_{dg}	Maximum diagnostic output voltage	-0.3	5.5	V	
$I_{dg, max}$	Maximum diagnostic output current	-1	10	mA	
$I_{sd\ cont.}$	Diode max. continuous current (1) (IPS511G)	—	1.4	A	
	(per leg/both legs ON - IPS512G)	—	0.8		
	(per leg/all legs ON - IPS514G)	—	0.7		
$I_{sd\ pulsed}$	Diode max. pulsed current (1)	—	10		
ESD1	Electrostatic discharge voltage (Human Body)	—	4	kV	C=100pF, R=1500Ω,
ESD2	Electrostatic discharge voltage (Machine Model)	—	0.5		C=200pF, R=0Ω, L=10μH
P_d	Maximum power dissipation (rth=125°C/W) IPS511G	—	1	W	
	(rth=85°C/W, both legs on) IPS512G	—	1.5		
	(rth=50°C/W, all legs on) IPS514G	—	2.5		
T_j max.	Max. storage & operating junction temp.	-40	+150	°C	
Vcc max.	Maximum Vcc voltage	—	50	V	

Thermal Characteristics

Symbol	Parameter	Min.	Typ.	Max.	Units	Test Conditions
R_{th1}	Thermal resistance with standard footprint	—	100	—	°C/W	8 Lead SOIC
R_{th2}	Thermal resistance with 1" square footprint	—	80	—		
R_{th1} (2 mos on)	Thermal resistance with standard footprint (2 mosfets on)	—	85	—		16 Lead SOIC
R_{th2} (1) (1 mos on)	Thermal resistance with standard footprint (1 mosfet on)	—	100	—		
R_{th2} (2 mos on)	Thermal resistance with 1" square footprint (2 mosfets on)	—	50	—		
R_{th1}	Thermal resistance with standard footprint	—	60	—		28 Lead SOIC
R_{th2} (2 mos on)	Thermal resistance with standard footprint (2 mosfets on)	—	55	—		
R_{th3} (4 mos on)	Thermal resistance with standard footprint (4 mosfets on)	—	50	—		
R_{th1}	Thermal resistance with 1" square footprint	—	45	—		
R_{th2} (2 mos on)	Thermal resistance with 1" square footprint (2 mosfets on)	—	40	—		
R_{th3} 4 mos on)	Thermal resistance with 1" square footprint (4 mosfets on)	—	35	—		

(1) Limited by junction temperature (pulsed current limited also by internal wiring)