

# Industrial Control


---

Student Guide

Version 1.1

Note regarding the accuracy of this text:

Many efforts were taken to ensure the accuracy of this text and the experiments, but the potential for errors still exists. If you find errors or any subject requiring additional clarification, please report this to [stampsinclass@parallaxinc.com](mailto:stampsinclass@parallaxinc.com) so we can continue to improve the quality of our documentation.

PARALLAX 

## Warranty

Parallax warrants its products against defects in materials and workmanship for a period of 90 days. If you discover a defect, Parallax will, at its option, repair, replace, or refund the purchase price. Simply call for a Return Merchandise Authorization (RMA) number, write the number on the outside of the box and send it back to Parallax. Please include your name, telephone number, shipping address, and a description of the problem. We will return your product, or its replacement, using the same shipping method used to ship the product to Parallax.

## 14-Day Money Back Guarantee

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. Parallax will refund the purchase price of the product, excluding shipping / handling costs. This does not apply if the product has been altered or damaged.

## Copyrights and Trademarks

This documentation is copyright 1999 by Parallax, Inc. BASIC Stamp is a registered trademark of Parallax, Inc. If you decide to use the name BASIC Stamp on your web page or in printed material, you must state: "BASIC Stamp is a registered trademark of Parallax, Inc." Other brand and product names are trademarks or registered trademarks of their respective holders.

## Disclaimer of Liability

Parallax, Inc. is not 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 or recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your BASIC Stamp application, no matter how life threatening it may be.

## Internet Access

We maintain Internet systems for your use. These may be used to obtain software, communicate with members of Parallax, and communicate with other customers. Access information is shown below:

E-mail: [stampsinclass@parallaxinc.com](mailto:stampsinclass@parallaxinc.com)  
Web: <http://www.parallaxinc.com> and <http://www.stampsinclass.com>

## Internet BASIC Stamp Discussion List

We maintain two e-mail discussion lists for people interested in BASIC Stamps (subscribe at <http://www.parallaxinc.com> under the technical support button). The BASIC Stamp list server includes engineers, hobbyists, and enthusiasts. The list works like this: lots of people subscribe to the list, and then all questions and answers to the list are distributed to all subscribers. It's a fun, fast, and free way to discuss BASIC Stamp issues and get answers to technical questions. This list generates about 40 messages per day.

The Stamps in Class list is for students and educators who wish to share educational ideas. To subscribe to this list, go to <http://www.stampsinclass.com> and look for the E-groups list. This list generates about five messages per day.

Table of Contents

Preface ..... iii  
 Preface ..... iii  
 Audience and Teacher’s Guides .....iv  
 Copyright and Reproduction .....v  
 Experiment #1: Flowcharting and Stamp Plot Lite ..... 7  
 Adjusting the Temperature for a Shower Example ..... 8  
 Conveyor Counting Example ..... 10  
 Exercise #1: Flowchart Design ..... 14  
 Exercise #2: LED Blinking Circuit ..... 14  
 Exercise #3: Analog Data ..... 17  
 Exercise #4: Using Stamp Plot Lite ..... 20  
 Questions and Challenge ..... 25  
 Experiment #2: Digital Input Signal Conditioning ..... 27  
 Exercise #1: Switch Basics ..... 32  
 Exercise #2: Switch Bounce and Debouncing Routines ..... 37  
 Exercise #3: Edge Triggering ..... 40  
 Exercise #4: An Electronic Switch ..... 47  
 Exercise #5: Tachometer Input ..... 52  
 Questions and Challenge ..... 64  
 Experiment #3: Digital Output Signal Conditioning ..... 71  
 Exercise #1: Sequential Control ..... 74  
 Exercise #2: Current Boosting the BASIC Stamp ..... 85  
 Questions and Challenge ..... 91  
 Experiment #4: Continuous Process Control ..... 97  
 Exercise #1: Closed Loop On-Off Control ..... 98  
 Exercise #2: Open-Loop vs. Closed-Loop Control ..... 113  
 Questions and Challenge ..... 125  
 Experiment #5: Closed-Loop Control ..... 127  
 Exercise #1: Establishing Closed-Loop Control ..... 130  
 Exercise #2: Differential-Gap Control ..... 136  
 Questions and Challenge ..... 142  
 Experiment #6: Proportional Integral Derivative Control ..... 145  
 Exercise #1: Bias Drive ..... 155  
 Exercise #2: Proportional Integral Control ..... 172  
 Exercise #3: Derivative Control ..... 179  
 Questions and Challenge ..... 187

## Contents

---

Experiment #7: Real-time Control and Data Logging.....	189
Exercise #1: Real Time Control .....	192
Questions and Challenge.....	199
Exercise #2: Interval Timing.....	199
Questions and Challenges.....	203
Exercise #3: Data Logging .....	204
Questions and Challenges.....	219
Appendix A: Stamp Plot Lite .....	221
Appendix B: Encoder Printouts.....	233
Appendix C: Potter Brumfield SSR Datasheet .....	235
Appendix D: National Semiconductor LM34 Datasheet .....	239
Appendix E: National Semiconductor LM358 Datasheet .....	245
Appendix F: Dallas Semiconductor 1302 Datasheet .....	251
Appendix G: Parts Listing and Sources .....	257
Appendix H: Commercial Incubator Challenge .....	261

## Preface

Industrial process control is a fascinating and challenging area of electronics technology and nothing has revolutionized this area like the microcontroller. The microcontroller has added a level of intelligence to the evaluation of data and a level of sophistication in the response to process disturbances. Microcontrollers are embedded as the “brains” in both manufacturing equipment and consumer electronic devices.

Process control involves applying technology to an operation that alters raw materials into a desired product. Virtually everything that you use or consume has undergone some type of automatic process control in its production. In a manufacturing environment, automatic process control also provides higher productivity and better product consistency while reducing production costs.

This text is intended to introduce you to the concepts and characteristics of microcontroller-based process control with the following experiment-based themes:

- a) Writing a procedural program from a flowchart for sequential process-control.
- b) Using pushbuttons, counting cycles and understanding simple I/O processes that form a system “under control”.
- c) Continuous process-control beginning with on-off control to more complex differential gap with multiple levels of control action.
- d) Proportional-integral-derivative control of a small desktop heating system.
- e) Time-based control of the above and introduction to data logging.

The hardware needed in the experiments to simulate the process has been kept to a bare minimum. While the microcontroller is the “brains” of the process, it is not the “muscle.” Actual applications require the microcontroller to read and control a wide variety of input and output (I/O) devices. Simple breadboard mounted pushbutton switches are used to simulate the action of mechanical and electro-mechanical switches found in industry. Visible light emitting diodes, small fans, and low-wattage resistors simulate motor starters and HVAC equipment. Information included in the experiments will help you understand the electrical interfacing of “real world” I/O devices to the BASIC Stamp.

The physical nature of the elements in a system determines the most appropriate mode of control action. The dynamics of a process include a study of the relationship of input disturbances and output action on the measured variables. It is difficult to understand the dynamics of a process without being able to “see” this relationship. For the authors, this defined a need to develop a graphical interface for the BASIC Stamp; hence the creation and release of StampPlot Lite. This software allows digital and analog values to be plotted on graphs, and time-stamped data and messages to be stored. StampPlot Lite is used throughout the experiments, and is especially helpful as you investigate the various modes of process control. Typical screen shots from program runs are included.

This text is the first major revision and we have strived to make it better than the first. Some changes and additions include:

- a) Addition of a 7<sup>th</sup> section on Time-Based control.
- b) A total rewrite of the PID section to better demonstrate and explain the theory.
- c) The additions of FET and PWM sample-and-hold circuitry and theory.
- d) The reworking of numerous example programs including more flowcharts and program explanations.

We thank our editors Ms. Cheri Barrall and Dale Kretzer, and of course Ken Gracey and Russ Miller of the Parallax staff for their review and improvement of this text. Further, we thank Dr. Clark Radcliffe of Michigan State University for his in-depth review. A variety of additional Parallax educational customers too numerous to list also provided valuable feedback for this second revision.

The authors are instructors at Southern Illinois University in Carbondale in the Electronic Systems Technologies program and also partners of a consulting and software company, SelmaWare Solutions. Visit the website to see examples of StampPlot Pro specifically tailored to users of this text.

We invite your comments and feedback. Please contact at us through our website, and copy all error changes to Parallax at [stampsinclass@parallaxinc.com](mailto:stampsinclass@parallaxinc.com) so the text may be revised.

Will Devenport and Martin Hebel  
Southern Illinois University, Carbondale  
Electronic Systems Technologies  
<http://www.siu.edu/~imsasa/est>

-- and --

SelmaWare Solutions  
<http://www.selmaware.com>

## Audience and Teacher's Guide

This text is aimed at an audience ages 17 and older. Effective during the first publication of this text in June, 2000, there is no Teacher's Guide edition planned. If a Teacher's Guide were to be published, it would likely be available the first part of year 2002. Solving these experiments presents no difficult technical hurdles, and can be done with a bit of patience.

## Copyright and Reproduction

Stamps in Class lessons are copyright © Parallax 2001. Parallax grants every person conditional rights to download, duplicate, and distribute this text without our permission. The condition is that this text, or any portion thereof, should not be duplicated for commercial use resulting in expenses to the user beyond the marginal cost of printing. That is, *nobody* should profit from duplication of this text. Preferably, duplication should have no expense to the student. Any educational institution wishing to produce duplicates for its students may do so without our permission. This text is also available in printed format from Parallax. Because we print the text in volume, the consumer price is often less than typical xerographic duplication charges. This text may be translated into any language with the prior permission of Parallax, Inc.







### Experiment #1: Flowcharting and StampPlot Lite

A flowchart is a detailed graphic representation illustrating the nature and sequencing of an operation on a step-by-step basis. A flowchart may be made of an everyday task such as driving to the store. How many steps are involved in this simple task? How many decisions are made in getting to the store? A formalized operation such as baking cookies can be flowcharted, whether

on a small-scale process in your kitchen or on a very large scale in a commercial bakery. And, of course, a flowchart also may be made of the steps and decisions necessary for a computer or microcontroller to carry out a task.

A relatively simple process is usually easy to understand and flows logically from start to finish. In the case of baking cookies, the steps involved are fairly easy. A recipe typically requires mixing the required ingredients, forming the cookies and properly baking them. There are several decisions to make: Are the ingredients mixed enough? Is the oven pre-heated? Have the cookies baked for the recommended time?

As processes become more complex, however, it is equally more difficult to chart the order of events needed to reach a successful conclusion. A BASIC Stamp program may have several dozen steps and possibly a number of "if-then" branches. It can be difficult to grasp the flow of the program simply by reading the code.

A flowchart is made up of a series of unique graphic symbols representing actions, functions, and equipment used to bring about a desired result. Table 1.1 summarizes the symbols and their uses.

Table 1.1: Flowchart Symbols



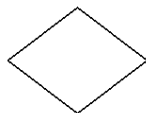
Start/Stop box indicates the beginning and end of a program or process.



Process box indicates a step that needs to be accomplished.



Input/Output box indicates the process requires an input or provides an output.



Decision box indicates the process has a choice of taking different directions based on a condition. Typically, it is in the form of a yes-no question.



Flowline is used to show direction of flow between symbols.



Connector box is used to show a connection between points of a single flowchart, or different flowcharts.



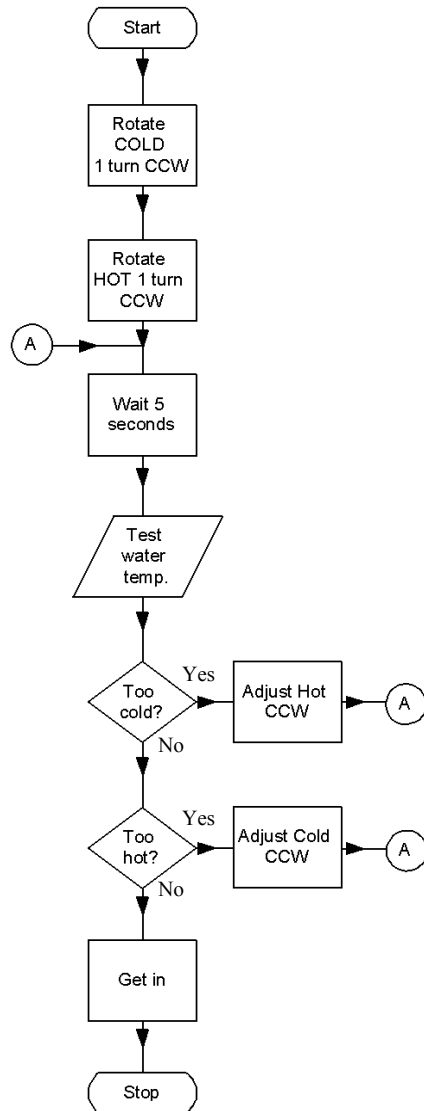
Sub-routine or sub-process box indicates the use of a defined routine or process.

### Example #1: Adjusting the Temperature of a Shower

Let's take an example flowchart of an everyday task: adjusting the temperature for a shower. The process of adjusting the water temperature has several steps involved. The water valves are initially opened, we wait a while for the temperature to stabilize, test it, and make some decisions for adjustments accordingly. If the water temperature is too cold, the hot valve is opened more and we go back to test it again. If the water is too hot, the cold valve is opened more. Once we make this adjustment, we go back to the point where we wait for a few seconds before testing again. Of course this doesn't take into account whether the valves are fully opened. Steps may be inserted during the temperature adjustment procedure to correct for this condition. Figure 1.2 shows a flowchart of this process.

This example demonstrates a process that may be used in adjusting the temperature, but could it also be the steps in a microcontroller program? Sure! The valves may be adjusted by servos, and the water temperature determined with a sensor. In most cases, a simple process we go through can be quite complex for a microcontroller. Take the example of turning a corner in a car. Can you list all the various inputs we process in making the turn?

Figure 1.1: Shower Temperature Example

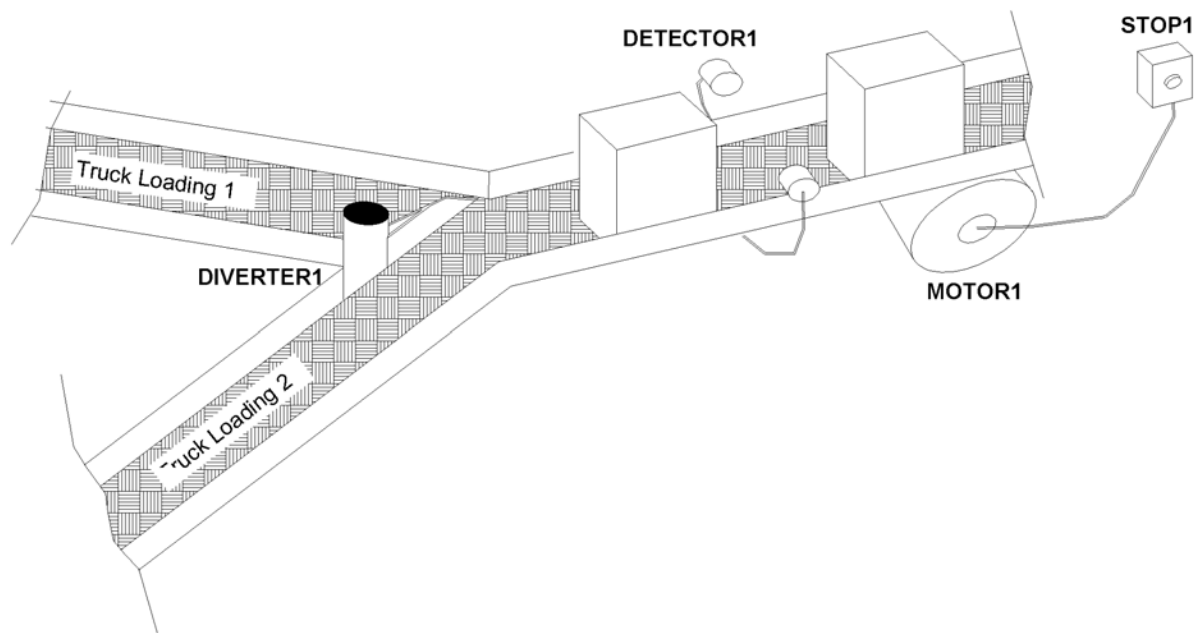


Example #2: Conveyor Counting Example

Let's look at a real scenario and develop a flowchart for it. In a manufacturing plant, items are boxed and sent down a conveyor belt to one of two loading bays with trucks waiting. Each truck can hold 100 boxes. As the boxes arrive, workers place them on the first truck. After that truck is full, the boxes must be diverted to the second truck so the loaded truck can be moved out and an empty one moved into position. Also, in the event of an emergency or problem, there must be a means of stopping the conveyor.

The physical aspects of the scenario are illustrated in Figure 1.2. The motor for the belt is labeled MOTOR1. The sensor to detect the boxes as they pass is labeled DETECTOR1. The lever to direct boxes to one truck conveyor or the other is labeled DIVERTER1. The emergency stop button is labeled STOP1.

Figure 1.2: Conveyor Counting Example



Let's list in order a brief description of what must occur:

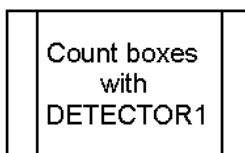
- Start the conveyor motor.
- Count the boxes as they pass.
- When 100 boxes have passed, switch the diverter to the opposite position.
- Whenever the emergency stop is pressed, stop the conveyor.

Now that we know the basic steps involved, let's develop a flowchart for the process. Let's begin by looking at the simple process flow in Figure 1.3 on the following page.

Notice the placement of the Input/Output box for checking the emergency stop button, STOP1. It ensures the button is tested during every cycle. What if we had placed it following the 100-count decision box? How long would it have taken from when the button was pushed until the conveyor stopped?

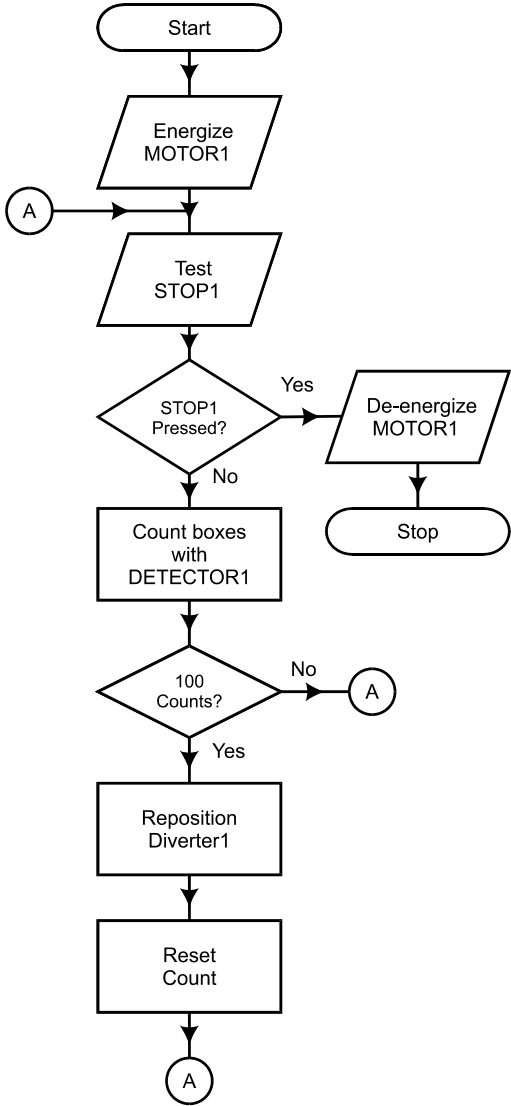
Does the flowchart describe everything our program needs to do? Definitely not, but it is a good start at determining the overall flow of the process. Look at the "Count Boxes with DETECTOR1" Process box. How exactly is this carried out? We may need to develop a flowchart to describe just this routine. If a process needs further detailing, we might replace the Process box with a Sub-Process box as shown in Figure 1.4.

Figure 1.4: Sub-Process Box



How involved is it to simply count a box passing by a detector? If DETECTOR1 is activated by "going low," do we count? When the detector stays low, how do we keep from recounting it again the next time our program passes that point? What if the box bounces on the conveyor as it enters our beam? How do we keep from performing multiple counts of the box? These answers may not be as simple as they seem. Even when performing a task as simple as counting a passing box, many variables must be taken into account.

Figure 1.3: Conveyor Counting Flowchart



Another consideration is the output of our detector. Can we directly measure the output using one of the BASIC Stamp inputs, or is there some circuitry needed to condition the signal first?

Let's consider an output in our conveyor counting example. How do we energize the motor? It is doubtful the 5-volt, milliamp-rated output of the BASIC Stamp will be able to drive a motor of sufficient horsepower to move a conveyor! How do we condition an output of the BASIC Stamp to control a higher voltage and current load?

These issues will be considered as you work through the chapters in this text. What may seem simple for us to do as humans may require some sophisticated algorithms for a microcontroller to mimic. We will use readily available electronic components, a BASIC Stamp module, and the Board of Education to simulate some complex industrial control processes.



## Exercises

### Exercise #1: Flowchart Design

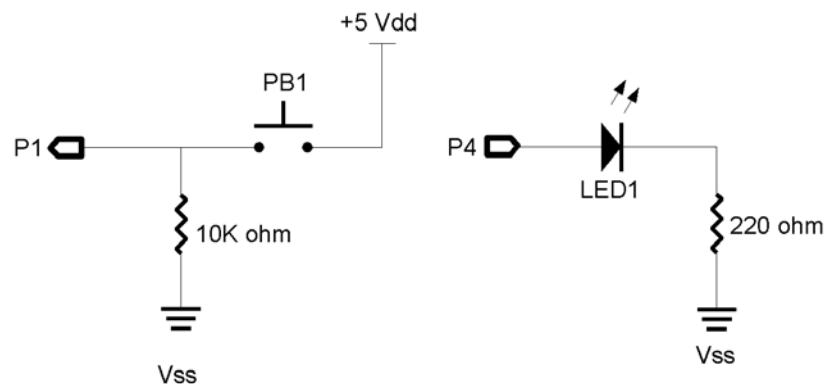
Develop a flowchart that will energize a heater below 100 degrees and de-energize it above 120 degrees.

### Exercise #2: LED Blinking Circuit

We'll use a simple circuit to demonstrate a flowchart process and the program to perform the task. You'll need to build the circuit shown in Figure 1.5. The following parts will be required for this experiment:

- (1) LED, green
- (2) 220-ohm resistors
- (1) 10K-ohm resistor
- (1) Pushbutton
- (1) 10K-ohm multi-turn potentiometer
- (1) 1 uF capacitor
- (Miscellaneous) jumper wires

Figure 1.5: Exercise #2 Blinking Circuit Schematic



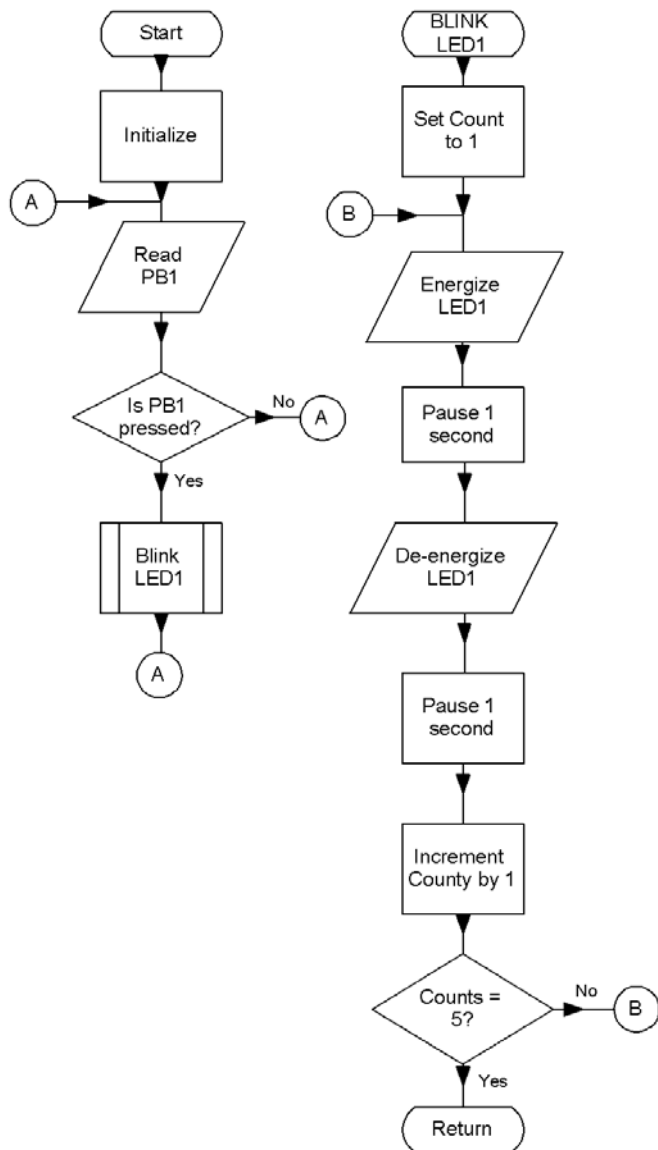


The circuit you are building consists of a single input button and a single output LED. Here is the process we want to perform: when the button (PB1) is pressed, blink the green LED (LED1) five times over 10 seconds. The flowchart for our process is shown in Figure 1.6.

Notice a few things about the flowchart. Our main loop is fairly simple. In the Initialize process box, we will define any variables needed and set initial outputs (LED off) and will loop unless PB1 is pressed, which calls our subroutine, `blink_led1`. Our subroutine doesn't begin with "Start," but the name of the process, so that we can identify it. The flowchart describes a process that we will repeat five times, alternately energizing and de-energizing our LED for one second each time.

Now that we have a flowchart to describe the process, how do we program it in PBASIC? Programmatically, we can sense PB1 using the `IN` statement. We have two ways we can call our subroutine. If the condition is true (1), then we can branch to our subroutine directly using an `IF-THEN` statement. This would be treated as a PBASIC `GOTO`. Once this completes, we would need to `GOTO` back to our main loop. Or, if the condition is false (0), we can branch back to our main loop from the `IF-THEN`, and use a `GOSUB` command to branch to our subroutine when true. We can then use a `RETURN` when our subroutine is complete.

Figure 1.6: Exercise #2 Blinking Circuit Flowchart



## Experiment #1: Flowcharting and StampPlot Lite

---

In our `blink_led1` subroutine, we need a loop to repeat five times. Choices for accomplishing this task may be to set up a variable we increment and check during each repetition, or use the **FOR-NEXT** statement to accomplish it for us.

The flowchart describes the general steps involved in accomplishing a process. The code required is flexible as long as it faithfully completes the process as described. The same flowchart may be used in multiple languages or systems and even for humans!

Program 1.1 is one way to write the code for our blinking LED process. Enter the text in the BASIC Stamp editor, download it to the BASIC Stamp, and press the pushbutton of the circuit you built. If it works properly, the LED will blink five times after the pushbutton is pressed.

```
'Program 1.1; Blinking LED Example
Cnt    VAR    BYTE    'Variable for counting
PB1    VAR    IN1     'Variable for PB1 input
LED1    CON    4      'Variable for LED1 output

INPUT 1          'Set PB1 as input
OUTPUT 4        'Set LED1 as output

LOW LED1        'Turn off LED

Start:
  IF PB1 = 0 then Start    'Not Pressed? Go back to loop
  GOSUB Blink LED1        'If it was pressed then perform subroutine
  GOTO Start              'After return, go back to start

Blink LED1:
  For Cnt = 1 to 5        'Subroutine to blink LED 5 repetitions
    HIGH LED1             'Setup loop for 5 counts
    PAUSE 1000            'Turn ON LED
    LOW LED1              'Wait 1 second
    PAUSE 1000            'Turn off LED
  NEXT                   'wait 1 second
                          'Repeat loop until done

RETURN            'return back to after gosub call
```

### Programming Challenge

Flowchart and program a process where the LED will blink four times a second while the pushbutton is NOT pressed!

### Exercise #3: Analog Data

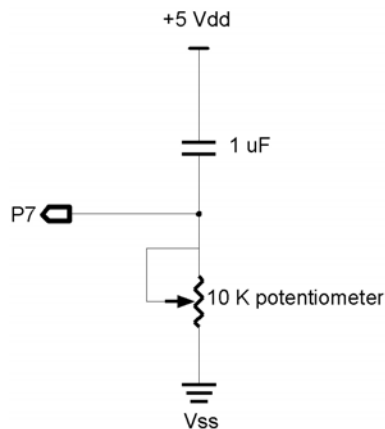
In many instances a process involves analyzing and responding to analog data. Digital data is simply on or off (1 or 0). This is comparable to the simple light switches in our homes. The light is on or it is off. Analog data on the other hand is a range of values. Some examples include the level of lighting if we use a dimmer switch instead of an on/off switch, or the temperature of the water coming out of our shower nozzle.

There are several methods to bring analog data into a microcontroller, such as using an analog-to-digital (A/D) converter that changes analog values into digital values that may be processed by the microcontroller. Another method used by the BASIC Stamp is a resistor/capacitor network to measure the discharge or charge time of the capacitor. By varying the amount of the resistance, we can affect and measure the time it takes the capacitor to discharge. In this experiment, resistance is set by manually adjusting a variable resistor. But the device may be more sophisticated, such as a photo-resistive cell that changes resistance depending on the amount of light shining on it, or a temperature sensor. More discussion on analog data is found in later sections, but for now let's perform a simple process-control experiment using an analog value.

Add the RC network shown in Figure 1.7 to your circuit from the previous experiment. It uses these parts:

- (1) 1 uF capacitor
- (1) 10K potentiometer

Figure 1.7: Schematic for Analog Data circuit added to Exercise #3



## Experiment #1: Flowcharting and StampPlot Lite

---

### PBASIC Command Quick Reference: RCTime

`RCTIME pin, state, resultvariable`

- `Pin` is the I/O pin connected to the RC network.
- `State` is the input voltage of that pin.
- `Resultvariable` is normally a word-length variable containing the results of the command.

The PBASIC command we will use to read the analog value of the potentiometer is **RCTIME**. A typical block of code to read the potentiometer is as follow:

```
Pot      VAR WORD
HIGH 7
PAUSE 10
RCTIME 7, 1, Pot
```

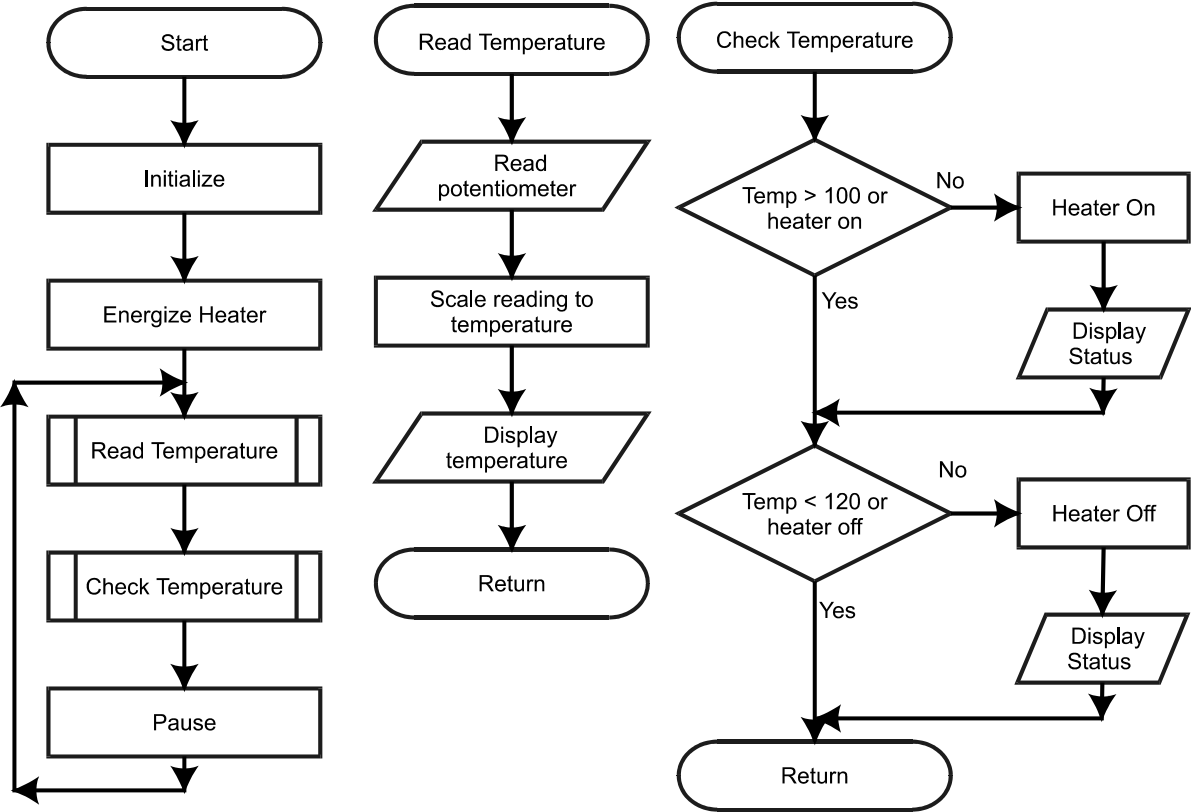
In order for the BS2 to read the potentiometer, the routine needs to take the following steps:

- +5V (HIGH) is applied to both sides of the capacitor to discharge it.
- The BASIC Stamp pauses long enough to ensure the capacitor is fully discharged.
- When **RCTIME** is executed, Pin 7 becomes an input. Pin 7 will initially read a high (1) because an uncharged capacitor acts as short.
- As the capacitor charges through the resistor, the voltage at Pin 7 will fall.
- When the voltage at Pin 7 reaches 1.4 V (falling), the input state is read as low (0), stopping the process and storing a value in `Pot` proportional to the time required for the capacitor to charge.

The greater the resistance, the longer the time required for a capacitor to discharge; therefore, the higher the value of `Pot`. In this manner, we can acquire an analog value from a simple input device.

Let's write a process-control program to make use of this input. Our process will be one where temperature is monitored and a heater energizes below 100 degrees and de-energized above 120 degrees. The potentiometer will represent a temperature sensor and the LED will represent the heater being energized. We will use the debug window to display our temperature and the status of the heater. The maximum potentiometer value, with this combination of resistor and capacitor, may reach 5000, so we will divide it by 30 to scale it to a more reasonable range. Figure 1.8 is the flowchart of the process.

Figure 1.8: Exercise 3 - Simple Heater Flowchart



## Experiment #1: Flowcharting and StampPlot Lite

---

Enter and run Program 1.2. Monitor the value in debug window while adjusting the potentiometer and note what occurs as the value rises above 120 and below 100.

```
'Program 1.2, Simple Heater
LED1  VAR    OUT4      'LED1 is on P4
RC    CON    7         'RC network is on Pin 7
Temp  VAR    WORD      'Pot is a variable to hold results

OUTPUT 4              'Setup LED as output
LED1 = 1              'Energize initially

Main:
  GOSUB ReadTemp      'Read pot value as temperature
  GOSUB CheckTemp     'check temp to setpoint
  PAUSE 250
  GOTO Main

ReadTemp
  HIGH RC              'Read Potentiometer
  PAUSE 10
  RCTIME RC, 1, Temp
  Temp = Temp/30      'Scale the results down,
                       'store as temperature

  DEBUG "Temp = ",dec Temp, CR
RETURN

CheckTemp:            'If Temp > 100, or heat already on,
                       'check if should be off
  IF (Temp > 100) OR (LED1 = 1) THEN CheckOff
  LED1 = 1             'If not, then energize and display
  DEBUG "The heater energized",CR

CheckOff:             'If Temp < 120 or heat is off already, all done
  IF (Temp < 120) OR (LED1 = 0) THEN CheckDone
  LED1 = 0             'if not, then energize and display
  DEBUG "The heater de-energized", CR

CheckDone:
RETURN
```

### Programming Challenge

Modify the process flowchart and program so the LED indicates an air conditioner cycling between 70 and 75 degrees.

### Exercise #4: Using StampPlot Lite

While the debug window for the BASIC Stamp is very useful for obtaining data and information from the BASIC Stamp, it can be difficult to visualize the data without careful scrutiny. Is the temperature increasing or decreasing? How quickly is it changing? At what point did the output change? What temperature is it cycling around?

Enter StampPlot Lite! StampPlot Lite (SPL) was specifically developed for this text. SPL accepts data from the BS2 in the same fashion the debug window does, only SPL interprets the data and performs one of 4 actions based on the structure of the data:

- A value is plotted on an analog scale in real time.
- A binary value starting with % is plotted as digital traces in real time.
- Strings beginning with ! are interpreted as instructions to control and configure SPL.
- Any other string is listed as a message at the bottom of SPL and optionally time-stamped.

A main rule of SPL is that each line must end in a carriage return (13 or CR).

Please review Appendix A for a more in-depth discussion of StampPlot Lite.

If you have not yet installed StampPlot Lite, install it on your computer by downloading it from <http://www.stampsinclass.com>. Double-click the setup button and install it in your designated directory.

Let's take another look at Program 1.2, our simple heater, but this time using StampPlot Lite to help visualize the process. Program 1.2 has been rewritten as Program 1.3 to utilize StampPlot Lite (bold lines are added/modified from program 1.2).

```
'Program 1.3; Simple Heater using StampPlot Lite
'Configure StampPlot Lite
PAUSE 500
DEBUG "!SPAN 50,150",CR           'Set span for 50-150
DEBUG "!TMAX 60",CR             'Set for 60 seconds
DEBUG "!PNTS 500",CR           '500 data points per plot
DEBUG "!TITL Simple Heater Control",CR 'Title the form
DEBUG "!SHFT ON",CR            'Allow plot to shift at max
DEBUG "!TSMP ON",CR
DEBUG "!PLOT ON",CR            'Enable plotting
DEBUG "!RSET",CR               'Reset Plot

LED1  VAR    OUT4           'LED1 is on P4
RC    CON    7              'RC network is on Pin 7
Temp  VAR    WORD           'Pot is a variable to hold results

OUTPUT 4                    'Setup LED as output
LED1 = 1                     'Energize initially
```

## Experiment #1: Flowcharting and StampPlot Lite

---

```
Main:
  GOSUB ReadTemp           'Read pot value as temperature
  GOSUB CheckTemp         'check temp. to setpoint
  PAUSE 250
GOTO Main

ReadTemp
  HIGH RC                 'Read Potentiometer
  PAUSE 10
  RCTIME RC, 1, Temp
  Temp = Temp/30
  'Scale the results down,
  'store as temperature
  DEBUG DEC Temp, CR      'Send temperature value
  DEBUG IBIN LED1,CR    'Send LED Status
RETURN

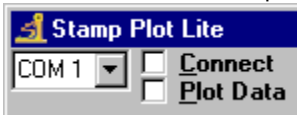
CheckTemp:
  'If Temp > 100, or heat already on,
  'check if should be off
  IF (Temp > 100) OR (LED1 = 1) THEN CheckOff
  LED1 = 1
  DEBUG "The heater energized",CR
  DEBUG "!USRS The heater is energized!",CR 'Update SPL status bar

CheckOff:
  'If Temp < 120 or heat is off, all done
  IF (Temp < 120) OR (LED1 = 0) THEN CheckDone
  LED1 = 0
  DEBUG "The heater de-energized", CR
  DEBUG "!USRS The heater is de-energized!",CR 'Update SPL Status Bar

CheckDone:
RETURN
```

Download this program to your BASIC Stamp, and follow these instructions to use StampPlot Lite.

- Start StampPlot Lite by using your Windows Start button and going to Programs/StampPlot/StampPlot Lite.
- Enter and run Program 1.3 on your BASIC Stamp.
- Close the BASIC Stamp editor's blue debug window.
- Select the correct COM port in StampPlot Lite and click 'Connect.'

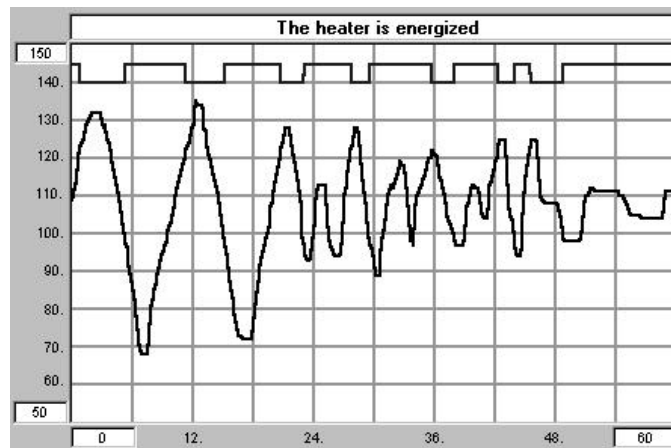


- Reset the BASIC Stamp by pushing the button on the Board of Education. Now you're ready to use this unique software utility.



At this point you should see data being plotted. Adjust the 10K-ohm potentiometer with your fingers or a small screwdriver. The analog line displays the value of the potentiometer. The digital trace at the top displays the status of the LED indicator. Figure 1.9 is a sample capture of the plot from our circuit.

Figure 1.9: StampPlot Lite Graph of Exercise #4



Note the correlation between the analog value and the switching of the digital output. Use the various controls on StampPlot Lite to become familiar with the functions and features. Analyze Program 1.3 and note the various configuration settings and data sent to the application. Refer to Appendix A for additional information on StampPlot Lite if you are having problems understanding the basics of the software utility.

### Programming Challenge

Modify your air conditioner challenge from Exercise #2 to use StampPlot Lite. Configure your program to transmit data approximately every 0.5 seconds. Calculate the number of data points needed to fill the screen within a maximum of 60 seconds, and test.

### Just for fun!

Enter and run the following program. The potentiometer simulates a single-handle shower (mixer) valve with adjustment delay. Adjust the shower temperature for a constant 110 degrees. See how fast you can stabilize the temperature at the set point! Press the reset button on the Board of Education and try again. We'll leave it up to you to figure out the program.

## Experiment #1: Flowcharting and StampPlot Lite

---

```
'PROGRAM 1.4: ADJUST THE SHOWER!
SetPoint      VAR BYTE
CurTemp      VAR BYTE
Diff          VAR BYTE
TempSet VAR WORD

RC            CON 7
LED1         CON 4
SetPoint = 110

PAUSE 500
DEBUG "!RSET",CR,"!SPAN 0,200",CR,"!TMAX 30",CR,"!PLOT ON",CR
DEBUG "!TSMP ON",CR,"!MAXS",CR,"!PNTS 100",13

DEBUG "!USRS ADJUST THE TEMP FOR ",DEC SetPoint,CR

Main:
  HIGH RC
  PAUSE 10
  RCTIME RC,1,TempSet
  TempSet = TempSet/ 30

  IF TempSet > CurTemp THEN Higher
  IF TempSet < CurTemp THEN Lower

GOTO Display

Higher:
  DIFF = TempSet - CurTemp/5
  CurTemp = CurTemp + Diff
GOTO Display

Lower:
  Diff = CurTemp - TempSet/5
  CurTemp = CurTemp - Diff

Display:
  LOW LED1
  DEBUG DEC CurTemp,CR
  IF CurTemp <> SetPoint THEN SkipBeep
  DEBUG "AT SETPOINT!",CR,"!BELL",CR
  HIGH LED1

SkipBeep:
  PAUSE 250
GOTO Main
```



## Questions and Challenge

1. List one everyday human process that involves a decision. List the steps in performing the process and the decisions needed to be made.
2. Develop a simple flowchart for the process in Question #1.
3. List an example of an electronics process in your home or school (such as that of an electric or microwave oven control, alarm clock, etc). Develop a simple flowchart to describe the process.
4. Develop the flowchart and code for the following process: The potentiometer simulates a temperature sensor. If the temperature exceeds 100 degrees, lock on the alarm (LED). Do not clear the alarm until the pushbutton is pressed.
5. Modify the program from Question #4 to use StampPlot Lite to display the temperature, alarm bit and status of the alarm.





**Experiment #2:  
Digital Input Signal  
Conditioning**

Process control relies on gathering input information, evaluating it, and initiating action. In industrial control, input information most often involves monitoring field devices whose outputs are one of two possible states. A switch is the most common example of a “bi-state” device. It is either open or closed.

Switches can provide control of an operation in three ways. One may be wired directly with the load and therefore control the full current and voltage. A switch also can be wired in the input circuit of a relay. In this case, the switch controls the relay’s relatively low power input and the output contacts control load power. The on/off status of a switch may also provide a digital input to a programmable controller.

How many switches have you used today? And, what processes were affected by the toggling of those switches? Table 2.1 lists a few possibilities, starting at the beginning of your day:

Table 2.1: Switch Possibilities at the Beginning of your Day

Switch Status	Result
First, you may slap the “SNOOZE” button on your alarm clock.	The buzzing stops and -- Ah! 5 more minutes of sleep!
Next, stumble to the bathroom and flip “ON” the bathroom light.	Ouch! Turn it “OFF.” Those vanity lights hurt!
Now, into the kitchen, start your coffeemaker, press down the toaster, and program your microwave. Open the refrigerator and the light comes on .	Breakfast is ready. And who knows if that light really goes off when you close the refrigerator?
Turn on the thermostat.	Heat or AC – your choice. What temperature? A setpoint is usually just a “switching point.”
Turn on your TV, change the channel, turn up the volume.	The pushbuttons on the front or the flashing infrared LED in your remote– they all still just switch data.
Make a phone call. Lift the receiver and check for dial tone. Key in the phone number .	The limit switch held down by the handset now is in its “off the hook” position. Each switch on the keypad allows a specific tone to be generated.
Boot your PC. Switch on the monitor. Left click the mouse to check your e-mail.	These are only three obvious ones. There are many more switches behind the scenes in your PC.
You are up to 15 switches and you haven’t even left your house!	

## Experiment #2: Digital Input Signal Conditioning

---

Some of the switches listed in Table 2.1 probably have direct control of electrical continuity to the loads involved. For example, the bathroom light switch controls the actual current flowing to the vanity light bulbs. The thermostat is an example of a switch controlling a low-voltage system that controls a relay in your furnace or air conditioner.

Most of the switches in Table 2.1, however, probably are providing a digital high or low signal being monitored by an electronic control system. It is the status of this input signal that is evaluated and used to determine the appropriate state of the outputs involved. The snooze button isn't physically opening the alarm circuit of your clock radio. When you "slapped" it, the momentary change of state was recognized by a programmable circuit. As a result, the program instructed the output to go off and add five minutes to the programmed alarm time. The start button on your microwave doesn't have to carry the actual current that powers the magnetron, inside light, and ventilation fan. However, pressing it creates an input causing the oven's microcontroller to close relays that do handle these loads.

Most often we think of switches as mechanical devices that make and break continuity between contact points in a circuit. In the case of the manual pushbutton and the limit switches pictured in Figure 2.1, this is exactly the case.

Figure 2.1: A Variety of Manual Pushbutton and Mechanical Limit Switches

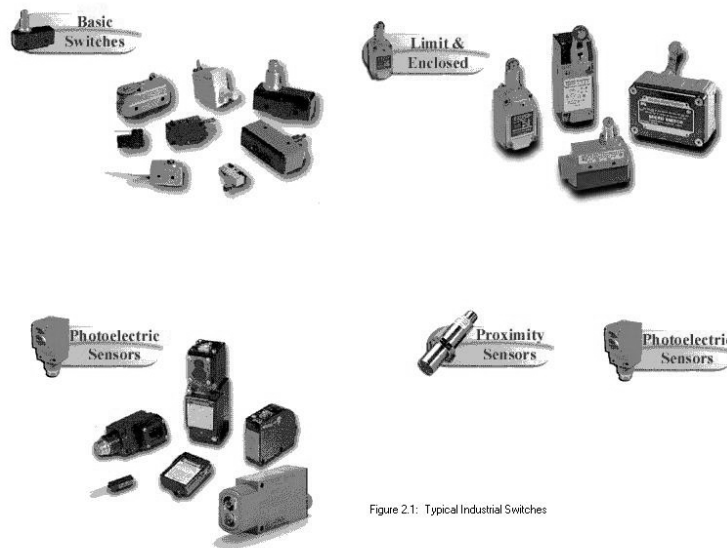


Figure 2.1: Typical Industrial Switches

Table 2.2 shows the schematic representation of various industrial switches. The symbols are drawn to represent the switch's "normal" state. Normal state refers to the unactuated or rest state of the switch. The pushbutton switches in this exercise kit are Normally Open (N.O.). Pressing the pushbutton results in a plunger shorting the contacts. The resistance goes from its open value of nearly infinite ohms to a value very near zero. A similar mechanism produces a like action in a Normally Open limit switch.

Table 2.2: Schematic Representation of Various Industrial Switches

	Push-button	Mechanical Limit	Proximity Switch	Relay Contacts
Normally Open				
Normally Closed				

While the concept of the switch is simple, there seems to be no limit to the physical design of switches that you will find in industrial control applications. Switches also may be designed as Normally Closed (N.C.); they are closed when at rest and actuation causes their contacts to open. As a technician, programmer, or system designer, you must be aware of the Normal (resting) position of a switch.

Figure 2.2: Schematic Representation of Pushbutton Switches

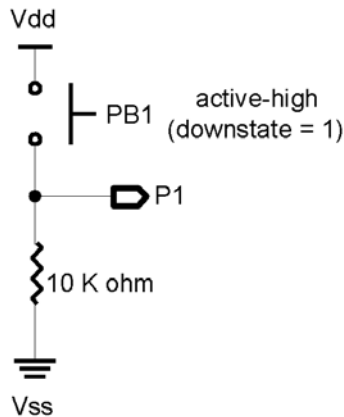


Figure 2.2a

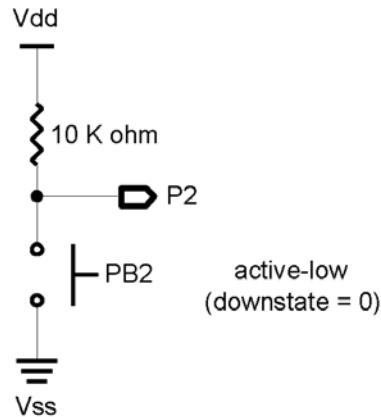


Figure 2.2b

### What's a...

#### Digital Input (TTL, CMOS, ECL, etc.)?

Logic devices are built with a variety of processes that operate at different voltages. The manufacturer's datasheet will list several critical values for each device. Absolute Maximum Ratings are voltages and currents which must not be exceeded to avoid damaging or destroying the chip. I/O pins on the BASIC Stamp II should not exceed 0.6 V or  $V_{dd} + 0.6$  V (5.6V) with respect to  $V_{ss}$ .

The logic transition between high and low is specified in the DC characteristics of the datasheet. A voltage of 0.2  $V_{dd}$  (1 V on the BASIC Stamp II) is guaranteed to be low, and which 0.45  $V_{dd}$  (2.25 V) or higher is guaranteed to be high. There is a gray area between these two voltages where the actual transition will occur. It is dependant on temperature and supply voltages where the actual transition will occur. It also varies with temperature and supply voltage but will normally occur at about 1.4 volts.

The input pins of the BASIC Stamp do not detect "changes in resistance" between the switch's contacts. These inputs expect appropriate voltage levels to represent a logic high or a logic low. Ideally, these levels would be +5 volts for a logic high (1) and 0 volts for a logic low (0).

To convert the two resistive states of the switch into acceptable inputs, it must be placed in series with a resistor across the +5 volt supply of the BASIC Stamp. This forms a voltage divider circuit in which the resistive status of the switch is compared to the resistive value of the reference resistor. Figure 2.2 shows the two possibilities for our simple N.O. pushbutton switch. Figure 2.2a will result in +5 volts being fed to the input pin when it is pressed. When the switch is open, there is no continuity; therefore, no current flows through the 10K resistor and the input pin is grounded.



### What's a...

#### Reference Resistor:

The 10K-ohm fixed resistor in Figures 2.2a and 2.2b is required to get dependable logic levels. It is wired in series with the switch. Its value must be much greater than the closed resistance of the switch and much less than its open resistance. When the switch is open in Figure 2.2a, the resistor gets no voltage and the input point is "pulled down" to ground. In Figure 2.2b, the open switch causes the input to be "pulled up" to +5 volts. You must consider the use of pull-up and pull-down resistors when working with all mechanical switches and some electronic switches.

In Figure 2.2b, the switch closure results in grounding of the input pin. Zero volts is a logic low. When the switch is opened, there is again no voltage drop across the 10K-ohm resistor and the voltage at the input is +5, a logic high. The circuits are essentially the same, although the results of pressing the switch are exactly opposite. From a programming standpoint, it is important to know with which configuration you are dealing.

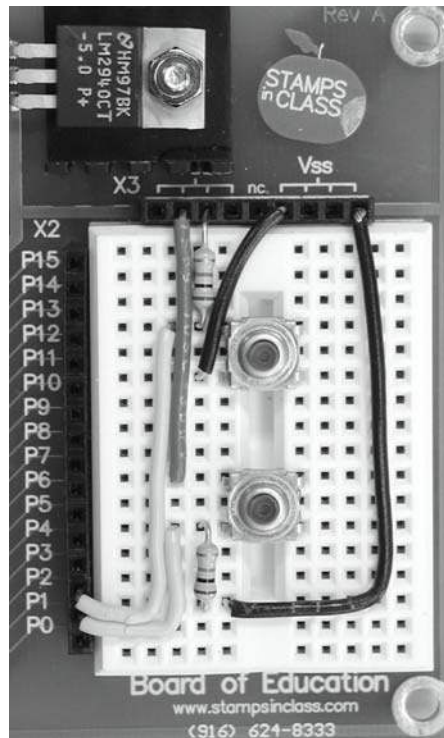


## Exercises

### Exercise #1: Switch Basics

To begin an investigation of programming for simple switch activity, wire the two pushbutton switches shown in Figure 2.2 onto the Board of Education breadboard. Connect the active-high configuration (Figure 2.2a) to I/O Pin 1 and the output of the active-low configuration (Figure 2.2b) to Pin 2. Note which one is which. As stated earlier, this is important. Figure 2.3 shows a pictorial of how the circuit is built on the Board of Education.

Figure 2.3: Pictorial of Parts Layout for circuits of Figure 2.2



The following program is written to use the StampPlot Lite interface for displaying the status of the switches. The procedure will be the same as you followed in Experiment #1, Flowcharting and StampPlot Lite. First, enter Program 2.1. You may omit from the program all comments which include the apostrophe (') and the text that follows.

```
'Program. 2.1:  Switch Level Detection with StampPlot Lite Interface

DEBUG "!TITL Pushbutton Test",CR          ' Titles the StampPlot screen

INPUT 1                                   ' Set P1 as an input
INPUT 2                                   ' Set P2 as an input
PB1 VAR IN1
PB2 VAR IN2

Loop:
  PAUSE 100                               ' Slow the program loop
  DEBUG IBIN PB1, BIN PB2, CR             ' Plot the digital status
  DEBUG DEC 0, CR                         ' Output a 0 to allow for screen shift
  IF (PB1 = 1) and (PB2 = 0) THEN Both    ' Test for both pressed
  IF PB1 = 1 THEN PB1_on                  ' Test if active-high PB1 is pressed
  IF PB2 = 0 THEN PB2_on                  ' Test if active-low PB2 is pressed
DEBUG "!USRS Normal states - Neither pressed", CR
                                           ' Report none pressed
GOTO Loop

PB1 on:
                                           ' Report PB1 pressed
  DEBUG "!USRS Input 1 is High - PB1 is pressed ", CR
GOTO Loop

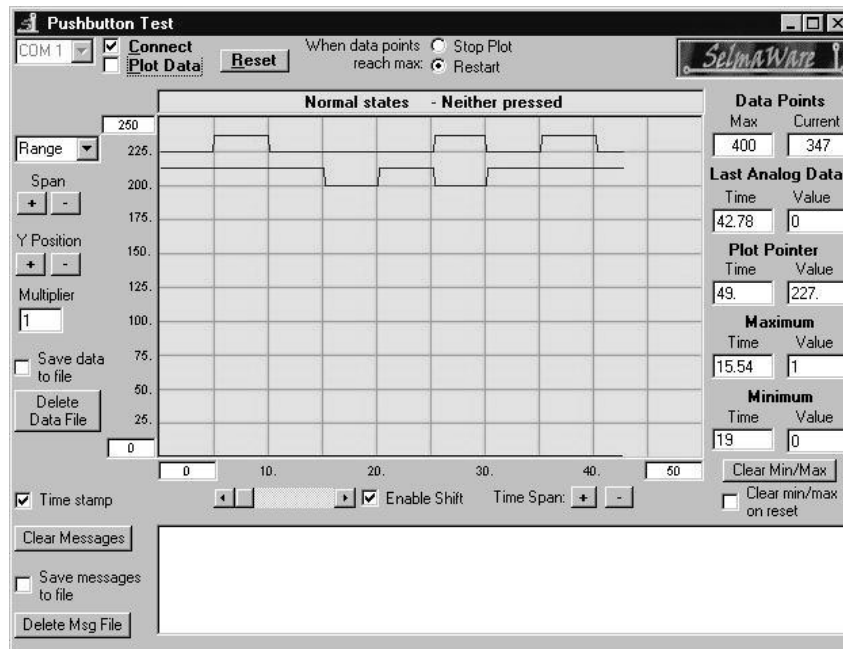
PB2 on:
                                           ' Report PB2 pressed
  DEBUG "!USRS Input 2 is Low - PB2 is pressed ", CR
GOTO Loop

Both:
                                           ' Report both pressed
  DEBUG "!USRS PB1 High & PB2 Low - Both pressed", CR
  DEBUG "!BELL", CR                       ' Sound the bell.
GOTO Loop
```

## Experiment #2: Digital Input Signal Conditioning

Run the program. **DEBUG** will scroll the switch status and the input's digital value. Close the debug screen and open StampPlot Lite. Select the appropriate COM port and check the Connect and Plot Data boxes. Press the reset switch on your Board of Education and the trace of **In1** and **In2** should start across the screen. Your display should look similar to Figure 2.4. Press the pushbuttons and become familiar with the operation of your system. Next, we will look at how the program works.

Figure 2.4: Typical Screen Shot of StampPlot Monitoring the Status of Pushbuttons



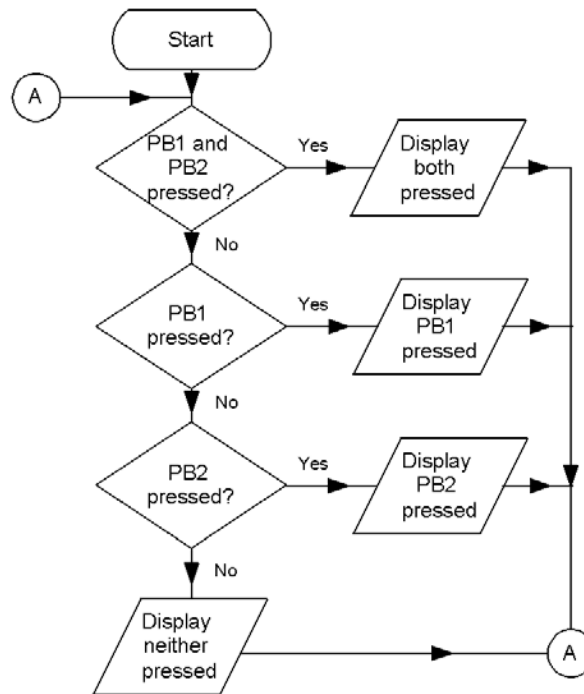
The purpose of this program is to run code based on the pressed or not-pressed condition of the two pushbuttons. This simple exercise gives insight to several considerations when dealing with digital inputs, programming multiple if-then statements, and using some of the PBASIC logical operators.

First, the statements **in1** and **in2** simply return the logic value of the input pins: +5 V = logic 1 and 0 V = logic 0. The active-high PB1 returns a 1 if pressed. The active-low PB2 returns a 0 when it is pressed. The program is testing for the “logical” status of the inputs; as the programmer, you must understand how this correlates to the “pressed” or “not pressed” condition of the pushbuttons involved. This is evident in the first line of the program loop where the logic operator **AND** is being used.

When you consider our switch configurations, it makes logical sense that if  $I_{n1}$  returns a logic high and  $I_{n2}$  returns a logic low then both switches are pressed. Output actions of industrial controllers often are dependent upon the status of multiple switches and contacts. A review of the PBASIC logical operators, including **AND**, **OR**, **XOR**, and **NOT**, can provide useful tools in meeting these requirements using the BASIC Stamp.

Another aspect of Program 2.1 is to notice the flow of the program loops. The **IF-THEN** structures test for a condition and **if** the condition is met, **THEN** the program execution is passed to the label. In this case, the label routine simply prints the conditions of the switches to the StampPlot Lite Status box. In industrial applications, this portion of the program would cause the appropriate output action to occur. Since the last line of each label is **GOTO Loop**, program execution returns to the top of the loop and any code below that **IF-THEN** statement is circumvented. The flowchart in Figure 2.5 shows how the program executes.

Figure 2.5: Flowchart for Program 2.1



## Experiment #2: Digital Input Signal Conditioning

---

If both switches are pressed, “**IF (PB1 = 1) and (PB2 = 0)**” is true. Program execution then would go to the **Both** label. The “both pressed” condition would be indicated in the User Status Bar and your computer bell would ring. After this, program execution is instructed to go back to **Loop** and test the switches again. As long as both switches remain pressed, the result of this test is continually true and looping is occurring only within this part of the program.

If either or both switches become not pressed, the next three lines of code will do a similar test for the condition.

Pressing PB1 results in “**IF PB1 = 1**” being true, execution is passed to the PB1 label action, and a return to the top of the loop; “**IF PB2 = 0**” is never tested. Is this good or bad? Neither, really. But, understanding the operation of multiple **IF-THEN** statements can be a powerful tool for programming applications. Forgetting this can result in frustrating and not-so-obvious bugs in your program. For instance, what would happen in our program if the test for both switches being pressed “**IF (PB1 = 1) AND (PB2 = 0) THEN Both**” was put after the individual switch tests?

### Quick Challenge

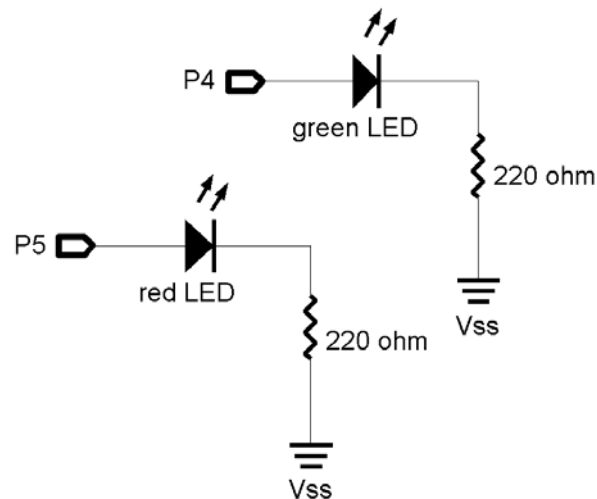
While running the program, try to reproduce the switch status shown in the screen shot of Figure 2.4.

Exercise #2 – Switch Bounce and Debouncing Routines

In the previous exercise, the steady-state level of the switch was being reported. The routine of reporting the switch status was performed on each program loop. What if you wanted to quickly press the switch and have something occur only once? There are two issues with which to contend. The first is: How quickly can you press and release the switch? You have to do it within the period of one program cycle. The second problem is contending with switch bounce. Switch bounce is the tendency of a switch to make several rapid on/off actions at the instant it is pressed or released.

The following program will demonstrate the difficulty in accomplishing this task. Two light-emitting diodes have been added as output indicators on Pin 4 and Pin 5. Wire the LEDs relative to Figure 2.6.

Figure 2.6: Active-High LED Circuit to be Added to the Schematic in Exercise #1



Enter and run the program according to StampPlot Lite procedures. The status of the pushbutton and the LEDs is being indicated. When PB1 is pressed, the LEDs will toggle. Can you be quick enough to make them toggle only once on alternate presses? Try it.

## Experiment #2: Digital Input Signal Conditioning

---

```
'Program 2.2 No Debouncing

PAUSE 500
DEBUG "!TITL Toggle Challenge",CR      ' Titles the StampPlot screen
DEBUG "!TMAX 25", CR                  ' Sets the plot time (seconds)
DEBUG "!PNTS 300", CR                 ' Sets the number of data points

INPUT 1                               ' Set P1 as an input
INPUT 2                               ' Set P2 as an input
OUTPUT 4                              ' Green LED
Out4 = 1                              ' Initialize ON
OUTPUT 5                              ' Red LED
Out5 = 0                              ' Initialize OFF

Loop:
  DEBUG IBIN In1, BIN In4, BIN In5, CR ' Plot the digital status.
  DEBUG DEC 0, CR                      ' Output a 0 to allow for screen shift
  IF In1 = 1 THEN Action               ' Test the switch
                                        ' Optional pause 5 if StampPlot locks up
GOTO Loop

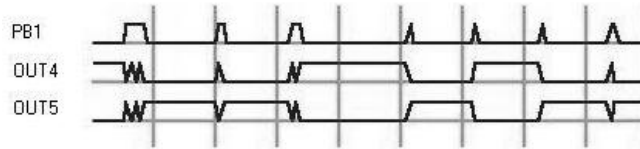
Action:                               ' Toggle last state
  TOGGLE 4
  TOGGLE 5
GOTO Loop
```

If StampPlot Lite isn't responding to data sent by the BASIC Stamp, you may need to insert a very short delay in the `Loop:` routine. A `PAUSE 2` or `PAUSE 5` (even up to 10 on slower computers) will alleviate any transmission speed problems you may encounter.

It is nearly impossible to press and release the pushbutton fast enough to perform the action only once. The problem is twofold as Figure 2.7 indicates. The program loop executes very fast. If you are slow, the program has a chance to run several times while the switch is closed. Add to this several milliseconds of switch bounce, and you may end up with several toggles during one press.



Figure 2.7: Slow Response and No Debounce Can be a Problem

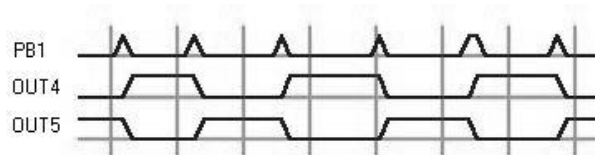


Further slowing the execution time of the program loop can help remedy the problem. (If the above program didn't work properly with StampPlot Lite, a delay in execution speed will allow for serial data transmission). Add a delay of 250 milliseconds to the **Action:** routine. This allows 250 milliseconds for the switch to settle after closing and then return to its open position.

Modify your program to include "PAUSE 250" to increase the loop time and negate switch bounce.

```
'Program 2.3 (modify program 2-2 to slow it down)
Action:          ' Toggle last state
  TOGGLE 4
  TOGGLE 5
  PAUSE 250      ' Added to allow for settling time
GOTO Loop
```

Figure 2.8: Adding a Pause Makes the Toggle Challenge Much Easier



By allowing settling time and pressing the button quickly, you make it much easier to get the **Action:** to occur only once. This technique helps debounce the switch and gives you enough time to release it before the next program cycle. The **PAUSE** must be long enough to allow for these factors. If the **PAUSE** is too long, however, a switch closure may occur and never be seen.

## Experiment #2: Digital Input Signal Conditioning

---

### Exercise #3 – Edge Triggering

Counting routines pose additional problems for digital input programming. Exercise #2 used the **PAUSE** command to eliminate switch bounce, which is compounded in industrial applications such as counting products on a conveyor. Not only does the switch have inherent bounce, but the product itself may have irregular shape, be wobbling, or stop for some time while activating the switch. There may be only one product, but the switch may open and close several times. Also, if the one product stays in contact with the switch for several program loop cycles, the program still should register it only once, not continually like in Program 2.2.

Program 2.4 uses a flag variable to create a program that responds to the initial low-to-high transitions of the switch. Once this “leading edge” of the digital input is detected, **Action:** will be executed. Then the flag will be set to prevent subsequent executions until the product has cleared and the switch goes low again. Enter Program 2.4.

```
' Program 2.4: Switch Edge Detection
' Count and display the number of closures of PB1.
' Reset total count with a closure of PB2.

PAUSE 500
DEBUG "!TITL Counting Challenge",CR           ' Titles the StampPlot screen
DEBUG "!TMAX 50",CR                          ' Sets the plot time (seconds)
DEBUG "!PNTS 300",CR                         ' Sets the number of data points
DEBUG "!AMAX 20",CR                          ' Sets vertical axis (counts)
DEBUG "!MAXR",CR                             ' Reset after reaching max data points

INPUT 1
INPUT 2
PB1 VAR In1
PB2 VAR In2
Flag1 VAR bit                               ' flag for PB1
Flag2 VAR bit                               ' flag for PB2

COUNTS VAR word                            ' word variable to hold count
Flag1 = 0                                    ' clear the flags and Counts
Flag2 = 0
COUNTS = 0

Loop:
  PAUSE 50
  DEBUG "!USRS Total Count = ",DEC Counts,CR
  ' Display total counts in Status box
  DEBUG DEC Counts, CR                      ' Show counts on analog trace
  DEBUG IBIN PB1, BIN PB2,CR              ' Plot the digital status.

  IF PB1 = 1 THEN Count it                  ' If pressed, count and display
  Flag1 = 0                                ' If not pressed, reset flag to 0
```

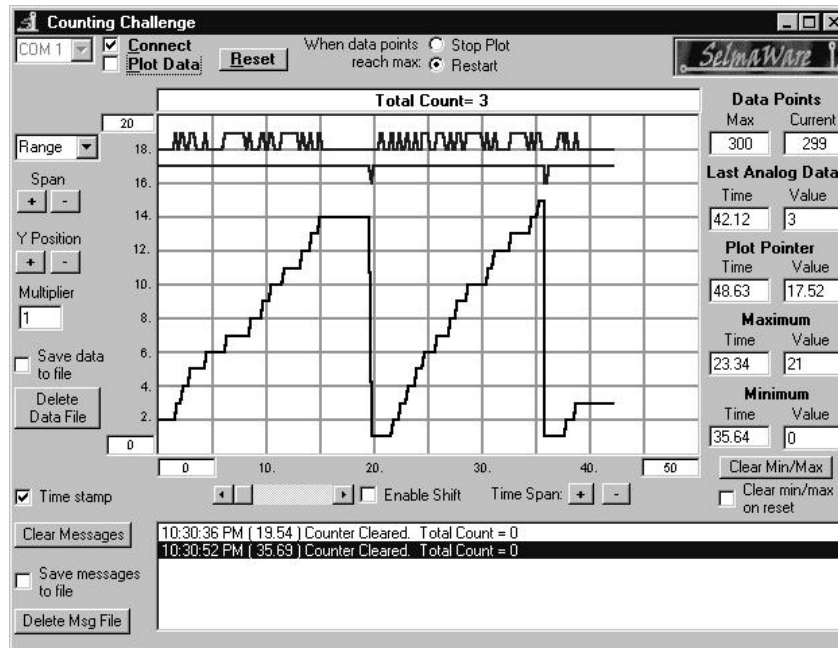
```
IF In2 = 0 THEN Clear_it      ' If PB2 is pressed, clear counts to 0
Flag2 = 0
GOTO Loop

Count_it:
  IF (PB1 = 0) OR (Flag1 = 1) THEN Loop      ' If no longer pressed OR the
                                              ' flag is set, skip
  Counts = Counts +1                        ' Increment Counts
  Flag1 = 1                                ' Once Action executes, set Flag to 1
GOTO Loop

Clear_it:
  IF(In2 = 1) OR (Flag2 = 1) THEN Loop      ' If no longer pressed,Or the flag is
                                              ' set, skip
  Counts = 0                                ' Clear counts to 0
  Flag2 = 1                                ' Prevents from clearing it again
  DEBUG "Counter Cleared. Total Count = ", DEC Counts, CR
GOTO Loop
```

When PB1 is pressed, the program branches to the `Count_it` routine. Notice that the first line of this routine tests to see if the switch is open or `Flag1` is set. Neither is true upon the first pass through the program. Therefore, `Counts` is incremented, `Flag1` is set to 1 and program execution goes back to `Loop`. If PB1 still is being held down, `Count_it` is run again. This time, however, with `Flag1` set, the `IF-THEN` statement sends the program back to `Loop` without incrementing `Counts` again. No matter how long the pushbutton is pressed, it will only register one “count” upon each closure. Although you are only incrementing the `Count` variable in this program, it could be part of a routine called for in an industrial application. Figure 2.9 is a screen shot that is representative of what you may see when running the program.

Figure 2.9: Running Program 2.4 - Edge Trigger Counting



### Programming Challenge 1: The Parking Lot.

Use the indicating LEDs on output Pins 4 and 5, along with the two pushbuttons, to simulate a parking lot application. Assume your parking lot can hold 24 cars. Pushbutton PB1 will be counting cars as they enter the lot. Pushbutton PB2 will count cars as they leave. Write a program that will keep track of the total cars in the parking lot by counting “up” with PB1 and “down” with PB2. Have the green LED on as long as there is a vacancy in the lot. Turn the red LED on when the lot is full. Continually display how many parking spaces are available in the User Status window (!USRS). Plot continually the number of cars in the parking lot.

### Additional StampPlot Lite Challenge

Keep a file of the number of times your parking lot went from “Vacancy” to “Full” (see Appendix A and the StampPlot Lite help file for information on using the Save Data to File option).

BUTTON Command: PBASIC's Debouncing Routine

Debouncing switches is a very common programming task. Parallax built into the PBASIC2 instruction set a command specifically designed to deal with digital input signal detection. The command is called `button`. The syntax for the command is shown below.

## PBASIC Command Quick Reference: BUTTON

*BUTTON pin, downstate, delay, rate, bytevariable, targetstate, address*

- *Pin:* (0- 15) The pin number of the input.
- *Downstate:* (0 or 1) Specifying which logical state occurs when the switch is activated.
- *Delay:* (0-255) Establishes a settling period for the switch. Note: 0 and 255 are special cases. If delay is 0, Button performs no debounce or auto-repeat. If delay is 255, Button performs debounce but no auto-repeat.
- *Rate:* (0-255) Specifies the number of cycles between autorepeats.
- *Bytevariable:* The name of a byte variable needed as a workspace register for the BUTTON instruction.
- *Targetstate:* The state of the pin on which to have a branch occur.
- *Address:* The label to branch to when the conditions are met.

To try it with our counting routine, load and run program Program 2.5.

```
' Program 2.5: Button Exercise with StampPlot Interface
' Use Button to count and display the number of closures of PB1.
' Reset total count with a closure of PB2.

PAUSE 500
DEBUG "!TITL Counting Challenge",CR           ' Titles the StampPlot screen
DEBUG "!TMAX 50",CR                          ' Sets the plot time (seconds)
DEBUG "!PNTS 300",CR                         ' Sets the number of data points
DEBUG "!AMAX 20",CR                          ' Sets vertical axis (counts)
DEBUG "!MAXR",CR                             ' Reset after max data points is reached

Wkspace1 VAR byte                            ' Workspace for the BUTTON command for PB1
Wkspace1 = 0                                 ' Must clear workspace before using BUTTON
Wkspace2 VAR byte                            ' Workspace for the BUTTON command for PB2
Wkspace2 = 0                                 ' Must clear workspace before using BUTTON

Counts VAR word                              ' Word variable to hold count
Counts = 0

Loop:
  PAUSE 50
  BUTTON 1,1,255,0,Wkspace1,1,Count_it      ' Debounced edge trigger detection of PB1
  BUTTON 2,0,255,0,Wkspace2,1,Clear_it      ' Debounced edge trigger detection of PB2
```

## Experiment #2: Digital Input Signal Conditioning

---

```
DEBUG "!USRS Total Count = ", DEC Counts, CR
      ' Display total counts in Status box
DEBUG DEC Counts, CR          ' Show counts on analog trace
DEBUG IBIN In1, BIN In2, CR   ' Plot the digital status.
GOTO Loop

Count_it:
  Counts = Counts +1          ' Increment Counts
GOTO Loop

Clear_it:
  Counts = 0                  ' Clear counts to 0
  DEBUG "Counter Cleared. Total Count = ", DEC Counts, CR
      ' Display in Text Box
GOTO Loop
```

Review the documentation concerning the **BUTTON** command in the BASIC Stamp Programming Manual Version 1.9. This is a very handy command for industrial applications. Experiment by changing the delay time from 50 to 100 and to 200. See if you can press the switch more than one time but only get one **Action** to take place. What would be the risk of allowing for too much settling time in “high speed” counting applications? Save this program; it will be modified only slightly for use with the next programming challenge.

### Electronic Digital Input Sources

It is very common for digital inputs to come from the outputs of other electronic circuits. These inputs may be from a variety of electronic sources, including inductive or capacitive proximity switches, optical switches, sensor signal-conditioning circuits, logic gates, and outputs from other microcontrollers, microprocessors, or programmable logic control systems.

There are several things to consider when interfacing these sources to the BASIC Stamp. Primarily: “Are they electrically compatible?”

1. Is the source’s output signal voltage within the BASIC Stamp input limits?
2. Is the ground reference of the circuit the same as that of the BASIC Stamp?
3. Is protection of either circuit from possible electrical failure of the other a concern such that isolation may be necessary?

Figure 2.10 shows a variety of electrical interfacing possibilities you may face.

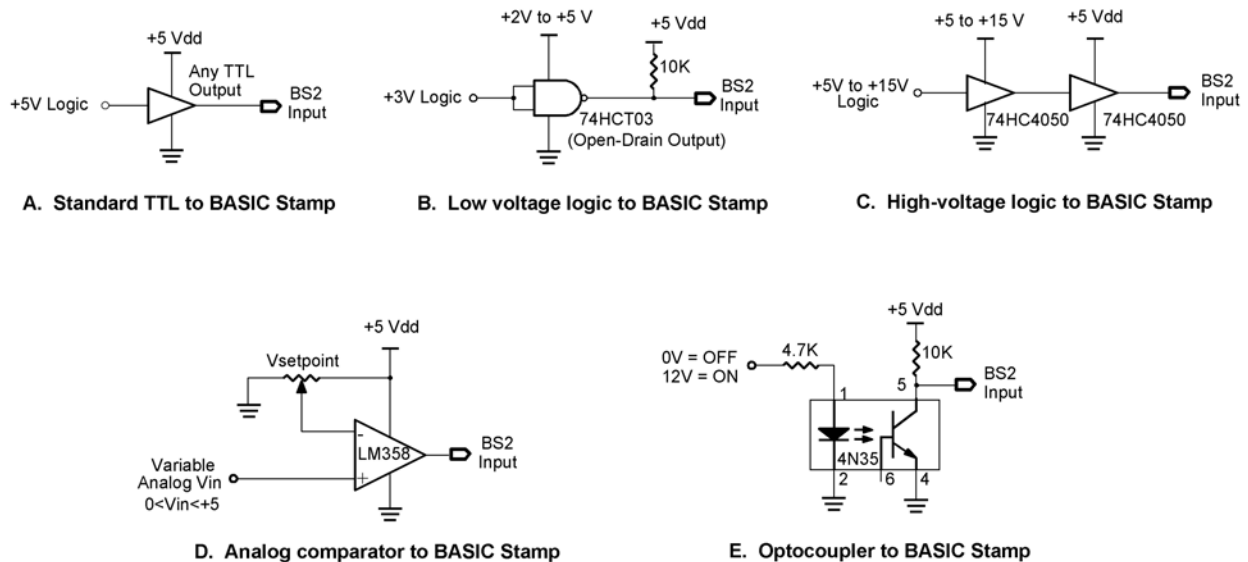
Once a compatible signal is established, the next question becomes, “Is the program able to respond to the signal?”

1. Is digital bounce an issue?
2. How fast is the data? What is its frequency? What is the minimum pulse time?
3. Is action to be taken based on the data’s steady-state level or on its leading or trailing edges?

Techniques to deal with switch bounce and edge triggering that were discussed relative to the manual pushbuttons also apply to the electronic switch.

## Experiment #2: Digital Input Signal Conditioning

Figure 2.10: Input Interfacing of Electronics to the BASIC Stamp



- TTL and CMOS logic inputs powered from a +5-volt supply can be applied directly to the BASIC Stamp's input pins. If the two systems are supplied from the same 5 volts, great. If not, at least the grounds must be common (connected together).
- Low-voltage (+3 V) devices can be interfaced using a 74HCT03 or similar open-drain gate with a pull-up resistor to the BASIC Stamp's +5-volt supply. Supply the chip with the low-voltage supply and make the grounds common.
- Higher-voltage digital signals can be interfaced using a 74HC4050 buffer or 74HC4049 inverter powered at +5 volts. These devices can safely handle inputs up to 15 volts. Again, the grounds must be common.
- A referenced comparator op-amp configuration can establish a High/Low output based on the analog input being above or below the setpoint voltage. The LM358 is an op-amp whose output will go from ground to nearly Vdd on a single-ended, +5-volt supply. It will be used in the upcoming application.
- An opto-coupler may be used to interface different voltage levels to the BASIC Stamp. The LED's resistor holds current to a safe level while allowing enough light to saturate the phototransistor. The input circuit can be totally isolated from the phototransistor's BASIC Stamp power supply. This isolation provides effective protection of each circuit in case of an electrical failure of the other.

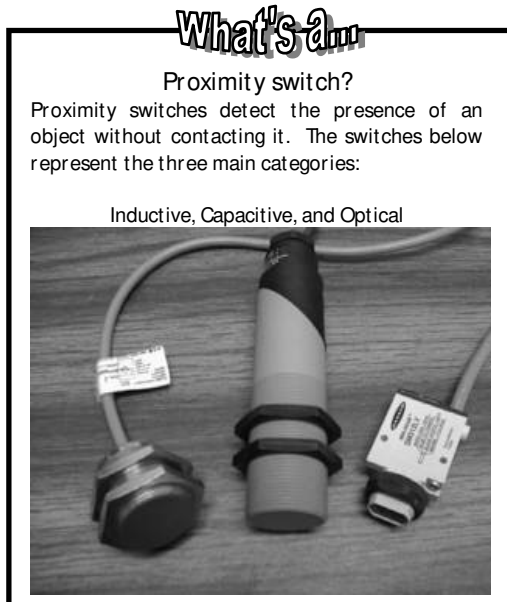


Exercise #4: An Electronic Switch

Electronic switches that provide “non-contact” detection are very popular in industrial applications. No physical contact for actuation means no moving parts and no electrical contacts to wear out. The pushbutton switch used earlier should be good for several thousand presses. However, its return spring eventually will fatigue, or its contacts will arc, oxidize, or wear to the point of being unreliable.

Industrial electronic switches operate on one of three principles.

- Inductive proximity switches sense a change in an oscillator’s performance when metal objects are brought near it. Most often the metal objects absorb energy via eddy currents from the oscillator causing it to stop.
- Capacitive proximity switches sense an increase in capacitance when any type of material is brought near them. When the increase becomes enough, it causes the switch’s internal oscillator to start oscillating. Circuitry is then triggered and the output state is switched.
- Optical switches detect the presence or absence of a narrow light beam, often in the infrared range. In retroreflective optical switches, the light beam may be reflected by a moving object into the switch’s optical sensor. Through-beam optical switches are set up such that the object blocks the light beam by going between the light source and the receiver.



The output of an electronic switch is a bi-state signal. It’s final stage may be any one of the types seen in Figure 2.10. As a technician and application developer, you must consider the nature of this signal circuit and condition it for the digital input of the microcontroller. The manufacturer’s datasheet will give you information on the operating voltage for the switch and typical load connections.

Although you can think of the BASIC Stamp’s digital input pin as the load, the electronic switch may require a reference resistor as used earlier in Figure 2.2. Most likely, the output of the proximity switch will be very near 0 volts in one state and near its supply voltage in the other state. It is always a good idea to test the switch’s output states with a voltmeter before applying it to the unprotected input of the microcontroller. If the output voltages are not within the compatible limits of the

## Experiment #2: Digital Input Signal Conditioning

---

BASIC Stamp, you will need to use one of the circuits in Figure 2.10 as an appropriate interface.

The following exercise focuses on the design and application of an optical switch. We will use this switch to detect and count objects. Then the switch will be used as a tachometer to determine RPM.

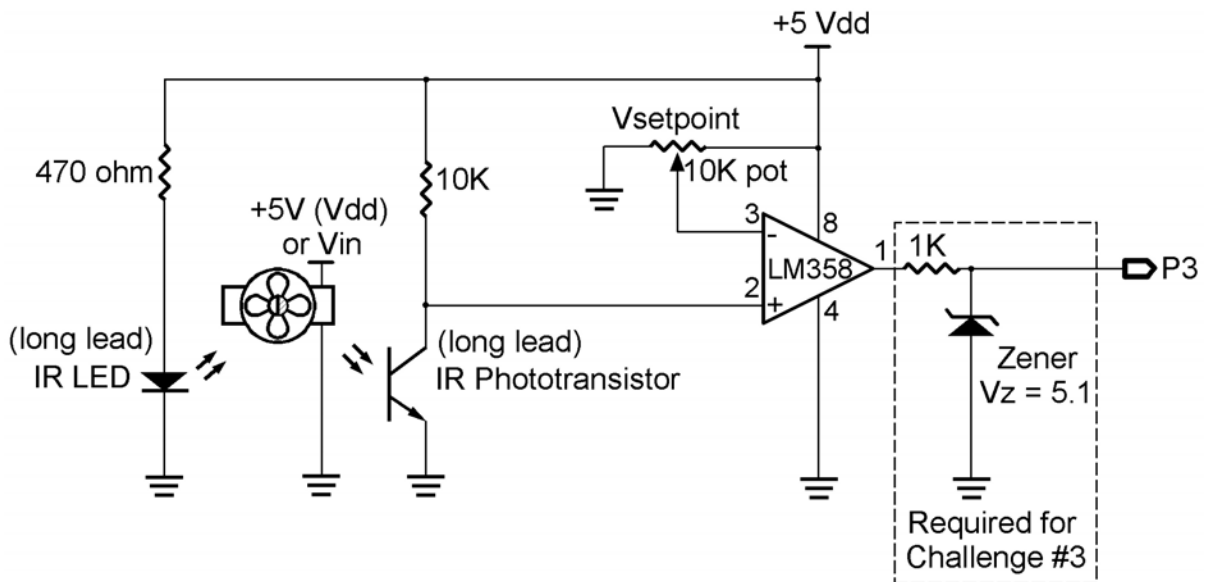
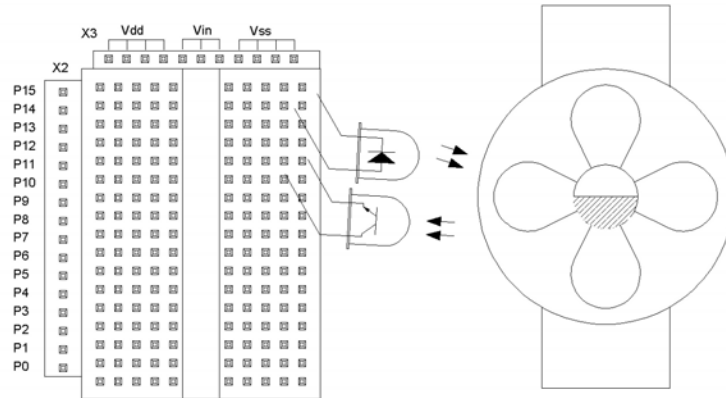
In Figure 2.11, the infrared light-emitting diode (LED) and the infrared phototransistor form a matched emitter/detector pair. Light emitted by the LED will result in phototransistor collector current. An increase in collector current drives the phototransistor toward saturation (ground). If the light is prevented from striking the phototransistor, it goes toward cutoff and the collector voltage increases positively. These conditions of light and no-light will most likely not provide a legal TTL signal at the collector of the transistor. Applying this signal to the input of a referenced comparator will allow us to establish a setpoint somewhere between the two conditions. The output of the comparator will be a compatible TTL logic signal. Its level is dependant on which side of the setpoint the phototransistor's output is on. The LM358 op-amp is a good choice for this application. It can operate on a +5-volt single supply and its output saturation voltages are almost equal to the supply potentials of +5 and ground.

Carefully construct the circuit in Figure 2.11 on the Board of Education breadboard. Mounting the devices near one end as pictured in the diagram allows for additional circuits in upcoming exercises. Make a 90° bend in the LED and phototransistor leads so the devices lie parallel to the the benchtop. The phototransistor and infrared LED should be placed next to each other, pointing off the edge of the breadboard.

The LED in Figure 2.11 is emitting a continuous beam of infrared light. With the LED and phototransistor side-by-side, there is little or no light coming into the phototransistor because there is nothing reflective in front of it. If an object is brought toward the pair, some of the LED light will bounce back into the phototransistor. When light strikes the phototransistor, the collector current will flow and the collector voltage will drop. In this setup, the scattered reflection of light off an object as it passes in front of the pair will be sensed by the phototransistor. The amount of reflected light into the sensor depends on the optical reflectivity of the target object and the geometry of the light beam. We will attempt to determine the presence of a flat-white object. With the emitter and detector mounted side-by-side, you will try for detection of the object at a distance of one inch.

You must make a couple of voltage measurements to calibrate the presence of the object. Begin by placing a voltmeter across the phototransistor's collector and emitter. Measure the voltage when there is no object in front of the sensor. Record this value in Table 2.3. Next, move a white piece of paper toward and away from the pair and notice the variation in voltage. As the paper is brought near the IR pair, the reflected light increases collector current and drives the transistor toward saturation –“low.” Record the voltage reading with the white paper approximately one inch in front of the sensor in Table 2.3. The difference between these measurements may be quite small, like 0.5 V, but that will be enough to trigger the op-amp. This signal is applied to the inverting input of the LM358 comparator. The potentiometer provides the non-inverting input reference voltage. This reference should be a value between the “no reflection” and “full reflection” readings.

Figure 2.11a and b: Retro-reflective Switch Pictorial and Schematic



## Experiment #2: Digital Input Signal Conditioning

---

Adjust the potentiometer to provide the proper reference voltage, which is halfway between the measurements.

Testing the output of the LM358 should result in a signal compatible with the BASIC Stamp. The output should be low with no object and high when the white object is placed in front of the emitter/detector pair. Measure these two output voltages of the LM358 and record the values in Table 2.3. If the output signal is compatible, apply it to the BASIC Stamp's Pin 3. Detecting light reflected by an object is called retro-reflective detection.

Table 2.3: LM358 Values

Condition	Phototransistor Voltage	LM358 Output Voltage
No object – no reflection		
Object – full reflection		
Reference voltage setpoint		

This ability to yield a switching action based on light received lends itself to many industrial applications such as product counting, conveyor control, RPM sensing, and incremental encoding. The following exercise will demonstrate a counting operation. You will have to help, though, by using your imagination.

Let's assume that bottles of milk are being transferred on a conveyor between the filling operation and the case packer. Cut a strip of white paper to represent a bottle of milk. Passing it in front of our switch represents a bottle going by on the conveyor. Only a slight modification of the previous program is necessary to test our new switch. If you have Program 2.5 loaded, simply modify the first `button` instruction by changing the input identifier from Pin 1 to 3. The modified line would look like this:

```
' Program 2.6 (modification to Program 2.5
' for the retroreflective switch input)

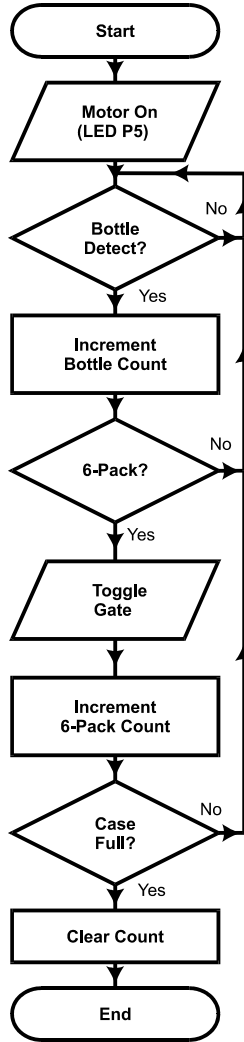
BUTTON 3,1,255,0,Wkspacel,1,Count it
' Debounced edge trigger detection of optical switch
```

### Programming Challenge #2: Milk Bottle Case Packer

Refer back to Experiment #1 and consider the conveyor diverter scenario in Figure 1.2. We will assume that the controller is counting white milk bottles. Our retroreflective switch detector could replace the "Detector 1" switch in the original figure. The active high PB1 would toggle the conveyor motor ON and OFF. The LED on P4 can indicate that the motor is ON by lighting up. The LED on P4 is controlling the diverter gate. When high the gate is to the right and when low, the gate is to the left. Your challenge is to start the motor with PB1 and count bottles as they pass. Each sixth bottle, **TOGGLE** the diverter gate's position as indicated

by the ON and OFF status of the LED on P4. After a case (4 six-packs) have been diverted to each side, turn off the motor. The process would start over by pressing the pushbutton again. Refer to Flowchart 2.12 to gain an understanding of the program flow.

Figure 2.12: Flowchart of Milk Bottle Challenge



## Experiment #2: Digital Input Signal Conditioning

---

### Exercise #5: Tachometer Input

Monitoring and controlling shaft speed is important in many industrial applications. A tachometer measures the number of shaft rotations in a unit of time. The measure is usually expressed in revolutions per minute (RPM).

A retroreflective switch can open and close fast enough to count white and black marks printed on a motor's shaft. Counting the number of closures in a known length of time provides enough information to calculate RPM. Figure 2.13 represents five possible encoder wheels that could be attached to the end of a motor shaft. If the optical switch is aimed at the rotating disk, it will pulse on-off with the alternating segments as they pass. The number of white (or black) segments represent the number of switch cycles per revolution of the shaft. The first encoder wheel has one white segment and one black segment. During each revolution, the white segment would be in front of our switch half the time, resulting in a logic high for half the rotation.

During the half rotation the black segment is in front of the disk, it absorbs the infrared light and with no reflected light, the switch will be low. One cycle of on-off occurs each revolution. The PBASIC2 instruction set provides a very useful command called `COUNT` that can be used to count the number of transitions at a digital input occurring over a duration of time. Its syntax is shown below.

#### PBASIC Command Quick Reference: COUNT

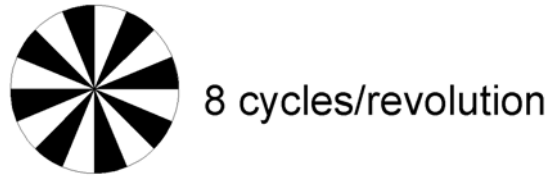
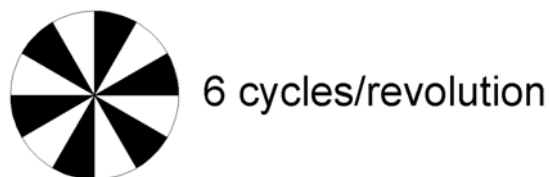
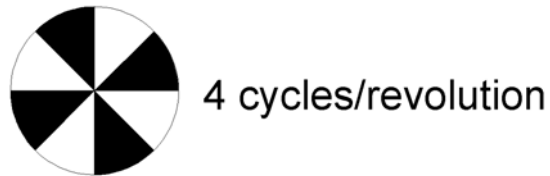
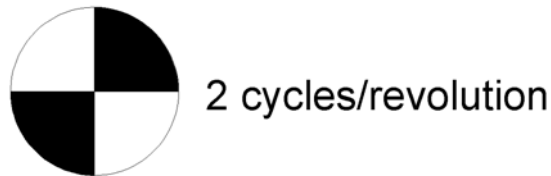
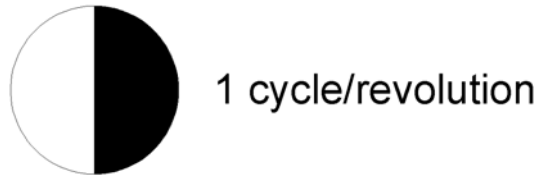
*COUNT pin, period, variable*

- *Pin: (0-15) Input pin identifier.*
- *Period:(0- 65535) Specifies the time in milliseconds during which to count.*
- *Variable: A variable in which the count will be stored.*

The following exercise uses the count instruction, the optical switch, and the shaft encoder wheels to capture speed data.

Lets begin by cutting out the first encoder wheel. Fold a piece of cellophane tape onto the back of the encoder wheel to hold it on the shaft hub of the fan motor (a full-size set of encoder wheels may be pulled from Appendix B of this text). The fan is rated at 12 V. Its speed changes with varying voltages from 12 V down to approximately 3.5 V. This is the dropout voltage of the brushless motor control circuitry. Test your fan by directly connecting it across the Vdd (+5 volt supply) and then test it across the +Vin (unregulated) supply. Pin 20 of connector X1 provides access to the unregulated supply (Vin). You must observe the poalarity on brushless motors. The red lead is positive (+V) and the black lead is connected to Vss. The fan should be located so the encoder wheel is pointed at the emitter/detector pair.

Figure 2.13: Retro-reflective Encoder Wheels  
(cutouts are available in Appendix B)



## Experiment #2: Digital Input Signal Conditioning

---

The first encoder wheel has one white and one black segment on it. As it rotates, the opto-switch should cycle on-off once for each revolution. Enter the Tachometer Test Program 2.7 below.

```
' Program 2.7 Tachometer Test - with the StampPlot Interface

' Initialize plotting interface parameters.
' (Can also be set or changed on the interface)
DEBUG "!AMAX 8000",CR          ' Full Scale RPM
DEBUG "!AMIN 0",CR            ' Minimum scaled RPM
DEBUG "!TMAX 100",CR         ' Maximum time axis
DEBUG "!TMIN 0",CR           ' Minimum time axis
DEBUG "!AMUL 1",CR           ' Analog scale multiplier
DEBUG "!PNTS 600",CR         ' Plot 600 data points
DEBUG "!PLOT ON",CR          ' Turn plotter on
DEBUG "!RSET",CR             ' Reset screen

Counts VAR word                ' Variable for results of count
RPM VAR word                   ' Variable for calculated RPM
Counts = 0                     ' Clear Counts

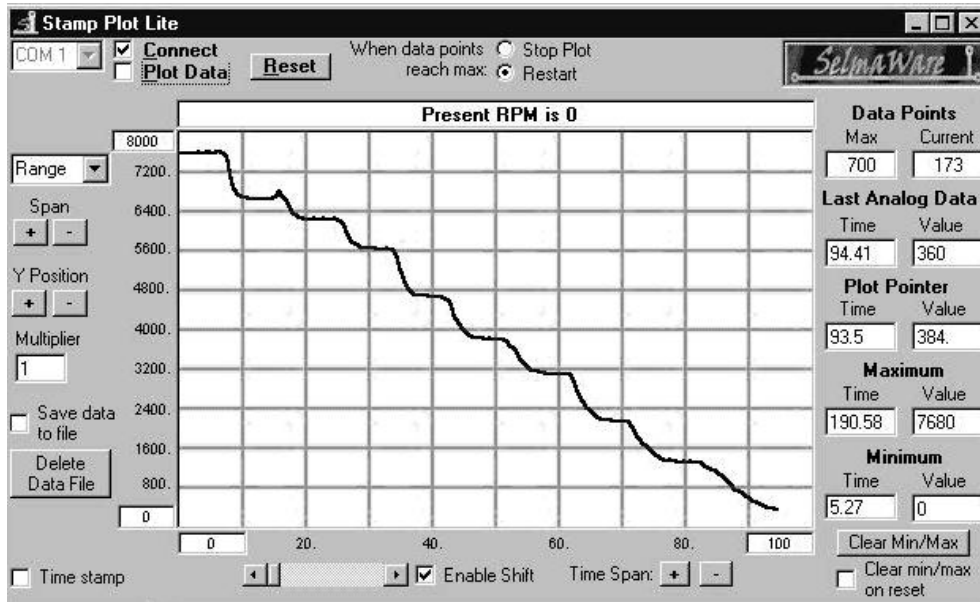
Loop:
  COUNT 3,1000, Counts         ' Count cycles on pin 3 for 1 second
  RPM = Counts * 60            ' Scale to RPM
                               ' Send out RPM value to plotter and status bar
  DEBUG DEC rpm, CR
  DEBUG "!USRS Present RPM is ", DEC RPM, CR
GOTO Loop
```

As the fan spins, the cycling of the photo switch will be counted for 1000 milliseconds (one second). With the duration of the `Counts` routine being one second and one cycle occurring with each rotation, we get the cycles per second of fan rotation. Most often, the speed of a rotating shaft is described in terms of revolutions per minute (RPM). Multiplying the revolutions per second times 60 converts cycles per second to RPM.

Run your program. The debug window will first appear with the serial information for configuration and display of the StampPlot Lite interface. Close the BASIC Stamp debug window and open StampPlot Lite. Check "Connect and Plot Data" and click on "Restart." Press the Board of Education reset button and your interface should start plotting. Figure 2.14 shows a representative screen shot of the interface plotting RPM at various motor voltages.

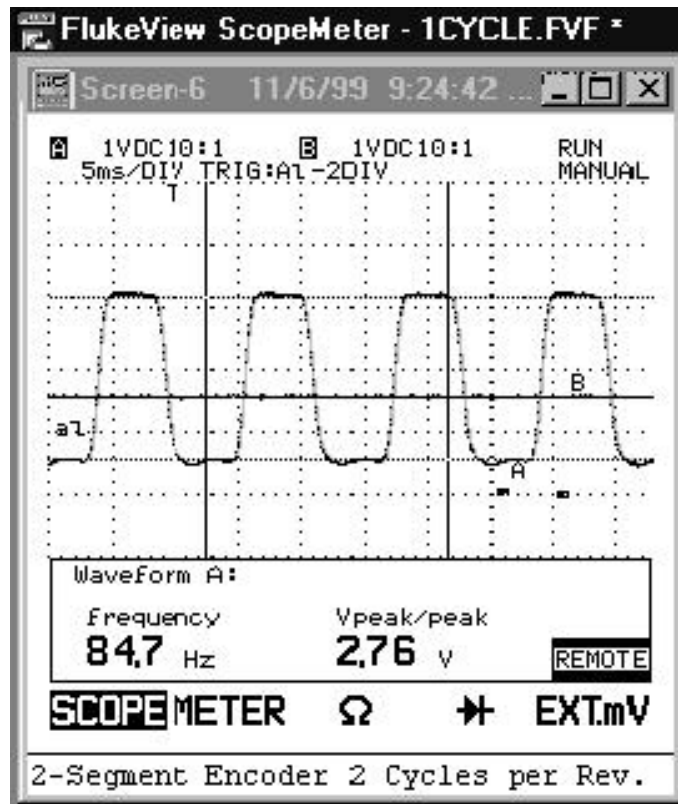


Figure 2.14: RPM of the Brushless DC Fan at Varying Voltages



The spinning encoder wheel may result in a slightly different phototransistor peak output for “light” and “no-light” conditions. If your system is not reporting correctly, change the setpoint by adjusting the potentiometer to the new average value. If you have access to an oscilloscope, measure the peak-to-peak output of the phototransistor and your potentiometer setpoint being applied to the comparator. Placing the setpoint midway between the peak-to-peak DC voltage levels would allow for optimal performance. Notice the frequency and wave shape of the signal. An example of the oscilloscope reading is pictured in Figure 2.15. The 84.7 Hz equated to a debug readout of “Counts = 84 RPM = 5040.” The 84.7 Hz measured by the oscilloscope reflects an actual RPM of  $84.7 \times 60 = 5,082$ . Only 84 complete cycles fell within the one-second capture time of our routine.

Figure 2.15: Two-Segment Encoder Oscilloscope Trace



Record your tachometer readout when the maximum voltage is applied to the motor. You can use the Board of Education's  $V_{in}$  (unregulated 9 V) for high speed, or the  $V_{dd}$  (regulated 5 V) for different speeds.

Counts = \_\_\_\_\_ RPM = \_\_\_\_\_

When testing your tachometer, notice the effects of slowing the motor with slight pressure from your finger. The counts will decrease by factors of one. In the Figure 2.13 example, it would decrease from 83 to 82 to 81, etc., and the resulting RPM readings drop by a factor of 60 (4980 to 4920 to 4860, etc.).

Because we are counting for one second and we get one cycle per revolution, the program can resolve RPM only to within an accuracy of 60. To get a more accurate assessment of RPM, you have a couple of choices: increase the time you count cycles, or increase the cycles per revolution.

Let's try the first choice. Increase the count time in Program 2.7 from 1000 milliseconds to 2000 milliseconds. By doing so, you are now reading during a two-second window and RPM would equal  $\{(Counts/2 \text{ seconds}) \times 60\}$ . This simplifies to  $RPM = 30 * Count$  and the resolution is now to within 30 RPM. In program 2.7, change the line `RPM = Counts * 60` from the scaling value of 60 to 30. Test your system.

Increasing the count duration time increases the accuracy of the RPM reading. Refer to Table 2.4.

Table 2.4: Given Encoder Frequency of 84.7 Hz  
From the 1 cycle/second Encoder is an RPM of 5082

Duration	Counts	Scaler	RPM	Resolution
1000 mS	84	60	5040	60 RPM
2000 mS	169	30	5070	30 RPM
3000 mS	254	20	5080	20 RPM
60000 mS	5082	1	5082	1 RPM

As you can see, to gain a resolution of one RPM, our count routine had to be one full minute (60,000) in duration. Unless you are very patient, this is unacceptable! In terms of programming, the BASIC Stamp is tied up with the `COUNT` routine for the total duration. During this time, the rest of the program is not being serviced. For this reason, long duration also is not good.

Another method of improving resolution is to increase the number of cycles per revolution. Cut out the second encoder wheel and tape it to your fan motor hub. This wheel has two white segments and will produce two count cycles per revolution. During a one-second-count duration, this encoder will produce twice as many pulses as the first encoder. The RPM calculation line of the code would be `RPM = Counts x 60 / 2` for this encoder, or `RPM = Counts x 30`. Try it!

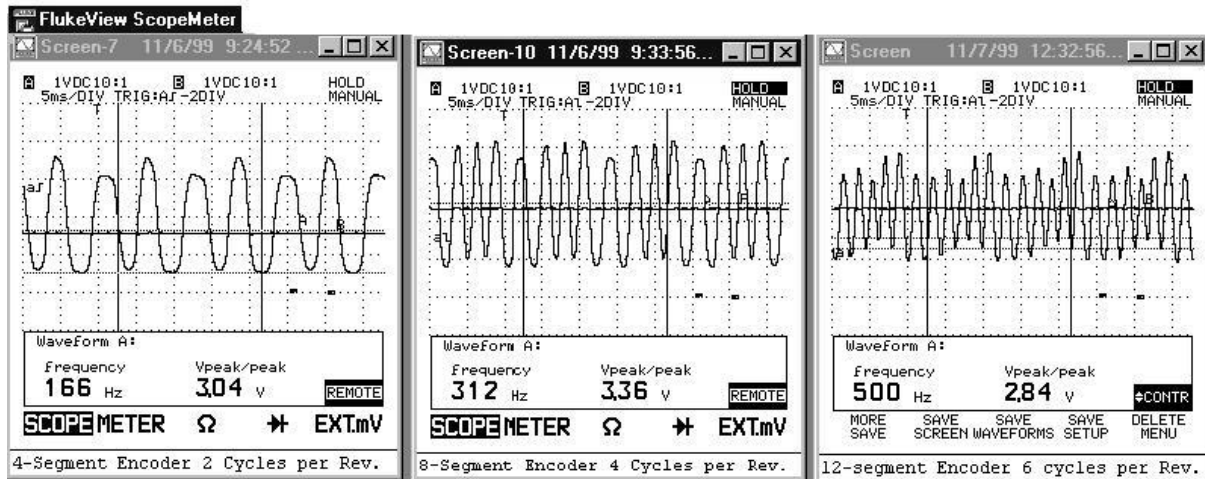
The third encoder wheel yields even more resolution by with four cycles per revolution. Tape this encoder to your motor's hub and change the program's RPM line to `RPM = Counts * 15`. You may have to vary the setpoint potentiometer as you switch from one encoder wheel to another.

## Experiment #2: Digital Input Signal Conditioning

If you use the six-cycle encoder, what value would you use to scale the Counts to RPM? Fill in your answer in Table 2.5.

Figure 2.16 includes oscilloscope traces recorded from using the two-cycle, four-cycle, and six-cycle encoder wheels on a shaft rotating at 4,980 RPM. It is the focal properties of the emitter/detector pair that will limit the maximum number of segments on the encoder wheel. You may find it difficult to use the six-cycle encoder wheels without devising some sort of shielding and/or focusing of the light beam.

Figure 2.16: Two-cycle, Four-cycle, and Six-cycle Encoder Wheel Oscilloscope Traces



The accuracy required of a tachometer system is dependent on the application. Commercial shaft encoders are available with resolutions greater than 500 counts per revolution. Fill in the appropriate values in Table 2.5 for an encoder with a resolution of 360 counts per revolution.

Table 2.5: Given a Shaft Speed of 4,980 RPM

Cycles per Revolution	Counts	Scaler	RPM
1	83	60	4,980
2	166	30	4,980
4	312	20	4,980
6	498	_____	4,980
360	_____	_____	_____

Challenge #3: Monitor and Control Motor Speed.

Varying the voltage applied to the small brushless motor varies its speed. The BASIC Stamp does not have a continuous analog output. The pulse-width modulation (PWM) command allows the BASIC Stamp to generate a controllable average analog voltage.

The syntax of PWM is shown below.

## PBASIC Command Quick Reference: PWM

*PWM Pin, Duty, Cycles.*

- *Pin*: specifies the output pin which is driven.
- *Duty*: is a value between 0 and 255 that expresses the average analog output between 0 and 5 volts.
- *Cycles*: is a value between 0 and 255 that specifies the duration of the PWM signal in milliseconds.

The command PWM 7,190,30 will produce at output pin 7 a series of pulses whose average high time is .75 (190/255) for a duration of 30 milliseconds. For this time, the average voltage at the pin is  $.75 * 5$  or 3.5 volts. To deliver this average voltage throughout the duration of a program loop, a sample and hold circuit must be developed. Figure 2.17 is a sample and hold circuit that will work well for the brushless fan. Capacitor  $C_{hold}$  charges during the PWM command to the average voltage. At the end of the Cycle time, PWM changes the direction of the output pin to an input. This places the pin in a high impedance condition and the charge on the capacitor is held due to the high impedance of pin 7, the dielectric of the capacitor, and the input to the op amp. The op amp is set to a gain of 3 by the  $Rf/Rin$  network ( $A_v = Rf/Rin + 1$ ). The output of this amplifier drives transistor Q1. It provides current boost for the majority of the load current. Ideally, a charge could be held indefinitely. Small capacitor leakage currents and op amp bias currents result in slight variations in

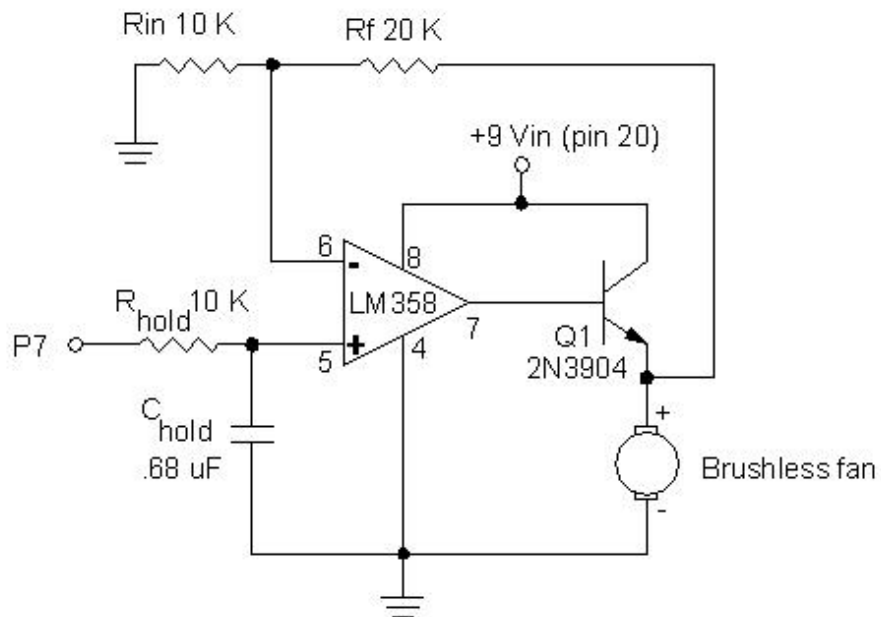
## Experiment #2: Digital Input Signal Conditioning

---

voltage between PWM commands. Usually the bias currents dominate and result in a slight rise in voltage during this time.

Figure 2.17 is designed around the second op amp in your LM358 package. Carefully add this circuit to your tachometer circuit on the Board of Education. Note that the supply voltage to the op amp is changed to the 9 volt unregulated supply. This allows this circuit to have a voltage output that will approach 14 volts. Your tachometer op amp comparator will also have a higher output. Note: It is imperative that the zener diode in Figure 2.11 is in place to clamp the input to P3 at 5 Volts. Your BASIC Stamp is at risk if voltages exceed 5V.

Figure 2.17: Brushless fan with sample and hold PWM drive.



### Testing the Sample and Hold

The fan's electronics requires 4 to 5 volts to operate. The voltage applied to the fan will be approximately equal to:  $(5V * \text{Duty}/255)*3$ . According to this equation, voltages from 4 to 12 will be produced by Duty values of 70 to 210. Replace "Duty" with values from this range in the following program. Use a voltmeter and Table 2.6 to record the voltage applied to the fan for the values of "Duty" listed.

```
'Program 2.8 Sample and Hold Test
Loop:

PWM 7, (Duty), 60

PAUSE 500

GOTO Loop
```

Duty	Voltage	Duty	Voltage
60			160
80			180
100			200
120			220
140			254

Next, the tachometer Program 2.7 will be modified to include your PWM with the Sample and Hold circuit. Recall that this program reported motor RPM by accumulating pulse counts from the encoder wheel over a period of 1 second. Without Sample and Hold, the motor would come to a stop during this one-second off period. Program 2.9 includes these modifications in bold print. The program will operate the motor at Duty Cycle increments between 70 and 210. Each increment will be tested for approximately 5 seconds. StampPlot will report the steps in the status box and plot the RPM continually. The formula for calculating the expected voltage assumes that the circuit is following the transfer function discussed earlier. You may modify this formula to better fit your circuit based on the tests performed in Program 2.8.

Modify program 2.7 as indicated below (additions are shown in bold). Run the program and record the speed voltage characteristics in Table 2.6.

```
'Program 2.9 (Modified Program 2.7 Tachometer Test - with the StampPlot Interface)

' Initialize plotting interface parameters.
' (Can also be set or changed on the interface)
DEBUG "!AMAX 8000",CR          ' Full Scale RPM
DEBUG "!AMIN 0",CR           ' Minimum scaled RPM
DEBUG "!TMAX 150",CR         ' Maximum time axis
DEBUG "!TMIN 0",CR          ' Minimum time axis
DEBUG "!AMUL 1",CR           ' Analog scale multiplier
DEBUG "!PNTS 600",CR        ' Plot 600 data points
DEBUG "!PLOT ON",CR         ' Turn plotter on
DEBUG "!RSET",CR            ' Reset screen

Counts VAR word              ' Variable for results of count
```

## Experiment #2: Digital Input Signal Conditioning

---

```
RPM VAR word           ' Variable for calculated RPM
Counts = 0             ' Clear Counts

OUTPUT 7              ' Declare the PWM pin.
x VAR word            ' Duty variable
y VAR byte            ' Iterations per Duty value
Tvolts VAR word

Loop:
  FOR x = 70 TO 210   ' Duty variable
  FOR y = 0 TO 5     ' Test a Duty value for 5 seconds

    PWM 7, x, 50     ' Deliver PWM at a Duty of x
    Tvolts = 50 * x / 255 * 3 ' Calculate voltage in tenths of a volt.

  COUNT 3,1000, Counts ' Count cycles on pin 3 for 1 second
  RPM = Counts * 60    ' Scale to RPM
                        ' Send out RPM value to plotter and status bar

  DEBUG DEC rpm, CR
  DEBUG "!USRS Duty = ",DEC x, " RPM is ",DEC RPM," at " ,DEC Tvolts," Tvolts", CR
    NEXT
    x = x + 9
  NEXT
  END
GOTO Loop
```



	Duty	Voltage	RPM
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			

StampPlot Lite will plot the speed voltage characteristics of your motor. Use the mouse cursor to read the stable RPM at each step on the plot. Summarize the response of the motor to changes in voltage.



## Questions and Challenge

### Questions

1. An industrial device whose output is either one of two possible states is termed \_\_\_\_\_.
2. What is the “ideal” resistance of a mechanical switch in the open state? In the closed state?  
Open-state resistance = \_\_\_\_\_ and, Closed-state resistance = \_\_\_\_\_
3. Explain the purpose of placing a resistance in series with a switch for conditioning a digital input signal.
4. A normally-open pushbutton switch configured in an “active low” state will be read as a logic \_\_\_\_\_ when not being pressed.
5. What is the absolute maximum input voltage to the BASIC Stamp?
6. For some CMOS devices, an input of 1.3 volts is in the \_\_\_\_\_ area of operation.
7. Low-voltage logic devices operate on \_\_\_\_\_ volts DC.
8. What type of proximity switch activates only on metal objects?
9. When light strikes the base of a phototransistor, the collector current will \_\_\_\_\_ and collector to emitter voltage will \_\_\_\_\_.
10. A car’s six-cylinder engine RPM can be determined by counting the pulses delivered to the ignition coil. Six pulses are required for one revolution. If 20 pulses occur in one second, what is the RPM of the engine?

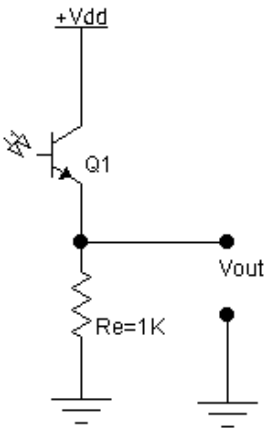


Experiment #2: Digital Input Signal Conditioning

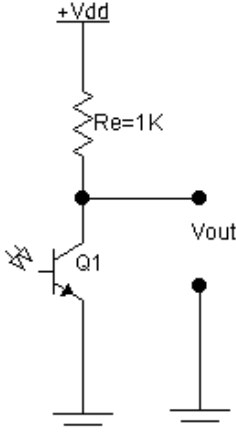
---

Analyze it!

- 1. Consider the two phototransistor circuits below. Which one has an increasing output voltage with increases in light level? Why? What is the output voltage of Circuit B if the light level saturates transistor Q1?

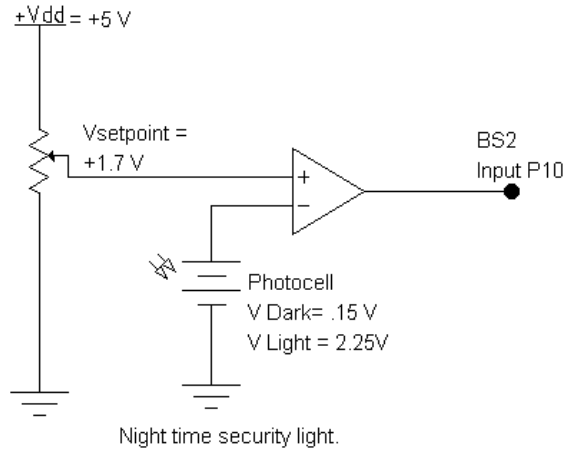


Circuit A

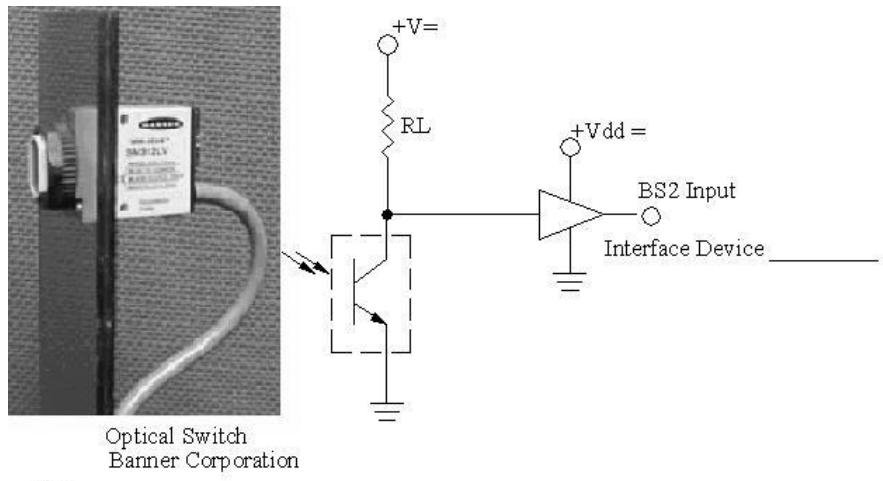


Circuit B

2. The comparator circuit below is used to determine when to turn on and off a dusk-to-dawn security lamp. What would be the output status of the comparator during “light” conditions? Would it be better to program for detecting the voltage level or the edge triggering of this circuit? Why?



3. The retroreflective optical switch below must be interfaced to the Basic Stamp. Its data sheet specifies that it to operates on 10 volts as a “current sink”. Refer to Figure 2.10 and fill in the appropriate values for the +V, +Vdd, and Interface device.



## Experiment #2: Digital Input Signal Conditioning

---

### Program it!

1. Pretend that your retro-reflective tachometer is providing the input to an anti-lock braking system on an automobile. In conjunction with this input, use a pushbutton to model the brake pedal switch. An active high LED will represent the braking action. Write a program that will detect the pressing of the brake pedal that would slow the vehicle. Have your program turn on the LED as long as speed is above zero. When shaft speed drops to zero, turn off the LED. Use a potentiometer to set initial motor speed. Configure the two pushbutton switches as active-high inputs. Wire one LED as an active-high output.
2. Write programs to duplicate the operation of an OR, AND, and XOR gate.

OR Gate		
PB1	PB2	LED
0	0	0
1	0	1
0	1	1
1	1	1

AND Gate		
PB1	PB2	LED
0	0	0
1	0	0
0	1	0
1	1	1

XOR Gate		
PB1	PB2	LED
0	0	0
1	0	1
0	1	1
1	1	0

Field Activity

How many digital (bi-state) field devices can you identify in a new car? List as many as you can. Make a note as to whether you suspect that the field device directly controls load current, drives some sort of relay, or if you think its status is being monitored by a microcontroller.







### Experiment #3: Digital Output Signal Conditioning

3

The outputs of a microcontroller can be used to control the status of output field devices. Output devices are those devices that do the work in a process-control application. They deliver the energy to the process under control. A few common examples include motors, heaters, solenoids, valves, and lamps. The low- power output capability of the BASIC Stamp (or any microcontroller) prevents it from providing the power required by these loads. With proper signal conditioning, the BASIC Stamp can control power transistors, thyristors, and relays. These are the devices that can deliver the load current and voltage demands of the field devices. In some applications, you may use a BASIC Stamp output to communicate with another microcontroller or electronic circuit. There may be compatibility issues of different logic families, separate power supplies, or uncommon grounds that require special consideration. The focus of this experiment is to present some of the signal conditioning techniques used to interface your BASIC Stamp to output field devices.

Appropriate signal conditioning design begins with a brief look at the characteristics and limitations of the BASIC Stamp's outputs. The output of the BASIC Stamp is considered "standard TTL" level. As we discussed in Experiment #2, this means it can switch between logic high of approximately 5 volts or logic low of nearly 0 volts. According to the BASIC Stamp's datasheet, each output can sink 25 mA and source 20 mA of current. Relating this to the partial diagram in Figure 3.1, notice how the load can be connected. In Figure 3.1a, the load is wired from the output pin to ground. When you set an output pin high, five volts appear across the load resistor ( $R_L$ ). Load current will flow from ground through the resistor and into the output pin. This is the current source mode, and the BASIC Stamp can deliver a maximum of 20 mA to the load.

Figure 3.1: BASIC Stamp Output Pin Current Capability

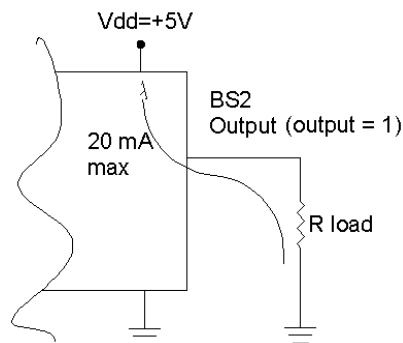


Figure 3.1a: Current Source

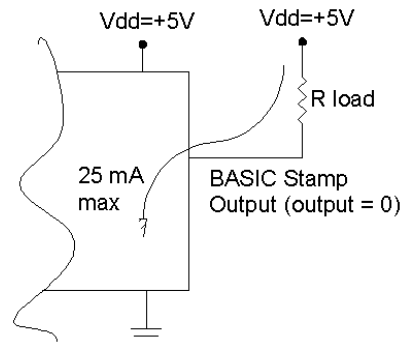


Figure 3.1b: Current Sink

## Experiment #3: Digital Output Signal Conditioning

In Figure 3.1b, the load is between the output pin and the +5-volt Vdd supply. Electrons will flow through the load now when the BASIC Stamp output pin is set Low (ground). Current will flow out of the output pin and up through the load resistor to Vdd. This is the current sinking mode, and the BASIC Stamp can deliver a maximum of 25 mA to the load when configured in this manner.

### What's a...

#### Output Capability of Digital Circuits

The output capability of digital circuits is listed in the manufacturer's datasheet. Devices usually can "sink" more current than they can "source." Some devices do not have the capability to source current because the internal path from their output to +V is not present. You may see this output design referred to as a device with an "open collector" output.

Outputs have been used to drive LEDs in previous exercises as pictured in Figure 3.2a. When the BASIC Stamp output is low, the diode is forward-biased at approximately one volt, and the remaining four volts, dropped across the 220-ohm resistor, limit current flow to approximately 22 mA. The light emitted by the diode gives visual indication of the output action. In previous programming challenges, you have assumed that the on-off status of an LED represents process action taking place. This is a valid assumption when you consider the operation of a solid-state relay (SSR). Figure 3.2b is a schematic representation of the solid-state relay. The input circuit (terminals 1-2) is equivalent to Figure 3.2a. The +3 to +24 VDC input identifies a range of control voltages. Control voltages must be above the minimum voltage to produce enough LED current for turn-on. Exceeding the maximum control voltage may cause damaging amounts of current to flow in the input LED. The light generated in the SSR strikes an optically controlled output circuit. The detail of this circuit is not shown, but is represented by the normally-open contact symbol. The current and voltage limitations of the output are listed in the device's datasheet and are usually printed on the device itself.

Figure 3.2: BASIC Stamp to LED and Solid-State Relay Schematics

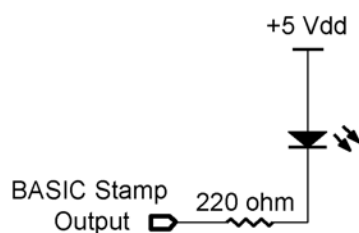


Figure 3.2a: BASIC Stamp to LED

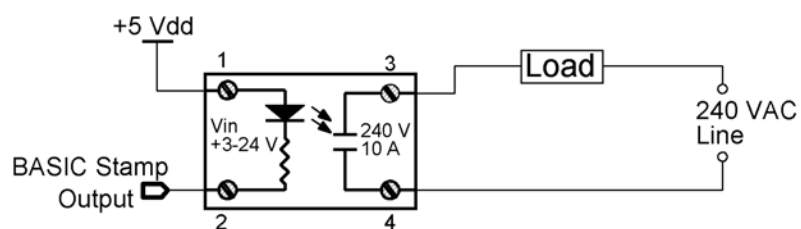


Figure 3.2b: BASIC Stamp to Solid-State Relay

Solid-state relays are available in a wide variety of output ranges. They may be designed to drive either AC loads or DC loads. The load you are driving defines the minimum specification of the SSR required.

An added benefit of the solid-state relay is electrical isolation. The BASIC Stamp is controlling the load by an optically-coupled signal. There is no electrical connection between the microcontroller and the high-power load device. Electrical failures of the load, or power line problems such as spikes, are not fed back to the BASIC Stamp. The datasheet for the Potter-Brumfield SSR may be found in Appendix C. Refer to this datasheet and find the following information.

Input voltage range: \_\_\_\_\_  
Input current requirement @ five volts: \_\_\_\_\_  
Maximum output load current: \_\_\_\_\_  
Maximum output load voltage: \_\_\_\_\_  
Electrical isolation: \_\_\_\_\_

A word of caution when selecting and implementing solid-state relays:

1. Do not push the specifications to their limits. Oversize the output capability of your selection by at least 20%.
2. Pay close attention to any heatsink requirements. Maximum load current capability is usually dependent upon incorporating a proper heatsink.
3. The load's supply source and all wiring and connections must be able to conduct the load's current. If relays are placed on the breadboard for prototyping, be aware that the breadboard traces are rated at only 1 amp.
4. Respect the output circuit voltage. Be sure all connections are solidly secured and correct before applying line voltage. There is the risk of electrical shock. Take measures to prevent contact with high-voltage potentials. Shield or encase these contacts. Clearly identify high-voltage potentials with appropriate labeling.
5. Some electronic relays will not contain an internal current-limiting resistor on the input. In these cases, an external current-limiting resistor must be added in series with the internal LED. The value of the resistor is based on your control voltage and the manufacturer's recommended input current specification.

Solid-state relays provide an easy interface for controlling loads in an industrial application. Become familiar with the SSR datasheet specifications in order to make the right selection for your application.



## Exercises

### Exercise #1: Sequential Control

The BASIC Stamp is well suited to perform sequential control operations. Many processes depend on the orderly performance of operations. Consider the machining operation pictured in Figure 3.3a.

Figure 3.3a: AutoDrill Sequence Operation

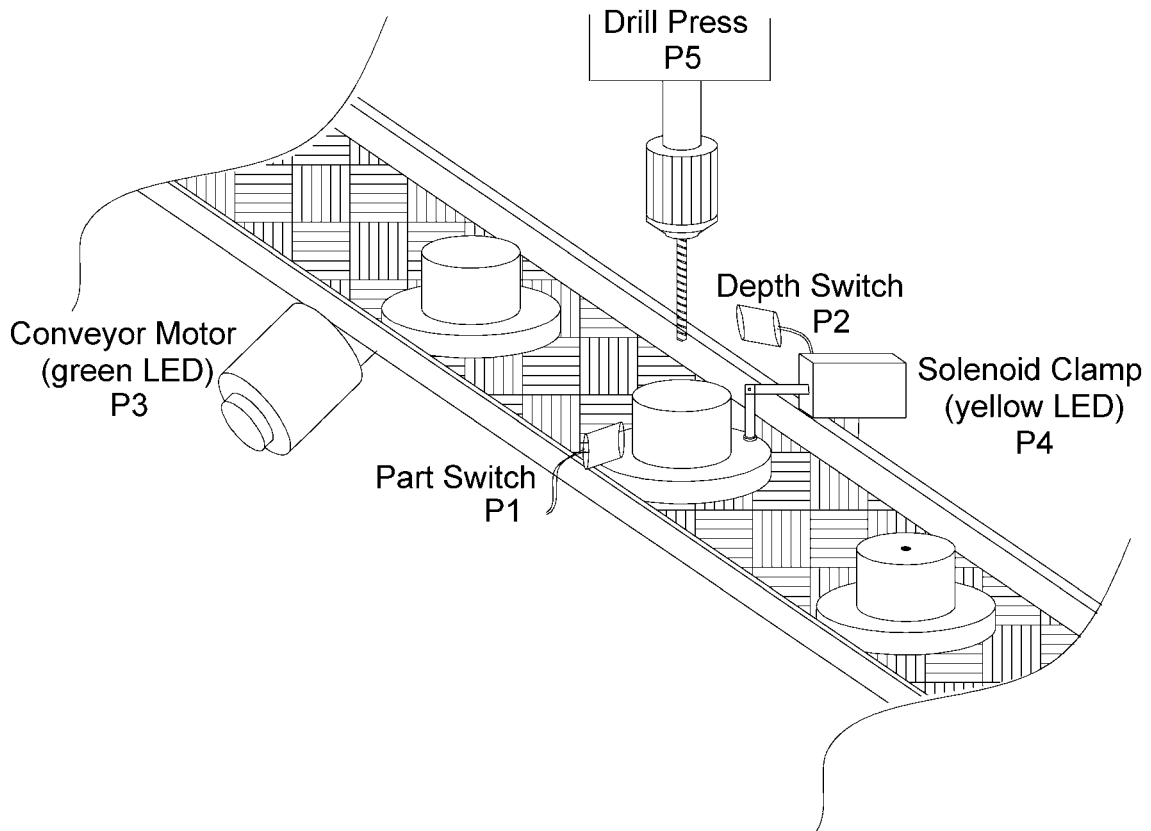
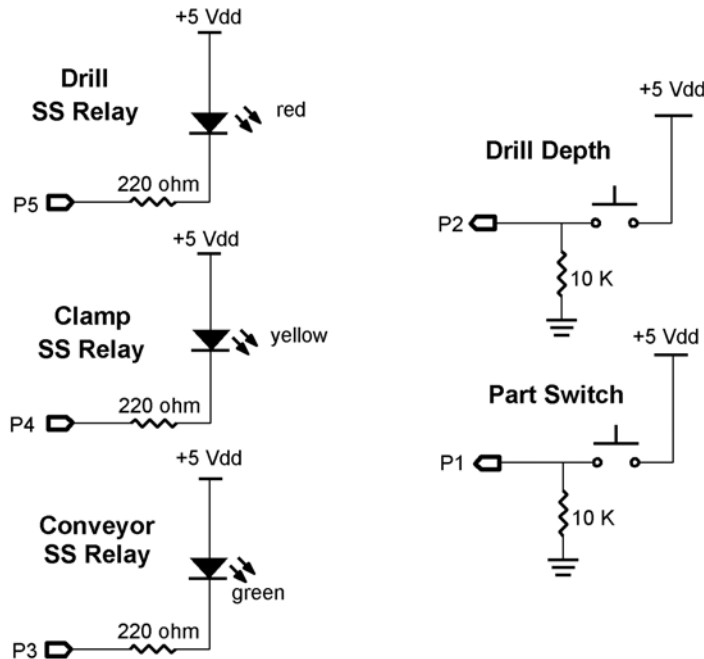


Figure 3.3b: Sequential Control  
(outputs are on the left; inputs shown to right)



A conveyor is moving parts through a machining station. When a part is detected in the staging area, the conveyor is turned off. After a short pause, the solenoid clamp is activated to hold the part; another short pause, and then the drill is brought down to the part. A proximity switch detects proper depth of the hole. When the depth switch closes, the “drill down” command is stopped and the drill is retracted. After allowing a short time for the drill to retract, the clamp is released and the conveyor is started. This moves the processed part out of the staging area, and the conveyor continues until a new part is detected. Upon detecting another part, the sequential process continues.

With proper signal conditioning, the BASIC Stamp can easily control this sequence. For our exercise purposes, you are asked to use your imagination to allow LEDs to simulate the SSRs that could control the conveyor, clamp, and drill. Two pushbuttons and your two fingers must simulate the part coming into position and the drill coming down to proper depth. Construct Figure 3.3b on your Board of Education. For easier

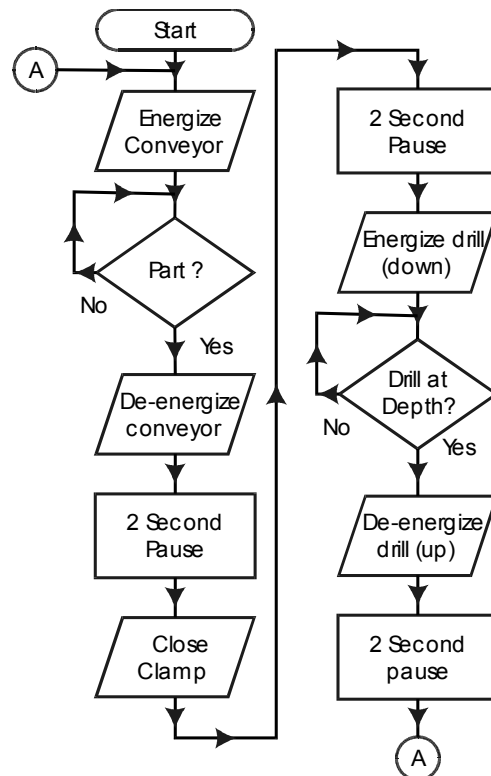
### Experiment #3: Digital Output Signal Conditioning

---

identification, use your green LED for the conveyor, your yellow LED for the clamp, and the red LED for the drill.

Sequential control lends itself well to flowcharting. The time required to develop your flowchart will be quickly saved as you write your program. Compare the flowchart in Figure 3.4 to the description of the machining process.

Figure 3.4: Sequential Control Flowchart



Program 3.1 follows the flowchart very closely. Study the program; compare its structure to the flowchart. Enter the program and try it. Again, you will have to use your imagination to simulate our process.

When the program starts, the green LED will be on. This represents the conveyor starting. To simulate a part coming into the staging area, you must press and hold Pushbutton P1. The conveyor LED will instantly turn OFF and the yellow LED indicating the Clamp relay will turn on after a one-second delay. After the Clamp has had

two seconds to secure the part, the drill will come down toward the part as indicated by the red LED. At this time bring another finger down to simulate the drill. Pushbutton P2 represents a proximity switch, which will indicate when proper drill depth has been reached. Your “drill” finger pressing P2 will be turned OFF; the red LED indicating the drill is retracting. Your finger now coming off of the P2 pushbutton indicates the bit has started retracting and two seconds will be allowed for the drill to clear the part. After this delay, the clamp will be opened (yellow light OFF) and the conveyor will start again. The part is completed and leaves the staging area. From this point the sequence starts again.

Run the program a few times. Other than the DEBUG report that a part has been completed, there is no need for your computer. Unplug the serial cable from the Board of Education and continue to simulate the sequential process. The BASIC Stamp could function as the “embedded controller” in this application. Wiring the actual field devices to the BASIC Stamp would allow it to continuously repeat the process.

After understanding this sequential process, we will redefine your two inputs and three outputs to simulate another operation. You will be challenged to develop the program necessary for this embedded control application.

```
'Program 3.1: Sequential Process Control Machining Operation - Embedded

INPUT 1           ' Part detection switch
INPUT 2           ' Drill depth switch
OUTPUT 3          ' Conveyor motor relay (green)
OUTPUT 4          ' Clamp solenoid relay (yellow)
OUTPUT 5          ' Drill press relay (red)

Off CON 1         ' Current sink loads
On CON 0          ' Negative logic

OUT3 = Off        ' Initialize outputs off
OUT4 = Off
OUT5 = Off
Start:
  OUT3 = On        ' Conveyor on
  IF IN1 = 1 THEN Process ' If pressed, start "Process"
GOTO Start

Process:          ' The process begins
  OUT3 = Off      ' Stop conveyor
  PAUSE 1000
  OUT4 = On       ' Begin clamping part in place
  PAUSE 2000     ' Wait 2 seconds to turn drill on

Drill_down:
  OUT5 = On       ' Turns on drill and drill begins dropping
  IF IN2 = 1 THEN Pull_drill ' If drill is deep enough, pull drill
GOTO Drill_down
```

## Experiment #3: Digital Output Signal Conditioning

---

```
Pull_drill:
  OUT5 = OFF           ' Turns off drill and drill retracts
  IF IN2 = 0 THEN Drill up ' Indicates drill is moving up
GOTO Pull_drill

Drill up:
  PAUSE 2000          ' Continue pulling drill up for 2 seconds

Release:
  OUT4 = Off          ' Open clamp to release part
  PAUSE 1000          ' Wait 1 second
  OUT3 = On           ' Conveyor on
  IF IN1 = 0 THEN Next_part ' Finished part leaves process area
GOTO Release

Next part:
  PAUSE 1000          ' Wait 1 second
  DEBUG "Part leaving clamp. Starting next cycle", CR
GOTO Start
```

The real beauty of microcontrollers is to have the capability of embedding all of the intelligence necessary to perform sophisticated control within the equipment.

There are times, however, that being able to retrieve information from the microcontroller adds to its capabilities. The StampPlot Lite interface can be effectively used to monitor the sequential machining process. Program 3.2 uses this interface. The machine functions in Program 3.2 are the same as those in the previous program. **DEBUG** commands have been embedded to send data to the StampPlot Lite interface. The program plots the status of the digital I/O, reports process steps in the User status bar, and keeps a time-stamped list of the total parts produced. Figure 3.5 is a representative screen shot of the sequential process being monitored by StampPlot Lite. Load Program 3.2 and run it. Study the StampPlot Lite **DEBUG** commands that have been added to the original program. Become familiar with their use. Graphical user interfaces such as this are very useful in the maintenance and data acquisition of embedded control systems. Use StampPlot to monitor the Sequential Control Mixing Challenge at the end of this section.

Program 3.2: Sequential Process Control Machining Operation with StampPlot Interface

```
Pause 500
DEBUG "!TITL Sequential Process Control Machining Operation", CR
' StampPlot title
DEBUG "!TMAX 100", CR           ' Set sweep plot time (seconds)
DEBUG "!PNTS 500", CR          ' Sets the number of data points
DEBUG "!AMAX 20", CR           ' Sets vertical axis (counts)
DEBUG "!CLRM", CR              ' Clear List Box
DEBUG "!CLMM", CR              ' Clear Min/Max
DEBUG "!RSET", CR              ' Reset all plots
DEBUG "!DELD", CR              ' Delete old data file
DEBUG "!PLOT ON", CR           ' Turn Plot on
DEBUG "!TSMP ON", CR           ' Time-stamp part completion
```



```

DEBUG "!SAVD ON", CR      ' Save data to file

INPUT 1                  'Part Detection Switch
INPUT 2                  'Drill Depth Switch
OUTPUT 3                 'Conveyor motor relay (green)
OUTPUT 4                 'Clamp solenoid relay (yellow)
OUTPUT 5                 'Drill press relay (red)

Off CON 1                'Current sink mode
ON CON 0                 'Negative logic

OUT3 = Off               ' Initialize outputs off
OUT4 = Off
OUT5 = OFF

Parts VAR byte
Parts = 0

Start:
  GOSUB Plot data        ' Plot the status
  OUT3 = On              ' Conveyor on
  DEBUG "!USRS Start conveyor",CR ' User status prompt
  IF IN1 = 1 THEN Process ' If pressed, start "Process"
  PAUSE 100
GOTO START

Process:
  GOSUB Plot data        ' Plot the status
  OUT3 = Off             ' Stop conveyor
  DEBUG "!USRS Detected part. Stop conveyor",CR ' User status prompt
  PAUSE 1000

  GOSUB Plot data        ' Plot the status
  OUT4 = On              ' Begin clamping part in place
  DEBUG "!USRS Clamp part.",CR ' User status prompt
  GOSUB Plot_data        ' Plot the status
  PAUSE 2000            ' Wait 2 seconds to turn drill on

Drill down:
  GOSUB Plot data        ' Plot the status
  OUT5 = ON              ' Turns on drill and drill drops
  DEBUG "!USRS Drill coming down!",CR ' User status prompt
  IF IN2 = 1 Then Pull drill ' If drill is deep enough, pull drill
  PAUSE 100
GOTO Drill down

Pull_drill:
  GOSUB Plot_data        ' Plot the status
  OUT5 = OFF             ' Turns off drill and drill retracts
  DEBUG "!USRS Stop Drill and Retract",CR ' User status prompt
  IF IN2 = 0 Then Drill_up ' Indicates drill is moving up

```

## Experiment #3: Digital Output Signal Conditioning

---

```
    PAUSE 100
GOTO Pull_drill

Drill_up:
  GOSUB Plot_data          ' Plot the status
  DEBUG "!USRS Drill coming up!!",CR ' User status prompt
  PAUSE 2000              ' Pull drill for 2 seconds

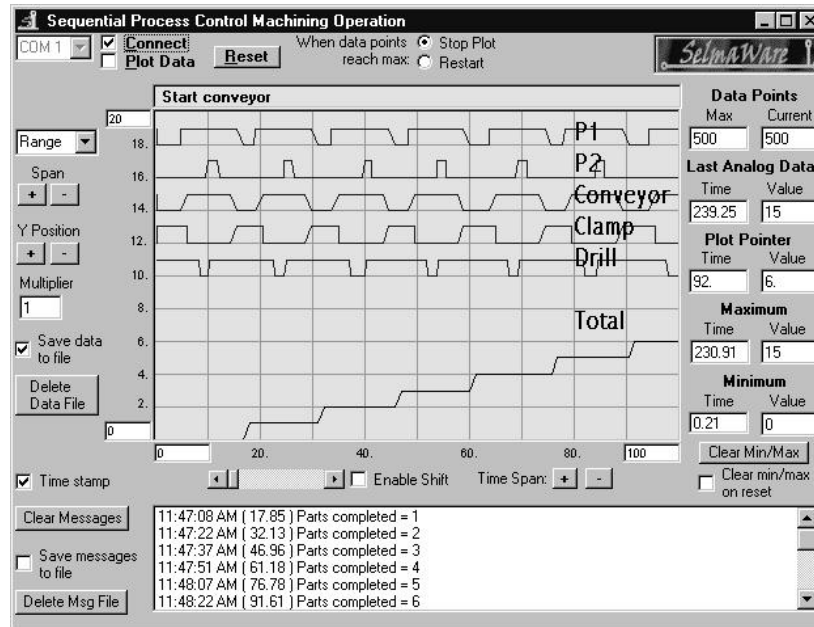
Release:
  GOSUB Plot_data          ' Plot the status
  OUT4 = Off              ' Open clamp to release part
  DEBUG "!USRS Clamp released. Conveyor moving.",CR
                          'User status prompt
  PAUSE 1000              ' Wait 1 seconds
  OUT3 = On               ' Conveyor on
  IF IN1 = 0 Then Next_part
  GOTO Release

Next_part:
  GOSUB Plot_data          ' Plot the status
  DEBUG "!USRS Part Complete. Start next cycle",CR
                          ' User status prompt
  Parts = Parts + 1       ' Parts counter
  PAUSE 1000              ' Wait 1 seconds
  DEBUG "Parts completed = ", DEC Parts,CR
                          ' Post parts count in the List Box

GOTO Start

Plot_data:
  DEBUG IBIN IN1,BIN IN2,BIN OUT3,BIN OUT4, BIN OUT5,CR
                          'Plot the digital status.
  DEBUG DEC Parts,CR      'Plot analog count
RETURN
```

Figure 3.5: Screen Shot of the Sequential Machining Process using StampPlot Lite



3

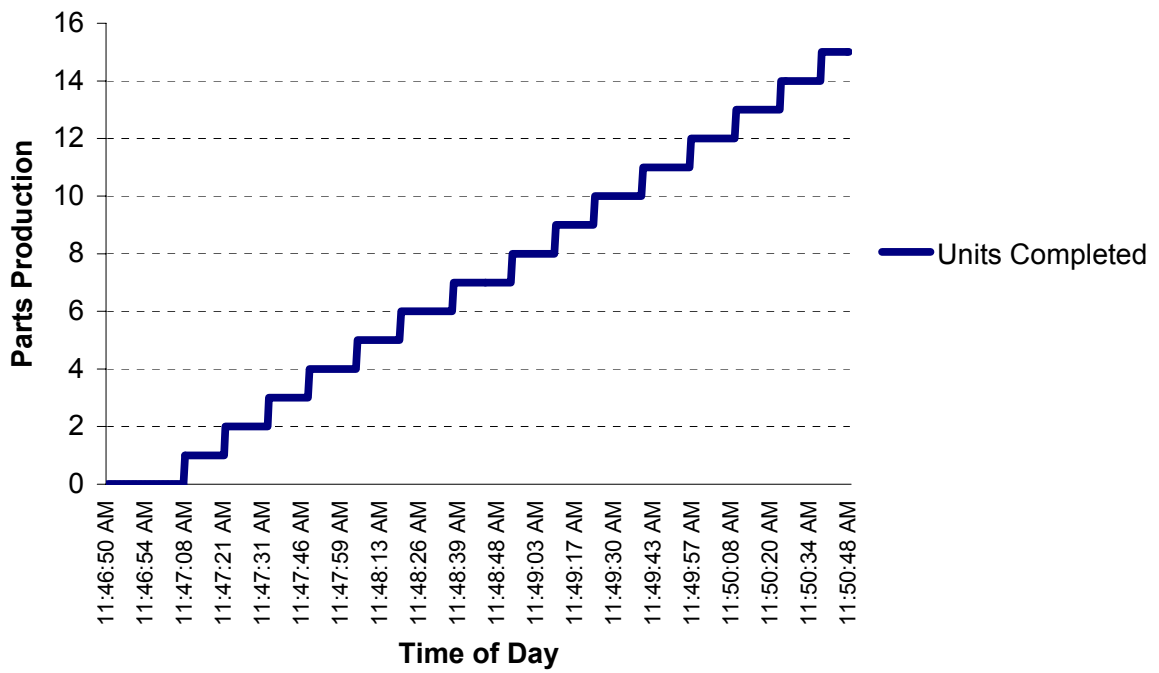
Note that the traces appear from top to bottom in the order which they were listed in the Debug digital plot command. Therefore, the top two traces are of the active high pushbuttons **IN1** (product in position) and **IN2** (depth switch). The next three traces are outputs **OUT3** (conveyor), **OUT4** (clamp), and **OUT5** (drill). Remember that the outputs are wired in the current sink mode. A High is OFF and a Low is ON.

Notice that in the initial setting for the StampPlot Lite interface, "Save data to file" (!SAVD) is ON. During the production run, the data at each sample point is saved into a text file, stampdat.txt. The data includes the time of day and program time that the sample was taken, the sample number, and the analog and digital values at the time of each sample. The data are comma delimited (separated by commas), and therefore, ready to be brought into a variety of spreadsheet or database software packages. Once the data is in the package, it is available for analysis and manipulation. Figure 3.6 represents a portion of the production run data, as it would appear in a Microsoft Excel spreadsheet. The complete file contains 500 samples (rows of data). Figure 3.7 is an Excel graph constructed from the data file.

Figure 3.6: Sequential Control Production Run (samples only)

Time of Day	Run Time	Sample number	Units Completed	Sample number	Digital Status
11:46:50 AM	0.21	1	1	1	111
11:46:50 AM	0.21	2	2	2	11
11:46:50 AM	0.21	3	3	3	11
11:46:50 AM	0.21	4	4	4	11
11:46:50 AM	0.27	5	5	5	11
11:46:50 AM	0.27	6	6	6	11
11:46:50 AM	0.27	7	7	7	11
11:46:50 AM	0.27	8	8	8	11
11:46:50 AM	0.27	9	9	9	11
11:46:50 AM	0.27	10	10	10	11
11:46:50 AM	0.32	11	11	11	11
11:46:50 AM	0.32	12	12	12	11
11:46:51 AM	0.43	13	13	13	11
11:46:51 AM	0.50	14	14	14	11
11:46:51 AM	0.71	15	15	15	11
11:46:51 AM	0.98	16	16	16	11
11:46:51 AM	1.26	17	17	17	11

Figure 3.7: Graph of Sequential Control Production Run

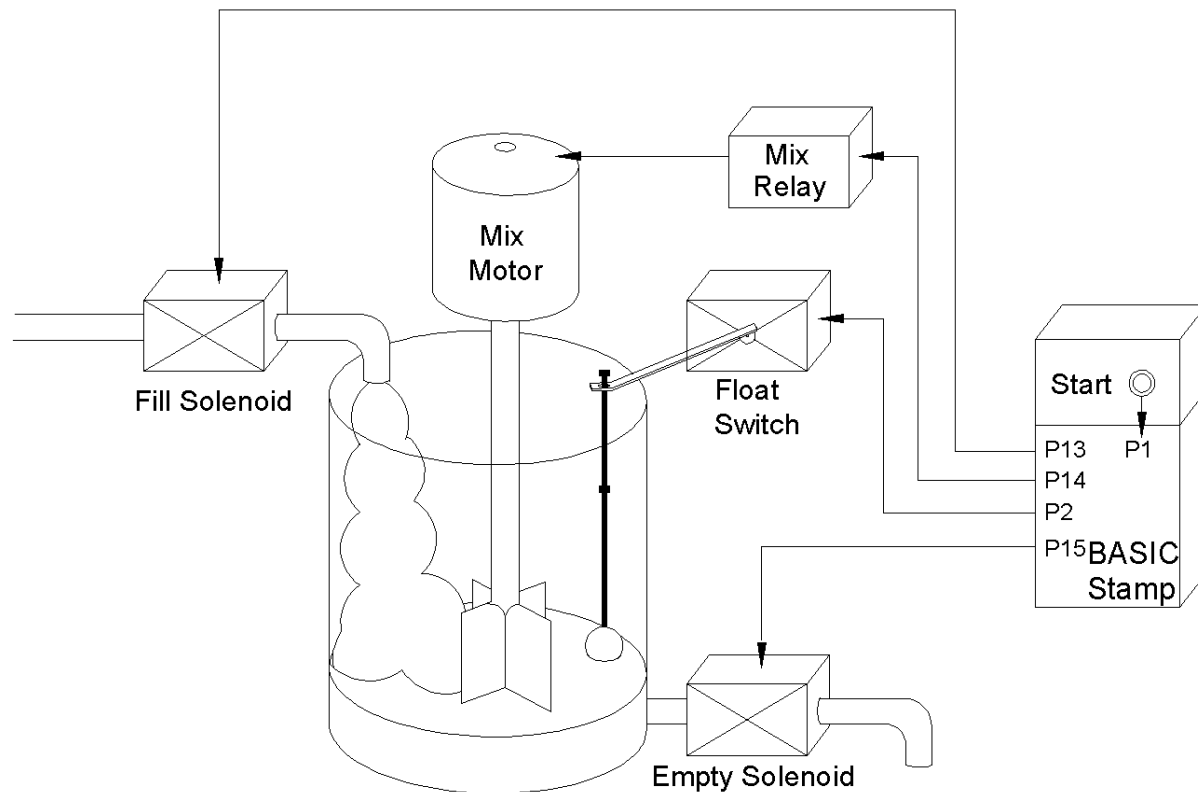


3

Programming Challenge: Sequential Mixing Operation

A mixing sequence is pictured in Figure 3.8. In this process, an operator momentarily presses a switch to open a valve and begin filling a vat. A mechanical float rises with the liquid level and closes a switch when the vat is full. At this time, the “fill” solenoid is turned off, and a mixer blends the vat contents for 15 seconds. After the mixing period, a solenoid at the bottom of the vat is opened to empty the tank. The mechanical float lowers, opening its switch when the vat is empty. At this point, the “empty” solenoid is turned off and the valve closes. The process is ready for the operator to start another batch.

Figure 3.8: Mixing Sequential Control Process



Assign the following to the BASIC Stamp inputs and outputs to simulate the operation.

Operator pushbutton	Input P1 (N.O. active high)
Float switch	Input P2 (N.O. active high)
Fill Solenoid	Output P13 (red LED)
Mix Solenoid	Output P14 (yellow LED)
Empty Solenoid	Output P15 (green LED)

Construct a flowchart and program the operation.

### Exercise #2: Current Boosting the BASIC Stamp

The BASIC Stamp's output current and/or voltage capability can be increased with the addition of an output transistor. Either the bipolar transistor shown in Figure 3.9a or the power MOSFET transistor in Figure 3.9b can be effective when loads need more power than the BASIC Stamp's output can deliver. Understanding each of these circuits will be important in future industrial applications.

For this exercise, and upcoming experiments, we have two loads that we wish to drive in this manner. They are a brushless DC fan and a 47-ohm, half-watt resistor. The brushless fan specifications include a full line voltage of +12 V and line current of 100 mA. The resistor will draw approximately 190 mA when powered by the +9 V Vin power supply.

Let's consider the design of the bipolar transistor for driving the 47-ohm resistor. The circuit values should be designed such that a high (+5V) output of the BASIC Stamp drives Q1 into saturation without drawing more current than the BASIC Stamp can source.

Figure 3.9: Current Boost Transistor Driver Circuits

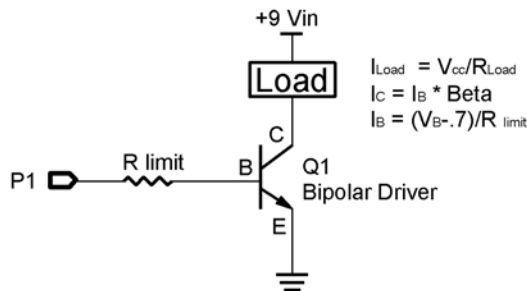


Figure 3.9a: BASIC Stamp to Bipolar Transistor Driver

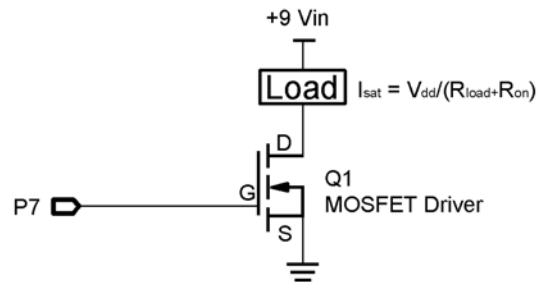


Figure 3.9b: BASIC Stamp to Power MOSFET

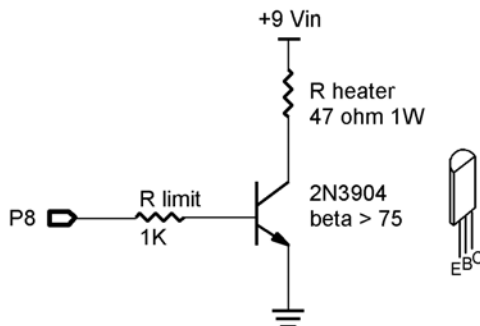


Figure 3.9c: BASIC Stamp Driving the Heater

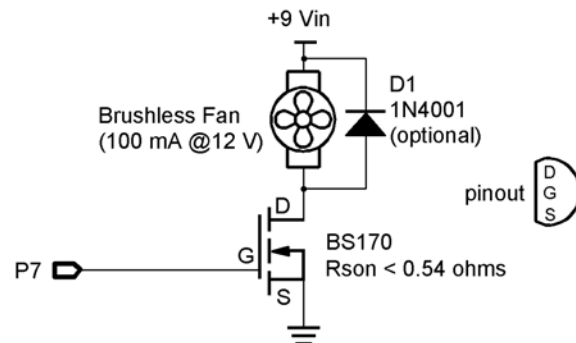


Figure 3.9d: BASIC Stamp Driving the Fan

Circuit component values stem from the load current and voltage requirements. The process of determining minimum component values is as follows: Since Q1 acts as an open collector current sink to the load, the load's supply voltage is not limited to the BASIC Stamp's +5-volt supply. If separate supplies are used, however, their common ground lines must be connected. When Q1 is driven into saturation, virtually all of the supply voltage will be dropped across the load and the load current will be equal to  $V_{supply}/R_{load}$ . Q1's maximum collector current capability must be higher than this load current. The Q1 base current required to yield the collector current may be calculated by dividing the load current by the "beta" of Q1.  $I_B = I_C/\beta_{Q1}$ . Given a 20 mA maximum BASIC Stamp output current, a minimum transistor beta may be calculated by rearranging this formula.  $\beta_{Q1(min)} = I_C/I_B$ . Where  $I_B$  is the 20 mA maximum BASIC Stamp drive current.

A transistor must be chosen that meets, and preferably exceeds, these minimum requirements. Exceeding the minimum values by 50 to 100% or more would be best. Once the transistor is chosen, an appropriate base-



limiting resistor value can be determined. This value must allow more base current than that defined by  $I_C/b_{Q1}$ , yet less than the 20 mA BASIC Stamp limit. The voltage drop across  $R_{limit}$  is equal to the +5 V BASIC Stamp output minus the PN junction drop of Q1 (approximately +5V- .7V, or 4.3V).

Following the procedure outlined above, the transistor must handle collector currents of at least 190 mA and have a beta specification greater than 10. Figure 3.9c represents a 2N3904 as Q1. Specifications for the 2N3904 include a collector current capability of 300 mA and a minimum Beta of 75. Added features to the selection of this transistor include that it is very common, it is inexpensive, and it can deliver the load current without need of a heat sink.  $R_{limit}$  values were selected based on minimum beta specifications and a desire to keep the BASIC Stamp output current demand well below 20 mA. This 1K-ohm resistor allows approximately 5 mA of base current, which ensures saturation.

Construct the transistor-driver circuits in Figure 3.9c on your Board of Education.

Next, consider the power MOSFET drive circuit in Figure 3.9b. The MOSFET is driven into saturation by applying gate voltage. A positive five volts from the BASIC Stamp's output is sufficient to place the MOSFET in an "ON" state. When the device is fully saturated, its ON-state resistance ( $r_{s_{on}}$ ) is typically less than 1 ohm. Applying a low (0V) to the gate places the device in cutoff. In this state there is virtually no load current and the MOSFET acts as an open switch.

The power MOSFET is very easy to drive with the BASIC Stamp. A metal oxide (MOS) layer between the source and the gate acts as a very good insulator. The extremely high input impedance provided by this MOS layer means that no gate current is required to control this device. Since no current is required to drive the gate, a single output from the BASIC Stamp can control multiple MOSFETs.

With proper heatsinking, the BS170 can handle load currents up to 5 amps. These features make the power MOSFET very easy to apply in industrial applications such as driving relays, solenoids and small DC motors. It should be noted that these types of loads are inductive. When switching off the load, this inductance can produce a reverse voltage transient that may be damaging to the MOSFET. The diode D1 provides protection for the transistor when driving inductive loads such as these. This diode is not necessary for the small brushless motor used in our experiments. Construct the circuit in Figure 3.9d.

Note: Power MOSFETs, like their CMOS cousins, are susceptible to damage from static discharge and reverse voltage transients. Care should be taken when handling and installing the device. Hold the device by its body, avoid touching its leads, and be sure that the work surface and soldering equipment is properly grounded.

## Experiment #3: Digital Output Signal Conditioning

---

Test the current-boost drive circuits using the following code. The Debug window will prompt you to make voltage measurements across the transistors and their loads at the proper times.

```
'Program 3.3: Current-boost for the fan and heater.

Output 7           ' Output for Fan drive
Output 8           ' Output for Heater drive

Loop:

OUT7 = 0
OUT8 = 0
DEBUG "Fan and Heater OFF - Measure the unloaded Vin voltage source.",CR
PAUSE 10000
OUT7 = 1
DEBUG "Fan ON - Measure the voltage across the fan.", CR
PAUSE 10000
DEBUG "Fan ON - Measure the saturation voltage across the MOSFET.", CR
PAUSE 10000
OUT7 = 0
DEBUG "Fan OFF",CR
PAUSE 2000

OUT8 = 1
DEBUG "Heater ON - Measure the voltage across the 47-ohm resistor", CR
PAUSE 10000
DEBUG "Heater ON - Measure the saturation voltage across the Bipolar.", CR
PAUSE 10000
OUT8 = 0
DEBUG "Heater Off",CR
PAUSE 2000

GOTO Loop
```

Record the voltage readings in Table 3.1.

Table 3.1 Transistor Current Boost			
Condition	On-state load voltage	On-state saturation voltage	Off-state cut off voltage
Bipolar Fig. 3.9c			
MOSFET Fig. 3.9d			

The fan will be running at full speed, and the resistor will be warming up due to the current flowing through it. Saturation voltages should be less than 300 mV. The line voltage  $V_{in}$  is provided by the unregulated 9-volt 300 mA supply. Off-state voltage may measure as high as 14 volts. When the loads are energized, this voltage will drop to around 9 volts.

In the next experiment, we will use the resistor to simulate a heating element. The fan will simulate a process disturbance that cools the heater. Our objective will be to investigate various types of control to maintain a constant temperature. Leave these circuits constructed on your Board of Education.

Before we leave this exercise, it is worth mentioning some other interfacing challenges that you may be confronted with as a designer. Consider the circuits in Figure 3.10.

**3**

Experiment #3: Digital Output Signal Conditioning

Figure 3.10: BASIC Stamp Output Interfacing

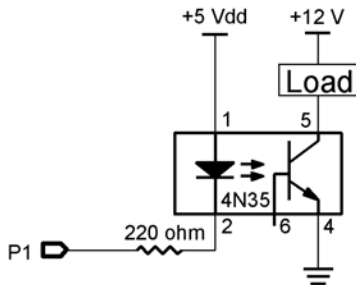


Figure 3.10a: BASIC Stamp to Optocoupler

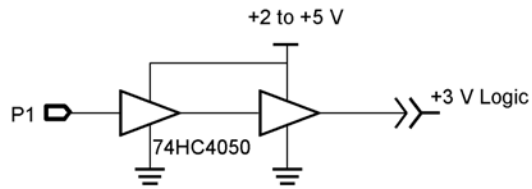


Figure 3.10b: BASIC Stamp to HCMOS

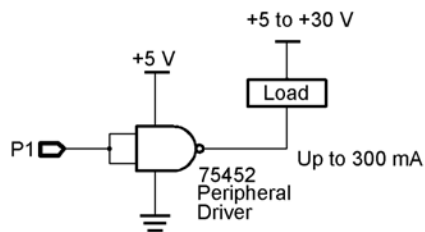


Figure 3.10c: BASIC Stamp to High Current Driver

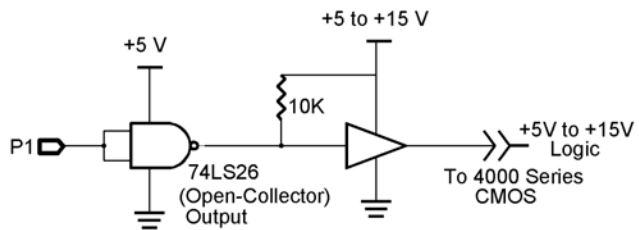


Figure 3.10d: BASIC Stamp to +15 V Logic

- The opto-coupler can be used to interface different voltages and to electrically isolate an output from the microcontroller circuit in Figure 3.10a.
- Figure 3.10b can be used to interface to HCMOS or 4000-series CMOS devices. The 74HC4050 can be operated on low voltages, allowing interfacing to +3-volt logic.
- There is a large variety of peripheral driver chips available. The 75452 driver depicted in Figure 3.10c can sink up to 300 mA of load current. Its open-collector output allows for loads up to 30 volts.
- Figure 3.10d includes the 74LS26 NAND gate. This is one of a family of open-collector gates. With the 10K-ohm pull-up resistor referenced to the next circuit stage, the BASIC Stamp can be interfaced to higher-voltage CMOS circuits.



## Questions and Challenges

3

### Questions

1. Output field devices are those devices that do the \_\_\_\_\_ in a process control application.
2. Field devices usually require more power than the BASIC Stamp can deliver. List three power interface devices that can control high-power circuits and be turned on and off by the BASIC Stamp.
  - a. \_\_\_\_\_
  - b. \_\_\_\_\_
  - c. \_\_\_\_\_
3. The BASIC Stamp output is acting as a current sink when the load it is driving is connected between the output pin and \_\_\_\_\_.
4. The BASIC Stamp can source \_\_\_\_\_ mA per output.
5. Electronic and electromagnetic relays offer a level of protection to the microcontroller because they provide electrical \_\_\_\_\_ between the BASIC Stamp and the power devices.
6. The input circuit of an SSR is usually an \_\_\_\_\_, which provides light that optically triggers an output device.
7. The current rating of an SSR should be oversized by at least \_\_\_\_\_ percent of the continuous load current demand.
8. Maximum continuous current ratings of solid-state relays usually involve applying a \_\_\_\_\_ for proper heat dissipation.
9. \_\_\_\_\_ control involves the orderly performance of process operations.
10. When the output current from the BASIC Stamp is not sufficient to turn on the control device, an output \_\_\_\_\_ may be used for current boosting.

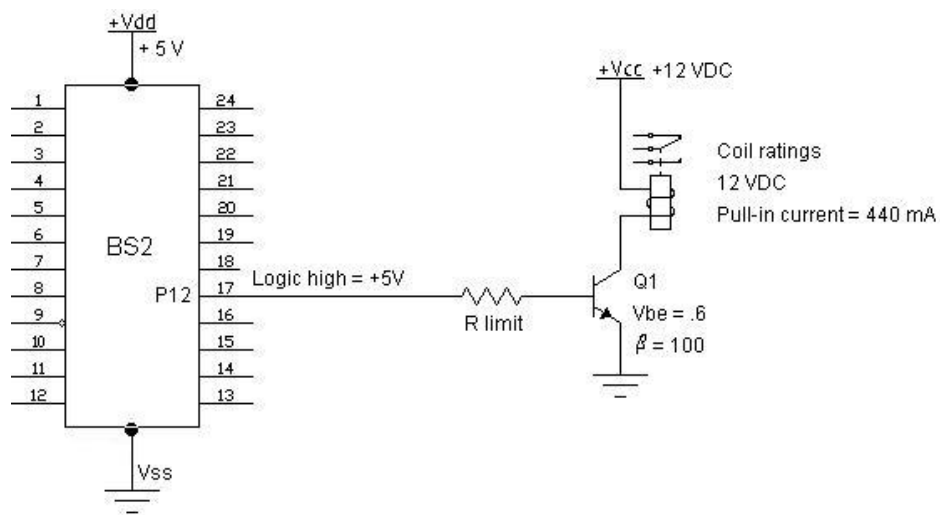
### Experiment #3: Digital Output Signal Conditioning

---

11. If a transistor has a Beta of 150, a BS2 must deliver \_\_\_\_\_ milliamps of base current to drive a 600 mA load.
12. When a power MOSFET is saturated, its Drain to Source resistance is given as a specification termed \_\_\_\_\_.
13. The contacts of an electromagnetic relay are shown in schematics in the “normal” position. Normal means the relay’s coil is \_\_\_\_\_ energized.
14. The “contacts” of an AC solid-state relay are actually the main terminals of a TRIAC. These contacts would be depicted in a schematic as being normally \_\_\_\_\_.

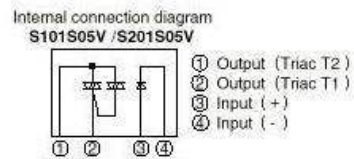
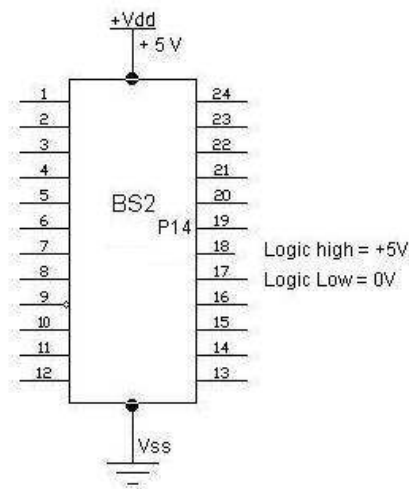
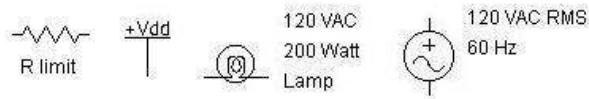
#### Design It!

1. Given the figure below, solve for the maximum value of the base limiting resistor ( $R_{limit}$ ) that would allow the 440 mA of coil current to flow when the BASIC Stamp output Pin 12 is high.



2. To ensure deep saturation of transistor Q1, the value of  $R_{limit}$  should be \_\_\_\_\_ than this value.

3. The internal connection diagram of the SHARP S101S05V solid-state relay is given below. Notice that its input circuit is just an LED. The datasheet specifies that the LED has a forward voltage drop of 1.2 volts and that 15 mA through the LED will turn on the relay. Use the following components to complete the diagram for controlling the SSR with Pin 14 of the BASIC Stamp. Configure it as a current sink. Calculate the proper value of  $R_{limit}$ . Draw the lamp and 120 VAC source as they would be connected to the SSR outputs.

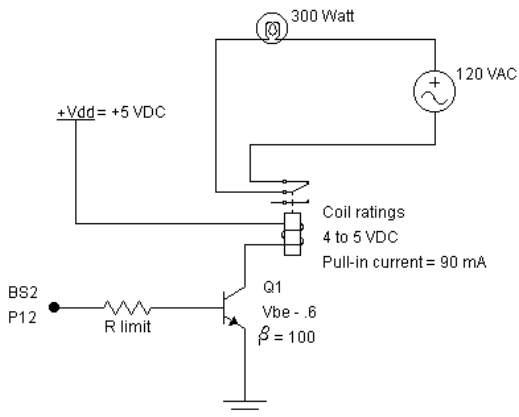


## Experiment #3: Digital Output Signal Conditioning

### Analyze it!

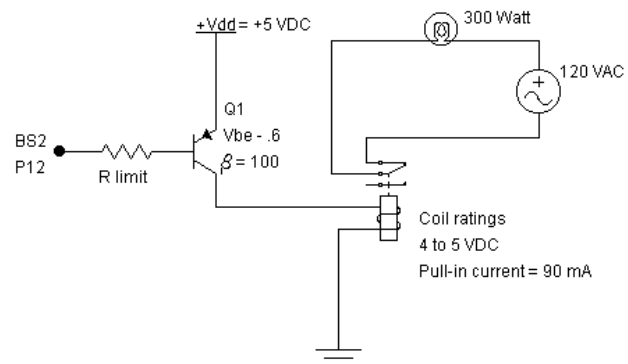
1. Consider circuits A and B below. Write a line of BASIC Stamp code that will result in turning the lamp ON for each.

Circuit A \_\_\_\_\_.



Circuit A: Current source drive and normally closed contacts.

Circuit B \_\_\_\_\_.



Circuit B: Current sink drive and normally open contacts.



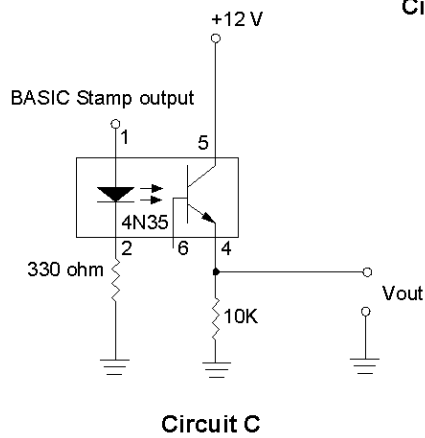
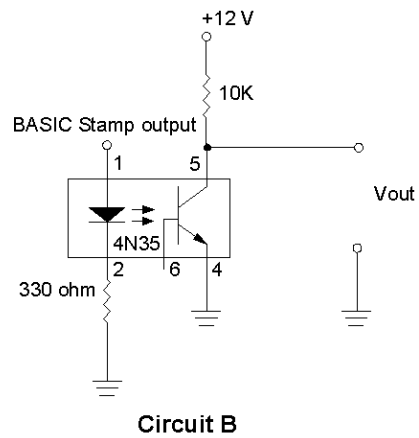
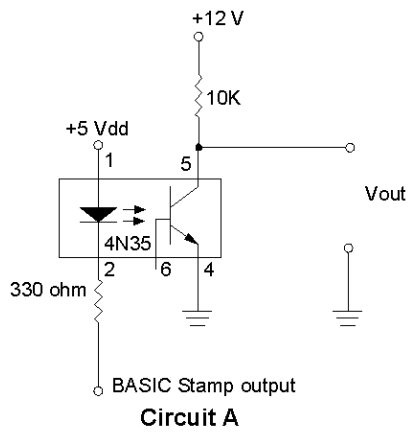
2. Study the three figures shown below. Would you write a logic High or a logic Low to the BASIC Stamp output to yield a 12-volt  $V_{out}$  value?

Circuit A \_\_\_\_\_

Circuit B \_\_\_\_\_

Circuit C \_\_\_\_\_

**3**



## Experiment #3: Digital Output Signal Conditioning

---

### Program it!

1. Given the input and outputs pictured back in Figure 3.3b, write a sequential program that will do the following:

Momentarily pressing P1 will cause P3 to turn ON for three seconds and then go OFF. Pressing P1 a second time will cause P4 to come ON for three seconds and then go OFF. When P4 goes OFF, P5 will come ON until P2 is pressed.

2. Try this one. Using the same I/O, write a program that will do the following:

Press and hold P1 and P3 goes ON. Holding P1 and pressing P2 causes P3 to go OFF and P4 to come ON. Releasing P1 while continuing to hold P2 turns OFF P4 and ON P5. And lastly, releasing P2 will turn all three outputs ON for three seconds, then all OFF, and the process is set to repeat.



## Experiment #4: Continuous Process Control

Continuous process control involves maintaining desired process conditions. Heating or cooling objects to a certain temperature, holding a constant pressure in a steam pipe, or setting a flow rate of material into a vat in order maintain a constant liquid level, are examples of continuous process control. The condition we desire to control is termed the “process variable.” Temperature, pressure, flow rate, and liquid level are the process variables in these examples. Industrial output devices are the control elements. Motors, valves, heaters, pumps, and solenoids are examples of devices used to control the energy determining the outcome of the processes.

4

The control action taken is based on the dynamic relationship between the output device’s setting and its effect on the process. Generally speaking, process control can be classified into two types: open loop and closed-loop. Closed-loop control involves determining the output device’s setting based on measurement and evaluation during the process. In open-loop control, no automatic check is made to see whether corrective action is necessary.

A simple example of open-loop control would be cooling your bedroom on a hot summer evening. Your choices are using a window fan or an air conditioner. The window fan is a device that you set – low, medium, or high speed – based on your evaluation of what the situation needs for control. This evaluation involves an understanding of what the cause-and-effect relationship is of your speed setting vs. the room conditions. There is also an element of prediction involved. Once you make the setting decision, you are in for the night. You are setting up an open-loop control system. If your evaluations are correct, you will have a great night’s sleep. If they are not, you may wake up shivering and cold or sweaty and hot! On the other hand, a room air conditioner allows you to set a certain desired temperature. A thermostat continuously compares the desired temperature with a measurement of actual room temperature. When room temperature is over the desired setpoint, the air conditioner is turned on. As the room cools below the setpoint, the air conditioner is turned off. As the night goes on and the outside temperature cools down, this closed-loop system will automatically spend less time on than off. This is an example of closed-loop feedback control, because the action is taken based on measurement of room temperature.

Which is better? Arguably, some people prefer air conditioning to a fan, but others do not. If the objective is to maintain a comfortable sleeping temperature, they both have their advantages. In terms of industrial control, the lower cost and simplicity of setting the window fan in an open-loop mode is very attractive. On the other hand, the automatic control of the closed-loop air conditioner ensures a more consistent bedroom temperature as the outside temperature changes.

## Experiment #4: Continuous Process Control

---

Determining the best control action for an application and designing the system to provide this action is what the field of process control engineering is all about.

Microcontrollers have proven to be a dependable, cost-effective means of adding a level of sophistication to the simplest of control schemes. The next three exercises will focus on the characteristics of various methods of continuous control. We will develop an environment in which we can model process control, get process variable data into the BASIC Stamp, and study open-loop control principles. The first two exercises will take a little time and effort, but will be worthwhile, because the setup and circuitry will be used again for Experiments #5 and #6.

Temperature is by far the most common process variable that you will encounter. From controlling the temperature of molten metal in a foundry to controlling liquid nitrogen in a cryogenics lab, the measurement, evaluation, and control of temperature are critical to industry. The objective of this exercise is to show principles of microcontroller-based process control and enlighten you about interfacing the controller to real-world I/O devices. The exercises are restricted to circuits that fit on the Board of Education and to output devices that can be driven by its 9-volt, 300-mA power supply. As you monitor and control the temperature of a small environment, realize that, through proper signal conditioning, the applications for which you can apply the BASIC Stamp are limitless.



### Exercises

#### Exercise #1: Closed-Loop, On-Off Control

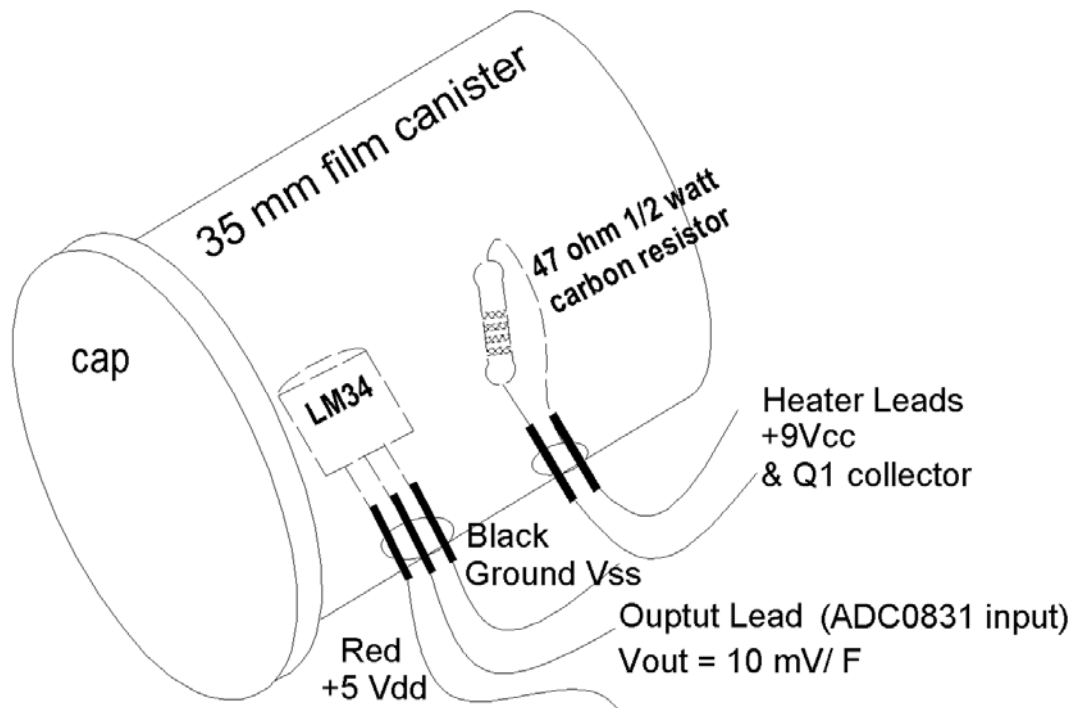
To set up our small environment, you will need the following parts:

- 35 mm plastic film container
- 47-ohm, half-watt carbon resistor with leads
- LM34DZ integrated circuit temperature sensor with leads
- Electrical tape

You will put the 47-ohm resistor and the temperature sensor inside the 35-mm film canister. Leads from these devices will come back to the Board of Education. Placing the cap on the canister creates a closed environment. High current through the resistor will heat the environment, and the sensor will convert temperature to an analog voltage. A current-boost transistor from Experiment #3 will drive the resistor/heater, and you will add an analog-to-digital converter to get binary temperature information into the BASIC Stamp. Figure 4.1 depicts this construction step. Follow the procedures on the next page to construct the canister environment and signal-conditioning circuitry.

**4**

Figure 4.1: Film Canister Heated Environment



### Preliminary Preparation

The 35-mm film canister has two holes drilled in it. The sensor and the “heater” will be placed into these holes.

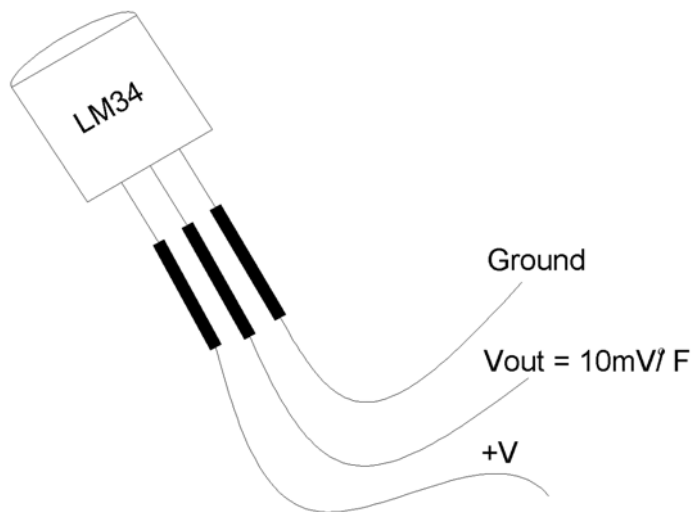
In the last exercise, we controlled the on-off status of a 47-ohm resistor acting as a heating element. The current-boost transistor acts as a switch, controlling the unregulated 9-volt supply to the resistor. As you should have seen, when the transistor turned on, the 9-volt supply was placed across the resistor and it became quite warm, since the power consumed was  $P = V^2/R = 9^2/47$ , or 1.7 watts! This is beyond the resistor’s half-watt rating, but we are using it as a heater. It may become discolored, but should be all right otherwise.

Place the resistor through the lower hole in the canister. Bend the leads and tape them down to the outside of the canister so the resistor is suspended in the middle of the canister.

The LM34 probe will be used to measure the temperature of our system. The LM34 is an excellent sensor in terms of its linearity, cost, and simplicity. The sensor’s output voltage changes 10 mV per degree Fahrenheit and is referenced at 0 degrees. With a DC power supply and a voltmeter, you have a ready-made Fahrenheit temperature sensor.

Refer to Figure 4.2 and the device’s datasheet in Appendix D.

Figure 4.2: LM34 Temperature Probe

**4**

Test your temperature probe by connecting your probe to the +5-volt V<sub>dd</sub> supply and ground. Use your voltmeter to monitor the LM34 output voltage of .01 volts per degree F. Simply move the decimal of the meter reading two places to the right to convert to temperature. For example; .75 V = 75 °F; .825 V = 82.5 °F; 1.05 V = 105 °F, etc. Then, try this:

- Measure and record the room temperature.
- Hold the device between your fingers and watch the temperature rise.
- Hold it until the temperature becomes stable. How hot are your fingertips?
- The LM34 can measure temperatures up to 300 degrees. Briefly wave a flame under it and monitor higher temperatures.

Now, insert the sensor through the top hole of the film canister. Bend and tape its leads to the canister so the sensor is suspended inside. Cap your canister, and your model environment is complete. Keep in mind that although our laboratory setup is small and low power, it could represent controlling the temperature of a large kiln, a brewing vat, or an HVAC system. Appropriate output signal conditioning identified in Experiment #3 can allow the BASIC Stamp to control almost any industrial device.

## Experiment #4: Continuous Process Control

---

Next, let's turn our attention to the Board of Education and set up the circuitry necessary for the experiment. Figure 4.3 represents the four circuits used in the exercises. All four circuits can fit on the Board of Education; you just have to be efficient with your use of the space. Figure 4.3a is a simple active-high pushbutton switch. It will be used to toggle the heater on and off. The output drive circuit depicted in Figure 4.3b also is very simple. Your board will contain the current-boost transistor (from Experiment 3) to drive the heater. Notice that an LED and current-limiting resistor have been added to indicate the status of this drive circuit. Note also that this circuit goes to the +5 V<sub>dd</sub> supply, not the 9-volt unregulated supply.

Since the BASIC Stamp microcontroller does not have analog input capability, we must add an analog-to-digital converter to change the analog output of the LM34 temperature sensor to digital data. A one-step solution to signal conditioning is to use the serial analog-to-digital converter shown in Figure 4.3c. National's ADC0831 is a suitable A/D converter for our application, and will prove to be a very useful device for this and future applications. It's worth a moment at this point to look at the device's features and limitations.

This converter will take an input voltage and convert it to 8-bit digital data with a range of 256 possible binary representations. Potentiometers can be used to externally set the analog voltage range spanned by the 0 to 255<sub>10</sub> digital output capability of the ADC0831. The voltage at V<sub>in(-)</sub> Pin 3 sets the zero value. The V<sub>ref</sub> voltage at Pin 5 sets the voltage span above V<sub>in(-)</sub> over which the resolution of 255 is spread. Setting V<sub>in(-)</sub> to .7 will reference the span of coverage at 70 degrees. V<sub>ref</sub> set to .5 volts results in a span of coverage of 50 degrees.



Figure 4.3: Process Control Circuitry

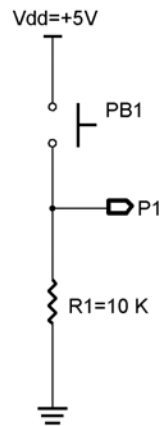


Figure 4.3a: Pushbutton

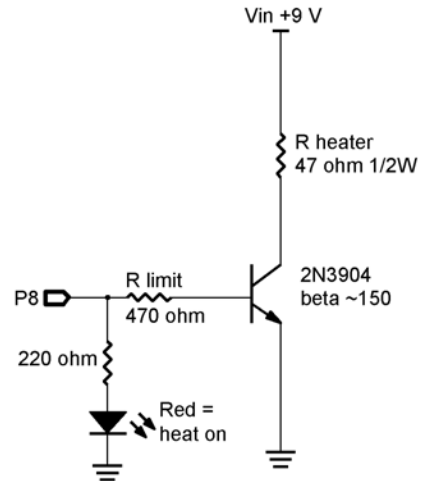


Figure 4.3b: BASIC Stamp Driving Heater with LED Indicator

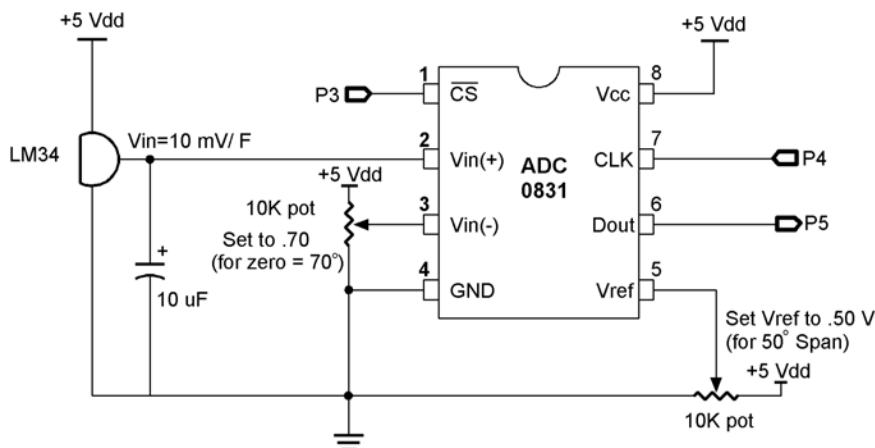


Figure 4.3c: ADC0831 Serial A-to-D



Figure 4.3d: Brushless Fan Directly Connected

## Experiment #4: Continuous Process Control

---

These voltages are used to focus the range of the A/D device to cover the analog voltage being converted. The application will operate from room temperature (70°) up to 120 °F. This temperature range equates to an LM34 voltage output of .70V to 1.20V. To maximize resolution, the span of interest is .5 V (1.2V-.7v); and the zero reference, termed the *offset*, is .7 V. Since we focus on just this range, each binary step represents approximately 0.2 degrees. This allows us to accurately resolve the temperature.

Construct the A/D converter circuit on the Board of Education. By using multi-turn trim potentiometers, the reference and span voltage levels can be set very accurately. Carefully measure and adjust these potentials. Attach the output of the LM34 to the input of the A/D converter.

Controlling the ADC0831 is relatively simple. A program control line tells the device to make a conversion. Once a conversion has been performed, the binary value can be output one bit at a time to the BASIC Stamp. The serial flow of data from the converter is controlled by output pins of the BASIC Stamp driving the “chip select” and “clock” lines. The chip select line (CS) is set low, followed by a low-to-high clock pulse. This starts the conversion. Subsequent clock pulses initiate the transfer of each binary bit starting with the most significant bit first. Parallax provides a convenient instruction, called **SHIFTIN**, specifically designed for controlling synchronous serial communication. Once the binary data is clocked into the BASIC Stamp, it is converted to temperature, based on the zero and spanning values. (The use of the ADC0831 is detailed in the Parallax Basic Analog and Digital text, which can be used as a further reference to this section.)

Program 4.1 has been written to test your converter and driver circuits, and exercise the StampPlot Lite Interface. The first section of the program configures StampPlot Lite. A section that establishes variables and constants follows this. The program is designed for the circuit of Figure 4.3. Double-check the proper connections to I/O Pins 1,3,4,5 and 8. Accurately set the Zero and Span voltages of the ADC0831 to .7 and .5, respectively.

Load the program. Running the program will result in the DEBUG window opening and scrolling values and messages to the screen. Close the DEBUG window and open the StampPlot Lite Interface. At this point, select the appropriate COM port and check “Connect.” Momentarily press the “Reset” button on the BASIC Stamp Board of Education to load the configuration and begin plotting the data. The user-status box will report the current temperature and the output of the A/D converter’s binary and decimal values. The temperature and status of the heater will be plotted on the interface. Pressing PB1 will **TOGGLE** the heater ON and OFF. (Feel free to omit the comments from this code, if you wish).

```

'Program 4.1: Analog-to-Digital & ON-OFF test with StampPlot Interface

'Pushbutton P1 toggles the heater fully ON and OFF. It then establishes
'constants and variables used to acquire data from the ADC0831 serial A-to-D. 'StampPlot is
used to graphically display results. Program assumes that the 'circuitry is set according to
Figure 4.3. ADC0831: "chip select" CS = P3, "clock" 'Clk=P4, & serial 'data output"Dout=P5.
'Zero and Span pins: Digital 0 = Vin(-) = '.70V and Span = Vref = .50V.

'Configure Plot
Pause 500                ' Allow buffer to clear
DEBUG "!RSET",CR         ' Reset plot to clear data
DEBUG "!TITL HEATER CONTROL SAMPLE",CR    'Caption form
DEBUG "!PNTS 6000",CR    ' 6000 sample data points
DEBUG "!TMAX 600",CR     ' Max 600 seconds
DEBUG "!SPAN 70,120",CR  ' 70-120 degrees
DEBUG "!AMUL .1",CR      ' Multiply data by .1
DEBUG "!DELD",CR        ' Delete Data File
DEBUG "!SAVD ON",CR     ' Save Data
DEBUG "!TSMP ON",CR     ' Time Stamp On
DEBUG "!CLMM",CR        ' Clear Min/Max
DEBUG "!CLRM",CR        ' Clear Messages
Debug "PLOT ON",CR      ' Start Plotting
DEBUG "!RSET",CR        ' Reset plot to time 0

' Define constants & variables

CS CON 3                 ' 0831 chip select active low from BS2 (P3)
CLK CON 4                ' Clock pulse from BS2 (P4) to 0831
Dout CON 5               ' Serial data output from 0831 to BS2 (P5)
Datain VAR byte          ' Variable to hold incoming number (0 to 255)
Temp VAR word            ' Hold the converted value representing temp

TempSpan VAR word        ' Full Scale input span in tenths of degrees.
TempSpan = 5000          ' Declare span. Set Vref to .50V and
                          ' 0 to 255 res. will be spread over 50
                          ' (hundredths).

Offset VAR word          ' Minimum temp. @Offset, ADC = 0
Offset = 700

                          ' Declare zero Temp. Set Vin(-) to .7 and
                          ' Offset will be 700 tenths degrees. At these
                          ' settings, ADC output will be 0-255 for temps
                          ' of 700 to 1200 tenths of degrees.

Wkspacel VAR byte        ' Workspace for the PBI's BUTTON command
Wkspacel = 0             ' Clear the workspace before using BUTTON
LOW 8                    ' Initialize heater OFF

Main:

```

## Experiment #4: Continuous Process Control

---

```
GOSUB Getdata
GOSUB Calc Temp
GOSUB Control
GOSUB Display
GOTO Main

Getdata:                                     ' Acquire conversion from 0831
  LOW CS                                     ' Select the chip
  LOW CLK                                    ' Ready the clock line.
  PULSOUT CLK,10                             ' Send a 10 uS clock pulse to the 0831
  SHIFTFIN Dout, CLK, MSBPOST,[Datain\8]    ' Shift in data
  HIGH CS                                     ' Stop conversion
RETURN

Calc Temp:                                   ' Convert digital value to
  Temp = TempSpan/255 * Datain/10 + Offset   ' temp based on Span &
RETURN                                       ' Offset variables.

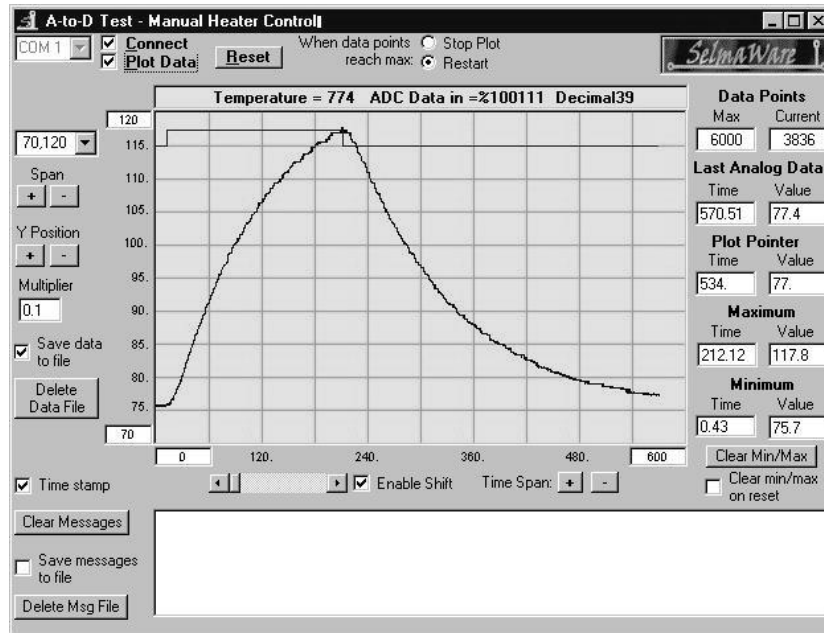
Control:                                     ' Manual heater control
  BUTTON 1,1,255,0,Wkspacel,1,Toggle it
RETURN

Toggle it:
  TOGGLE 8
RETURN

Display:                                     ' Plot Temp, binary ADC, & Temp status
  DEBUG DEC Temp,CR
  DEBUG IBIN OUT8,CR
  DEBUG "!USRS Temperature = ", DEC Temp,"   ADC Data in = %", BIN Datain, "   Decimal",
DEC Datain, CR
RETURN
```

The StampPlot Lite interface will give you a dynamic representation of temperature changes in your canister. Toggle the heater ON and OFF and watch the response. The screen shot in Figure 4.4 represents the closed canister heating to 120 degrees and then cooling after the heater is turned off. Play with your system to become more familiar with its response; then, let's take a little closer look at the subroutines that make up the program.

Figure 4.4: Screen Shot Using Program 4.1



4

The main loop of this program simply executes three subroutines, `Getdata`, `Calc_Temp`, and `Display`. When running, the BASIC Stamp jumps back to the `Getdata` subroutine first. The last line of this routine instructs the processor to `RETURN` to the main loop and executes the next instruction, `GOSUB Calc_Temp`. The `Calc_Temp` subroutine executes, and it ends with a return. The BASIC Stamp returns to `GOSUB_Display`. After `Display` executes, its `RETURN` goes back to the instruction of `GOTO Main` and the process starts over. This is an organized approach to structuring our program. Later, when we include evaluation and control in our program, we simply add another subroutine, such as `GOSUB_Control`.

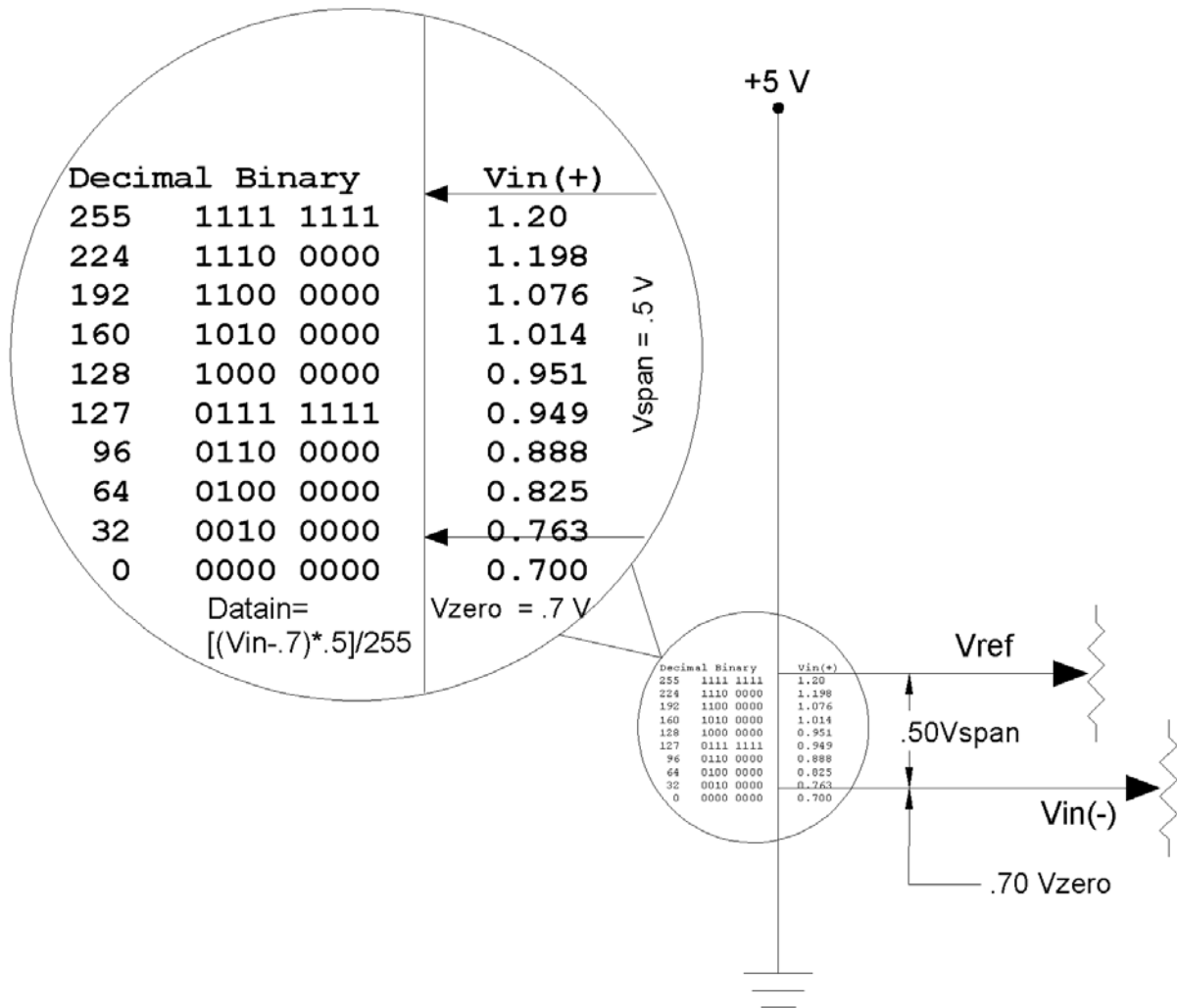
Let's take a closer look at the two primary subroutines of program 4.1. The `Getdata` subroutine begins with a high-to-low transition on the "chip select" line. This readies the A/D for operation.

## Experiment #4: Continuous Process Control

---

The **LOW CLK** and **pulsout CLK,10** instructions tell the A/D converter to make a conversion of the  $V_{in(+)}$  voltage at this time. The ADC0831 is an 8-bit successive approximation converter. Its 256 possible digital combinations are spread over a voltage range determined by the potentials at the  $V_{in(-)}$  and  $V_{ref}$  pins.  $V_{in(-)}$  defines the voltage for which 0000 0000 would be the conversion.  $V_{ref}$  defines the range of input voltages above this point over which the other 255 digital combinations are spread. Figure 4.5 represents the Zero and Span settings for our application.

Figure 4.5: Zero and Span Settings for Our Application



## Experiment #4: Continuous Process Control

---

With these settings, the ADC0831 is focused on a temperature range of 70 to 120 degrees. There can be an infinite number of possible temperature values within the .7 to 1.2-volt output range of the LM34. Only a few representative values are given. Since the 8-bit A/D converter has a resolution of 255, it can resolve this range of 50-degree temperatures to within .31 degrees. The conversion will be a binary number equal to  $[(V_{in} - .7) / .5] * 255$ . Let's try a value within the range. Let's say the temperature is 98.6, which results in an LM34 output of .986 volts.

If  $V_{in} = .986$ , what would be the binary equivalent?

$[(.986 - .7) / .5] * 255 = 145.86$ . The answer is truncated to the whole integer of 145.

The binary word would be 1001 0010.

The binary conversion will be held and ready for transfer.

The **SHIFTIN** instruction is designed for synchronous communication between the BASIC Stamp and serial devices such as the ADC0831. The syntax of the instruction is **SHIFTIN dpin, cpin, mode, [result\bits]**. The parameters indicate:

- which pin data will arrive on (dpin),
- which pin is the clock (cpin),
- (mode) identifies which bit comes first, the least significant (LS) or most significant (MS), and on which edge of the clock it is released, rising (PRE) or falling (POST),
- and, what the word width is and where you want it stored [Datain\8].

For our system, we previously declared Pin 5 as dpin and Pin 4 as the clock (CLK) pin. The ADC0831 outputs the most significant bit first on the trailing edge of the clock. Therefore, MSPOST is the mode. And, finally, the 8-bit data will be held in a byte variable that we declared as **Datain**.

After the binary data is brought into the BASIC Stamp, it is available for our program to use. It would be most convenient to use if it were expressed in terms of the actual measurement units. For our application, that would be in degrees. The next subroutine, **Calc\_Temp**, does just that. By knowing the zero and span transfer function of the conversion process, we use the standard  $y = mx + b$  formula. Where:  $y$  = Temperature,  $m$  = slope of the transfer function, and  $b$  is the offset. Temperature will be resolved and expressed in tenths of degrees.

Refer to the **Calc\_Temp** formula: **Temp = Tempspan/255 \* Datain/10 + 700**

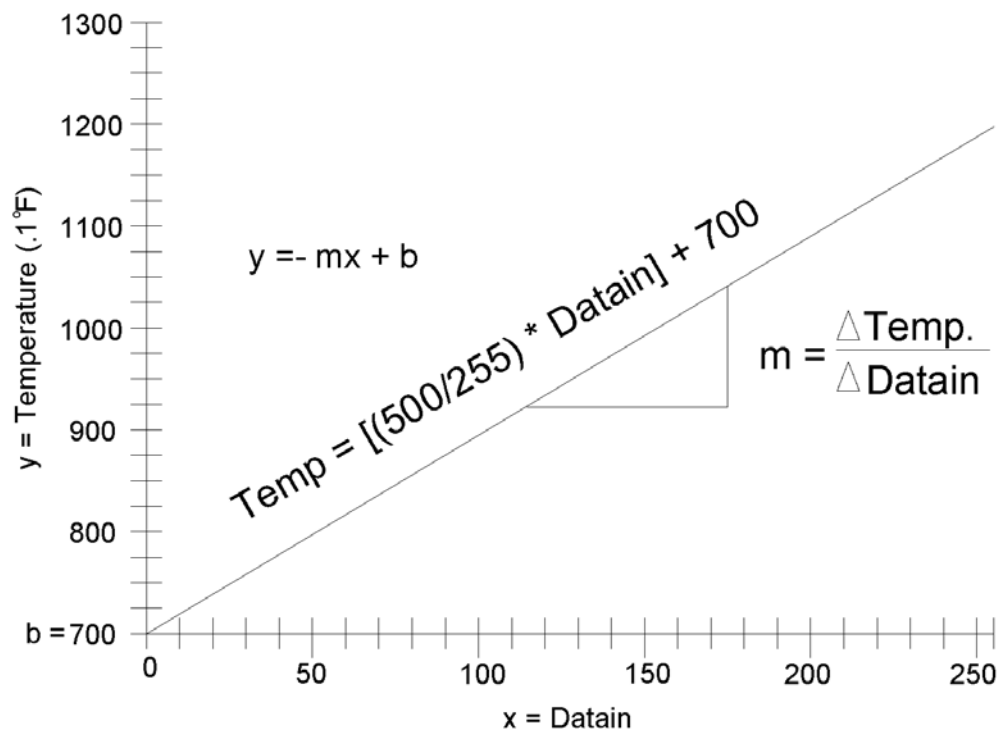


To increase the accuracy in resolving the slope ( $m$ ), the **Tempspan** variable is scaled up by 10, to 5000 hundredth degrees. The slope is therefore,  $5000/255 \approx 19$  or .19 degrees per bit. Multiplying 19 times **Datain** tells you how far the measurement is into the span. This is in one one-hundredth of a degree at this point; therefore, divide by 10 to scale it back to tenths. Adding this to the Zero value of 700 (70 degrees) results in the actual temperature in tenths of a degree. Resolution is approximately .2 degrees over a range of inputs from 70.0 to 120.0 degrees.

The graph in Figure 4.6 plots the transfer function of the input A/D decimal equivalent input to temperature of the canister. Changing the span of coverage changes the slope of the transfer function. Changing the Zero value changes the y intercept.

4

Figure 4.6: Transfer Function



## Experiment #4: Continuous Process Control

---

An additional word of caution about the BASIC Stamp math operation:

- A formula will be executed from left to right unless bracketing is used to set precedence.
- At no point can any subtotal exceed 32,759 or -32,760.
- Also, all remainders will be truncated, not rounded up.

Challenge #1: Change the Zero and Span voltages and edit the program to match the new range.

1. Your system should be able to raise the temperature of the closed canister beyond the 120-degree limit set by Program 4.1. Change the Zero and Span potentiometers for coverage of a temperature range from 75 degrees to 200 degrees. This allows for a wider range of coverage, but what is the resolution of your system now? Be patient, and let your system stabilize. Record the maximum temperature of your system.
2. Set the Zero and Span of your system to focus on the very narrow range of one degree below your room temperature to four degrees above it. Set the `Calc_Temp` variables to display in hundredths of degrees. Track these changes by leaving the cap off of the canister and simply touching the sensor with your warm finger. As you see, the resolution is great, but the trade-off is a decreased range of operation.

Having the ability to control the span and reference of the ADC0831 allows you to focus on a range of analog input. This helps maximize the resolution and accuracy of your system. The following exercise will require the original range of 70 to 120 degrees. Return the Zero and Span potentiometers back to .7 and .5 volts, respectively.

Now, after all of that, we can get back to a study of control theory!

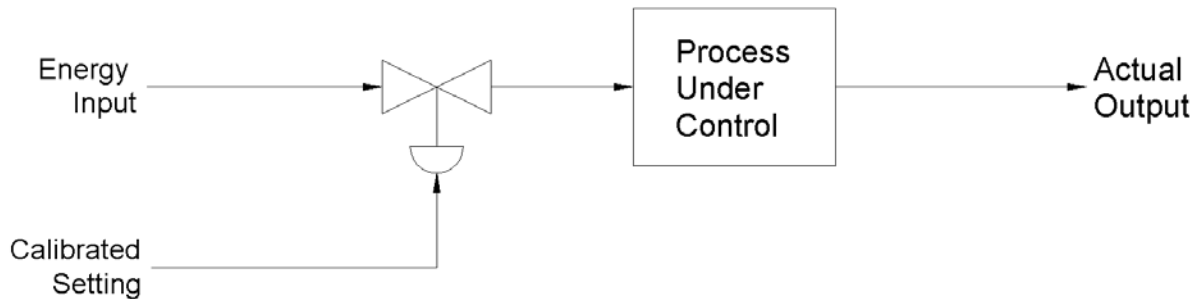
Exercise #2: Open-Loop vs. Closed-Loop Control

## Open-Loop Control

The simplest form of control is open loop. The block diagram in Figure 4.7 represents a basic open-loop system. Energy is applied to the process through an actuator. The calibrated setting on the actuator determines how much energy is applied. The process uses this energy to change its output. Changing the actuator's setting changes the energy level in the process and the resulting output. If all of the variables that may affect the outcome of the process are steady, the output of the process will be stable.

**4**

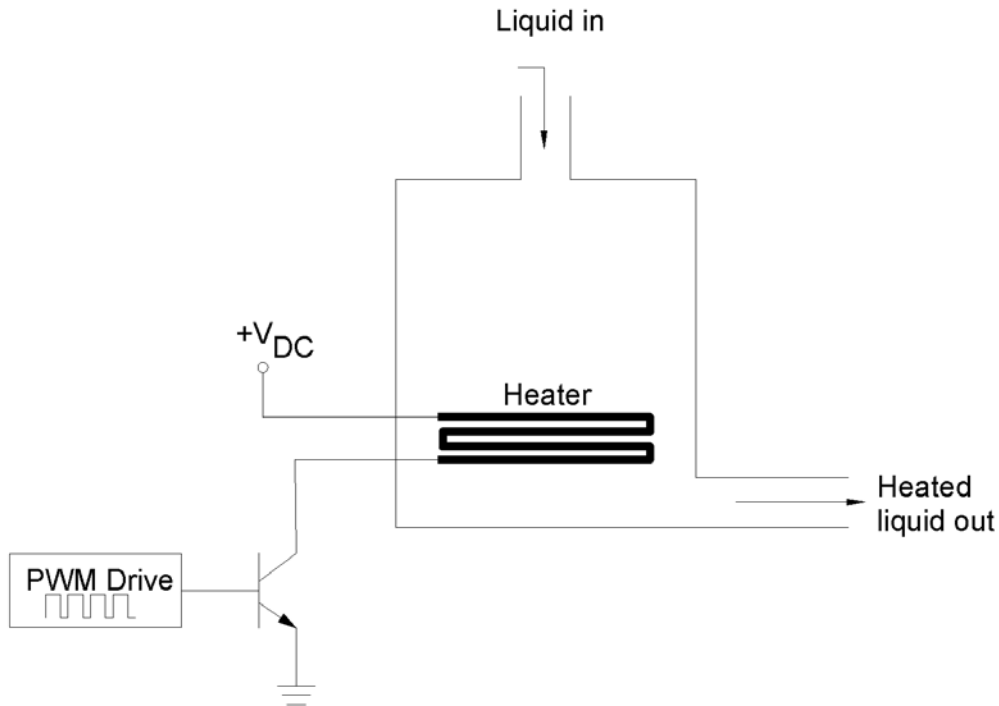
Figure 4.7: Open-Loop Control



The fundamental concept of open-loop control is that the actuator's setting is based on an understanding of the process. This understanding includes knowing the relationship of the effects of the energy on the process and an initial evaluation of any variables disturbing the process. Based on this understanding, the output "should" be correct. In contrast, closed-loop control incorporates an on-going evaluation (measurement) of the output, and actuator settings are based on this feedback information.

Consider the temperature control process shown in Figure 4.8. The material being drawn from the tank must be kept at a  $101^{\circ}$  temperature. Obviously, this will require adding a certain amount of heat to the material. (The drive on the transistor determines the power delivered to the heating element.) The question becomes "How much heat is necessary?"

Figure 4.8: Open-Loop Heating Application



For a moment, consider the factors that would affect the output temperature. Obviously, ambient temperature is one. Can you list at least three others? How about:

- The rate at which material is flowing through the tank.
- The temperature of the material coming into the tank.
- And, the magnitude of air currents around the tank.

These are all factors that represent BTUs of heat energy taken away from the process. Therefore, they also represent BTUs that must be delivered to the process if the desired output is to be achieved. If the drive on the heating element were adjusted to deliver the exact BTUs being lost, the output would be stable.

In theory, the drive level could be set and the desired output would be maintained continuously, as long as the disturbances remained constant.

Let's now assume that it is your objective to keep the interior of your film canister at a constant temperature. A good real-world example would be that of an incubator used to hatch eggs. To hatch chicken eggs, it is important to maintain a 101°F environment.

Turning on the heater will warm up the interior of the canister. In our earlier test, you turned on the heater's drive transistor, and the temperature rose above 101°F. Obviously, to maintain the desired temperature, we will not need to have full power applied to the resistor. Through a little testing, you can determine just what drive level would be needed to yield the correct temperature.

The drive to the power transistor in Figure 4.8 is labeled as PWM. This is the acronym for pulse-width modulation. PWM is a very efficient method of controlling the average power to loads such as heating elements. The square wave is driving the transistor as a current-sinking switch. When the drive is high, the transistor is saturated, and full power is applied to the heater. A logic Low applied as base drive puts the transistor in cutoff; therefore, no current is applied to the load. Multiplying the percentage of the total time that the load receives full power times the full power will give the average power to the load. This average on-time is the duty cycle and is usually stated as a percentage. A 50% duty cycle would equate to half of the full power drive, 75% duty cycle is three-quarters full power, etc. It was stated earlier that the 47-ohm "heater" resistor in our canister would receive 1.7 watts when fully powered by the 9-volt unregulated source supply. The pushbutton switch was used to toggle the power on and off. If you were to press the switch rapidly at a constant rate, the resistor would receive 1.7 watts during the ON time and 0 watts during the OFF time. This 50% duty cycle would result in an average power consumption of .85 watts ( $P_{\text{average}} = P_{\text{full}} * \text{duty cycle}$ ). Complete the table in Figure 4.9 below for power consumed at duty cycles of 75% and 25% for your system.

Figure 4.9: Average Power

$P_{\text{average}} = P_{\text{full}} * \text{duty cycle}$		
Full Power ( $P_{\text{full}}$ )	Duty cycle	Average Power ( $P_{\text{avg}}$ )
1.7 W	100%	1.7 W
1.7 W	75%	
1.7 W	50%	0.85 W
1.7 W	25%	
1.7 W	0%	0 W

## Experiment #4: Continuous Process Control

---

PBASIC provides a useful instruction for providing pulse-width modulation. Its syntax is: **PWM pin, duty, duration**

Where: **pin** is the output pin you are driving.  
**duty** is the duty cycle relative to 255 being 100%.  
**duration** is the window of time in milliseconds over which the duty cycle is provided.

### Challenge 2: Graphing PWM duty vs. Vout.

Use your multi-meter to measure the average voltage across the heating element at various PWM commands. Change the duty variable in Program 4.2 to increments between 0 and 255. Plot the average voltage on the graph in Figure 4.10.

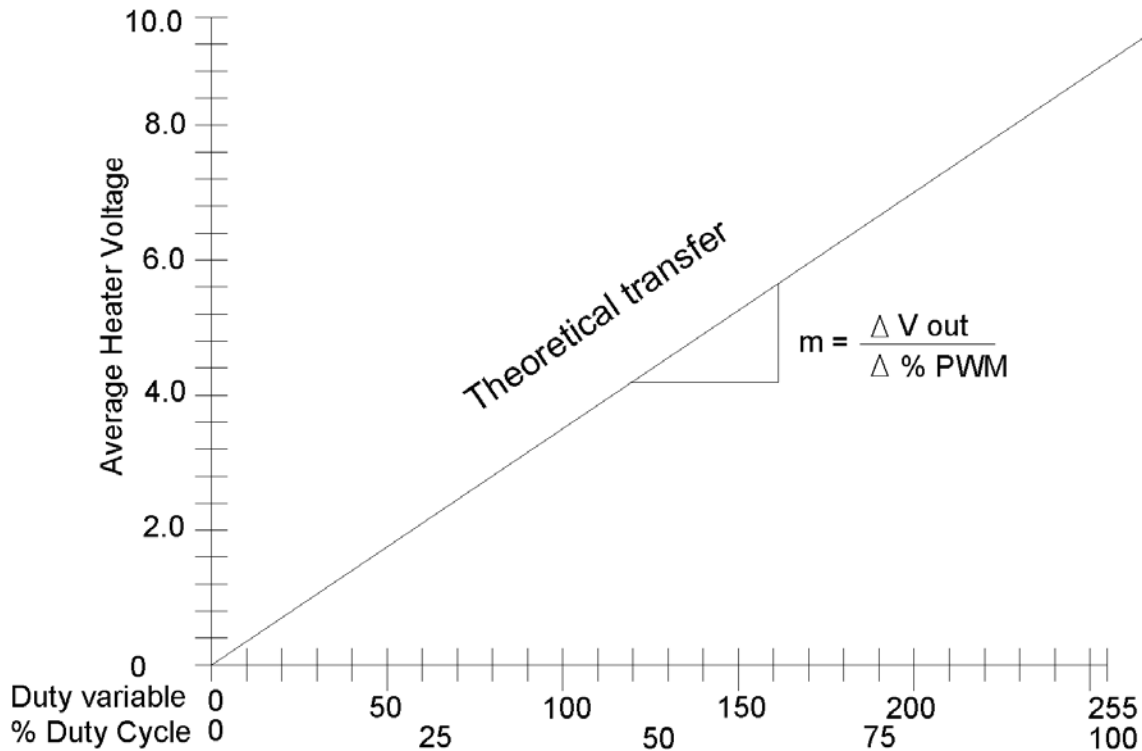
```
'Program 4.2: PWM vs. Vout
' Change the Duty = 50 in increments of 10 between 0 and 100. Measure the
' average output voltage that results.

DutyCycle    VAR    byte
Duty         VAR    byte

DutyCycle = 50                                ' Begin with a 50% duty cycle

Loop:
  Duty = (DutyCycle * 255/100)                 ' Scale DutyCycle to PWM (0-255) duty
  PWM 8, Duty, 200                             ' Apply a PWM of "Duty" to the heater
  DEBUG " Testing at a Duty Cycle of ", DEC DutyCycle, "%.", CR
GOTO Loop
```

Figure 4.10: Graph of Heater Voltage vs. PWM Duty Cycle



4

If you are not familiar with PBASIC's PWM instruction, refer to the BASIC Stamp Manual Version 2.0 pp. 247-250. One aspect of using the command should be understood. PWM applies pulses for a period of time defined by the duration value. During the time when the rest of the program is executing, there is no output applied to the load. As a result, the average voltage at a 100% duty cycle (duty = 255) will result in a value less than the full voltage expected. The slower your program cycle time, the greater is this disparity. To get a better understanding of cycle time, place a `PAUSE 200` in Program 4.2. Compare the resulting output voltage with earlier readings. Change the length of the pause and notice the results.

Recall the Sample and Hold circuit introduced in Challenge #3 of Experiment #2. This circuit held the average PWM voltage across the brushless motor during the entire program loop. This was necessary because of the long program loop and fast response of the fan. The Sample and Hold circuit was effective in delivering the desired average voltage regardless of the program loop-time. In terms of power control, it is more efficient

## Experiment #4: Continuous Process Control

---

to not use Sample and Hold. This can be understood if you consider driving the circuit at 50%. A 50% drive would result in  $\frac{1}{2}$  of the supply voltage appearing across the load continually. This is great. Right? Well if half of the supply voltage is across the load continually, the other half must be across the transistor. This collector-to-emitter voltage times the collector current represents power wasted in the transistor. When system response is fast (like the brushless fan) you have no choice but to use this type of linear power control.

The resistor heating element in our model incubator is a good example of a slow responding system. Straight PWM control of the resistor wastes little power in the transistor because it is only operated in an ON/OFF switching mode. As long as the PWM period is much longer ( $>10x$ ) than the time required to run the rest of the program loop there will little discrepancy in the Duty cycle and expected average voltage.

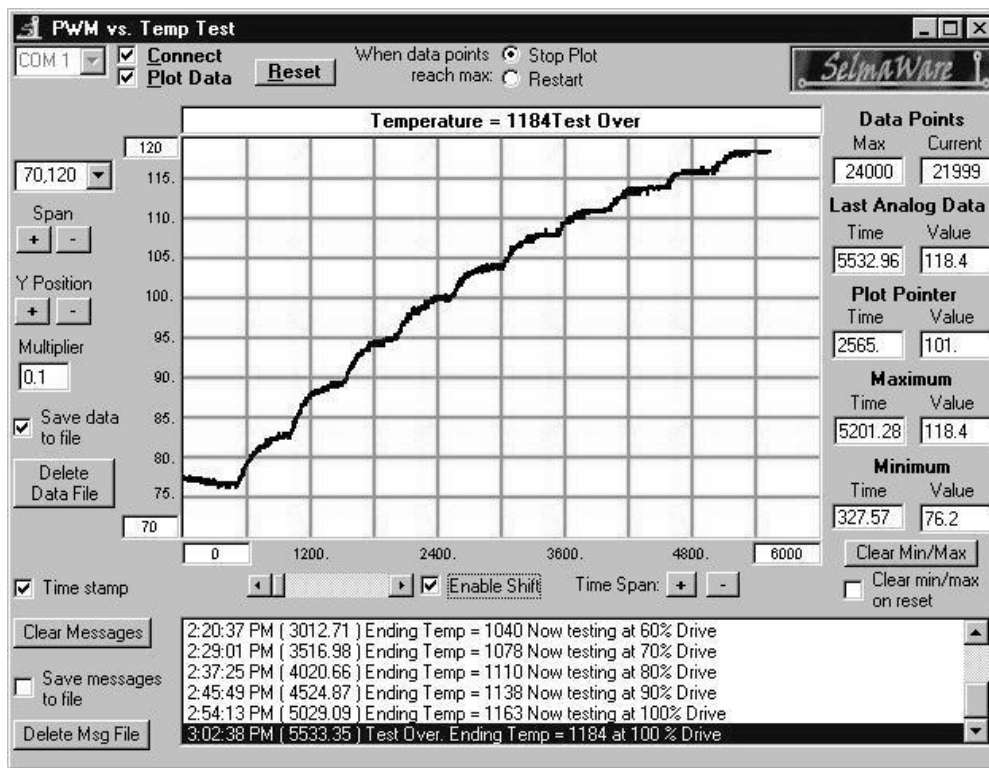


Challenge #3: Analyzing your Open-Loop System

The following program is developed to study the relationship between PWM drive on your heater and the resulting stable temperature. The program will apply PWM drive levels in 10% increments. Each increment will last approximately four minutes. The program will end after 100% drive has been applied. StampPlot Lite will give you a graphical representation of your system's response, along with time stamp information in the list box. Furthermore, if you are really interested, the StampPlot Lite data file can be imported into a spreadsheet, applied to a graph and analyzed.

4

Figure 4.11: Screen Shot of PWM Drive vs. Temperature



## Experiment #4: Continuous Process Control

---

Figure 4.11 is typical of a StampPlot Lite screen shot resulting from this test. Load Program 4.3. Before running the program, be sure your canister has cooled to room temperature. Place the cap on your canister and start the program. When the DEBUG window appears, close it and start StampPlot Lite. Connect using StampPlot Lite and press the restart button to reload the program and begin the test.

```
'Program 4.3: PWM vs. Temp Test with StampPlot Interface

'This program tests the canister's temperature rise for incremental increases of
'PWM drive. Program runtime is approximately 40 minutes. This can be adjusted
'by 'changing the "tick" and/or "Drive" increments.
'Program assumes that the circuitry is set according to Figure 4.3.
'ADC0831: "chip select" CS = P3, "clock" 'Clk=P4, & serial data output"Dout=P5.
'Zero and Span pins: Digital 0 = Vin(-) = .70V and Span = Vref = .50V.

'Configure Plot
Pause 500                                'Allow buffer to clear
DEBUG "!RSET",CR                          'Reset plot to clear data
DEBUG "!TITL PWM vs. Temp Test",CR       'Caption form
DEBUG "!PNTS 24000",CR                   '24000 sample data points
DEBUG "!TMAX 6000",CR                     'Max 6000 seconds
DEBUG "!SPAN 70,120",CR                   '70-120 degrees
DEBUG "!AMUL .1",CR                       'Multiply data by .1
DEBUG "!DELD",CR                          'Delete Data File
DEBUG "!SAVD ON",CR                       'Save Data
DEBUG "!TSMP ON",CR                       'Time Stamp On
DEBUG "!CLMM",CR                          'Clear Min/Max
DEBUG "!CLRM",CR                          'Clear Messages
DEBUG "!PLOT ON",CR                       'Start Plotting
DEBUG "!RSET",CR                          'Reset plot to time 0

' Define constants & variables

CS CON 3                                  ' 0831 chip select active low from BS2 (P3)
CLK CON 4                                  ' Clock pulse from BS2 (P4) to 0831
Dout CON 5                                  ' Serial data output from 0831 to BS2 (P5)
DataIn VAR byte                             ' Variable to hold incoming number (0 to 255)
Temp VAR word                                ' Hold the converted value representing temp
TempSpan VAR word                            ' Full Scale input span in tenths of degrees.
TempSpan = 5000                              ' Declare span. Set Vref to .50V and
                                              ' 0-255 res. will be spread over 50
                                              ' (hundredths).

Offset VAR word                              ' Minimum temp. @Offset, ADC = 0
Offset = 700                                ' Declare zero Temp. Set Vin(-) to .7 and
                                              ' Offset will be 700 tenths degrees. At these
                                              ' settings, ADC output will be 0 - 255 for temps
                                              ' of 700 to 1200 tenths of degrees.

LOW 8                                        ' Initialize heater OFF
```

```

Drive VAR word          ' % Drive
Duty VAR word           ' variable for PWM duty cycle
Tick VAR word

Drive = 0                ' Initialize variable to 0
Tick = 0
Duty = 0

' Get and display initial starting values.

GOSUB Getdata
GOSUB Calc Temp
DEBUG "Temp = ", DEC Temp, " Duty = ", DEC Duty,CR
DEBUG "!USRS Begining Test! -- Testing at ", DEC Drive, "% Drive.",CR

Main:                    ' main loop
  PAUSE 10
  GOSUB Getdata
  GOSUB Calc Temp
  GOSUB Control
  GOSUB Display
  GOTO Main

Getdata:                'Acquire conversion from 0831
  LOW CS                 'Select the chip
  LOW CLK                'Ready the clock line.
  PULSOUT CLK,10        'Send a 10 uS clock pulse to the 0831
  SHIFTIN Dout, CLK, MSBPOST,[Datain\8] 'Shift in data
  HIGH CS               'Stop conversion
RETURN

Calc Temp:              'Convert digital value to
  Temp = TempSpan/255 * Datain/10 + Offset 'temp based on Span &
RETURN                 'Offset variables.

Display:                'Plot present temperature
  DEBUG DEC Temp,CR
RETURN

Control:                ' Testing system at different % duty cycles
  PWM 8,Duty,200        ' PWM
  Tick = Tick + 1      ' increment tick variable
  IF Tick = 2000 Then Increase ' Program cycles per drive level change
RETURN

Increase:               'Bump up the drive
  Drive = Drive + 10    'Drive increments = 10%
  Duty = (Drive * 255/100) 'Scale %Drive to Duty
  If Duty > 256 Then Stopit 'Stop test after 100% PWM
  DEBUG "Ending Temp = ", DEC Temp, " Now testing at ", DEC Drive, "% Drive", CR
  DEBUG "!USRS Testing at ", DEC Drive, "% Drive",CR

```

## Experiment #4: Continuous Process Control

---

```
Tick = 0
RETURN

Stopit:                                ' Stop and print summary
  DEBUG "Test Over. Ending Temp = ", DEC Temp," at 100 % Drive",CR
  DEBUG "!USRS Temperature = ", DEC Temp,"Test Over",CR
END
```

Challenge #4: Open-Loop Control--Desired Setpoint = 101° Fahrenheit

It is our objective to maintain a constant canister temperature of 101° Fahrenheit. Follow these procedures. Record values in the table of Figure 4.12.

1. Study the StampPlot Lite analysis that resulted from running Program 4.2. From the Text Box listing, record in the table the beginning ambient temperature, the temperature at the end of the 50% drive test, and the ending maximum temperature after 100% drive.
2. Use your cursor to find the Drive level that resulted in a temperature of 101 degrees.
3. Next, modify the `Control1` subroutine of Program 4.2 so that the duty cycle remains at the constant value declared initially. Do this by removing the two lines indicated below.

```
Control:                                     ' Testing system at different % duty cycles
PWM 8,duty,200                               ' PWM
tick = tick + 1                          ' increment tick variable
IF tick = 2000 Then Increase            ' Program cycles per drive level change
RETURN
```

4. At the beginning of the program, declare the `DutyCycle` to be the value that yielded 101° in our test StampPlot Lite. Run the program and allow the system to stabilize. How close was your estimation? Bump it up or down accordingly to find the setting that yields the desired result. In line #5 of the table, record the percent of drive that places the system at or near 101 degrees. Let it run for a moment and take note of the system stability. Once a drive setting has been established, an open-loop system will stabilize; and, as long as the disturbances that affect the process stay constant, so will the output.
5. Plug in the brushless fan across the Vdd supply and aim it directly toward the canister. The moving air represents a change in the disturbance on your process. According to theory, heat will be removed from the process at a greater rate and the new stable temperature will be lower than 101°. With the fan blowing on the canister, try to find the new “correct” drive for this condition. Record your data in line #6 of the table.

4

Figure 4.12: Open-Loop Control Table

Line#	Condition	% Drive	Temp
#1	Desired Temperature		101°
#2	Ambient Temperature	0	
#3	50 % Drive Temperature	50%	
#4	Full Drive Temperature	100%	
#5	Appropriate % Drive for 101° Without fan disturbance		101°
#6	Appropriate % Drive for 101° With direct fan disturbance		101°
#7	Appropriate % Drive for 101° With partial fan disturbance		101°

6. Finally, leaving the proper setting established in line #6, change the position of the fan so it is blowing less directly on the canister. This represents a medium disturbance level on the system. Assess the situation and make your best guess as to the proper drive setting required by this new condition. Program the BASIC Stamp for this drive level. Once the system stabilizes, record your results in line #7 of the table.

Challenge #5: Determining an Open-loop Setting

1. Select a new “desired temperature” for your system. Predict and program an open-loop drive value that will maintain this temperature.
2. Place a couple of glass marbles in your canister. See how increasing the mass of the system affects the response and the drive setting necessary to maintain the new condition. What conclusions can you draw from the system’s behavior?

There are many variables that can affect the relationship of drive level and temperature in your small environment. Given some time to experiment and become familiar with the dynamic relationship between temperature, drive level, and disturbances, you could get pretty good at assessing the conditions and setting the right amount of drive in an open-loop manner. As we see, however, if any condition of our process changes, so will the output. Open-loop control can be useful in some applications. When a process requires that its output remain constant for all conditions, then closed-loop control must be employed. In closed-loop control, action is taken based on an evaluation of the measurement and the desired setpoint. This evaluation results in what is called an “error signal.” Experiment #5 and #6 will guide you through 5 modes of closed-loop control.



## Questions and Challenge

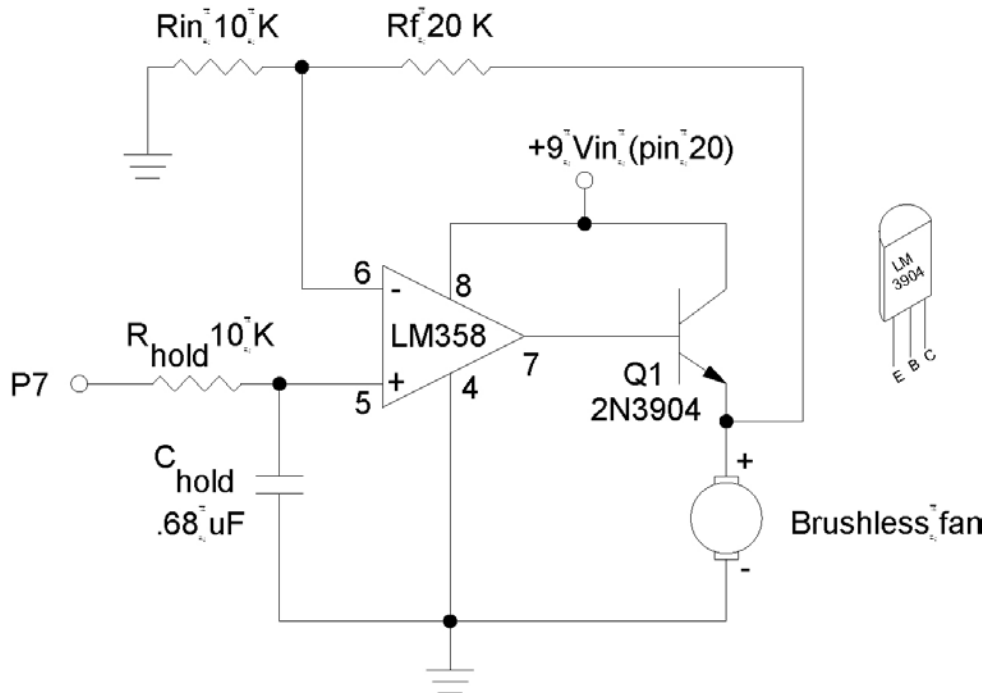
4

1. Give two examples of continuous process control other than those given in the text.
2. How is the drive level in open-loop control determined?
3. What is the primary advantage of open-loop control?
4. What is the primary disadvantage of open-loop control?
5. The ADC0831 will convert a range of analog input to one of 256 possible binary values. The number 256 identifies the \_\_\_\_\_ of the converter.
6. The purpose of the chip select and clock lines to the ADC0831 are to \_\_\_\_\_ the conversion process.
7. If the LM34 were placed in a 98.6-degree environment, the expected output would be \_\_\_\_\_ volts.
8. In pulse-width modulation, the amount of drive action is based on the \_\_\_\_\_ of time ON over the total time.
9. If a 40-watt heater were pulse-width modulated at a 75% duty cycle, the average power consumed would be \_\_\_\_\_ watts.
10. When disturbances change in an open-loop process, so does the \_\_\_\_\_.

### Challenge: Open-loop Control of the Fan

Recall in Experiment #2, the circuit in Figure 4.13 was introduced to control the speed of the brushless motor fan.

Figure 4.13: Sample and Hold PWM Drive



Because of the fan's quick response to voltage fluctuations, the Sample and Hold circuit is necessary to effectively control speed using the PWM instruction. Construct this circuit to be able to vary fan speed. Directly connect the resistor across the +V<sub>in</sub> supply. With the fan directly pointed toward the canister, experiment with different PWM drive levels to the fan. What drive level is necessary to cool the canister to 101°F?

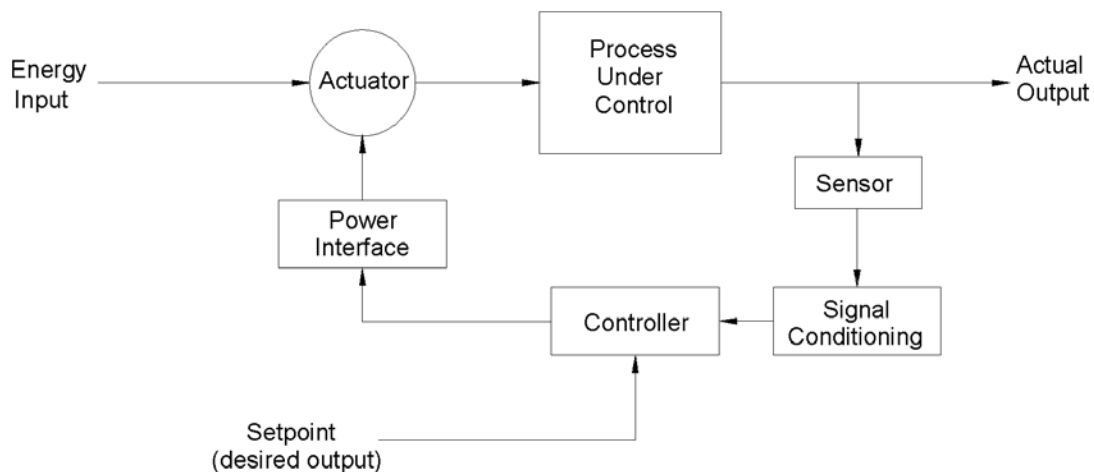




## Experiment #5: Closed-Loop Control

An open-loop control system can deliver a desired output if the process is well understood and all conditions affecting the process are constant. However, Experiment #4 showed us that an open-loop control system couldn't guarantee the desired output from a process that was subject to even mild disturbances. There is no mechanism in an open-loop system to react when disturbances affect the output. Although you were able to find a drive setting that would yield the desired temperature in Experiment #4, when the fan was moved closer or further from the heater, the fixed setting was no longer valid. Closed-loop control provides automatic adjustment of a process by collecting and evaluating data and responding to it accordingly. A typical block diagram of an automatic control system is depicted in Figure 5.1.

Figure 5.1: Closed-Loop Control



In this diagram, an appropriate sensor is measuring the Actual Output. The signal-conditioning block takes the raw output of the sensor and converts it into data for the Controller block. The Setpoint is an input to the Controller block that represents the desired output of the process. The controller evaluates the two pieces of data. Based on this evaluation, the controller initiates action on the Power Interface. This block provides the signal conditioning at the controller's output. Experiment #3 discussed several methods of driving power interface circuits. The Power Interface has the ability to control the Actuator. This may be a relay, a solenoid valve, a motor drive, etc. The action taken by the Actuator is sufficient to drive the Actual Output toward the desired value.

## Experiment #5: Closed-Loop Control

---

As you can see, this control scenario forms a loop, a closed-loop. Furthermore, since it is the process's output that is being measured, and its value determines actuator settings, it is a feedback closed-loop system. The input changes the process output → the output is monitored for evaluation → the evaluation changes the input → that changes the process output, etc., etc.

The type of reaction that takes place upon evaluation of the input defines the process-control mode. There are five common control modes. They are on-off, on-off with differential gap, proportional, integral, and derivative. The fundamental characteristic that distinguishes each control mode is listed below in Table 5.1.

Table 5.1: Five Common Control Modes

Process Control Mode	Evaluation	Action
On-off	Is the variable above or below a specific desired value?	Drive the output fully ON or fully OFF.
On-off with differential gap	Is the variable above or below a range defined by an upper and lower limit?	Output is turned fully ON and fully OFF to drive the measured value through a range.
Proportional	How far is the measured variable away from the desired value?	Take a degree of action relative to the magnitude of the error.
Integral	Does the error still persist?	Continue taking more forceful action for the duration the error exists.
Derivative	How fast is the error occurring?	Take action based on the rate at which the error is occurring.

Continue taking more forceful action for the duration the error exists.

This exercise will focus on converting the open-loop temperature control system of Experiment #4 into an on-off closed-loop system. Our system will show advantages and disadvantages to this method of control. The characteristics of the system being controlled determines how suitable a particular control mode will be. Experiment #6 will use the same circuitry to overview and apply proportional, integral, and derivative control modes. Leave the circuit constructed after completing this step.

Figure 5.2 is a schematic of the circuitry necessary for the next two exercises. As you see, this is identical to Experiment #4. The 35-mm film canister provides the environment we wish to control. The heater drive provides full power for developing heat in the resistor. The LED is also driven by Pin 8. Remember that the LED is driven by the +5-V<sub>dd</sub> supply, and the heater is driven by the +9-volt unregulated line supply. The LM34 sensor will provide temperature data. In closed-loop control, we will monitor the temperature and use it to determine control levels. The fan's air currents will act as a disturbance to the process.

Figure 5.2: Closed-Loop Control Circuitry

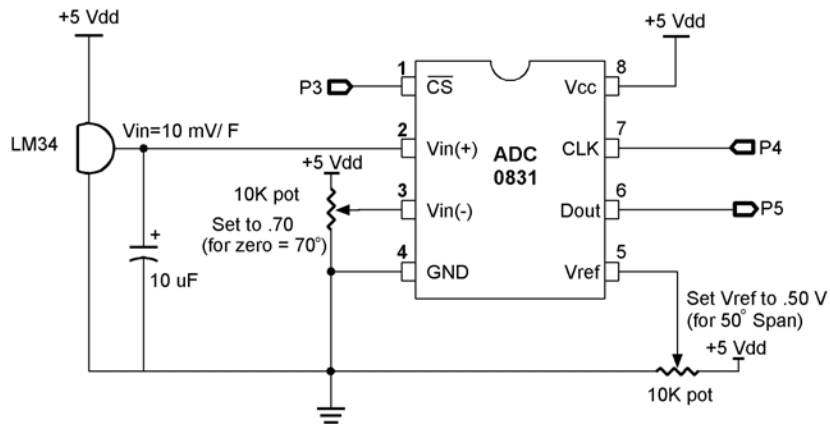


Figure 5.2a: LM34 to ADC0831 Serial A-to-D

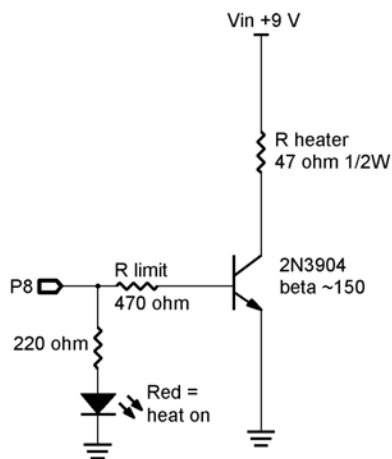


Figure 5.2b: BASIC Stamp Driving the Heater with LED Indicator



Figure 5.2c: Brushless Fan - Directly Connected

5

## Experiment #5: Closed-Loop Control

---

If the circuit isn't already on your board, carefully construct it. Use space on the small Board of Education efficiently to allow for the circuitry. Take your time, plan your layout, and be careful not to inadvertently short any wires. Refer back to Experiment #4 for details on the film canister construction, the operation of the LM34, and the use of the ADC0831 analog-to-digital converter.

Double-check the Zero and Span voltages of the ADC0831. Use your voltmeter to set the Zero voltage ( $V_{in(c)}$ ) to .7 and the Span ( $V_{(ref)}$ ) to .5 volts. This will establish a full-scale temperature measurement range from 70 to 120 degrees F.



### Exercises

#### Exercise #1: Establishing Closed-Loop Control

Let's assume it is our objective to maintain temperature within the canister at  $101.50\text{ }^{\circ}\text{F} \pm 1$  degree. This would be representative of the requirements of an incubator used for hatching eggs. Maintaining the eggs at the setpoint temperature of  $101.5\text{ }^{\circ}\text{F}$  is perfect, but the temperature could go up to  $102.50$  or down to  $100.50$  without damage to the embryos. Although it may be hard to imagine an incubator when you look at your film canister, the BASIC Stamp would be well suited as the controller in a large commercial hatchery incubator.

To maintain temperature at the desired value seems like a pretty "common sense" task. That is, simply measure temperature; if it is above the setpoint, turn the heater OFF; and, if it is below, turn the heater ON. The simplest kind of control mode is on-off control. There are drawbacks to this control mode, however. During the following exercise, you will establish on-off control of your model incubator. Pay close attention to the characteristics exhibited by your model. These characteristics would also apply to real control applications.

#### Procedure

Programming for this application requires data acquisition, evaluation, and control action. Our display routine will also include storing and displaying the minimum and maximum overshoot in the process.

The structure and much of the content of Program 4.1 may be used to acquire and calculate our measurement. Instead of turning the heater on continually, a new subroutine will be added to evaluate and control it. Evaluation will be based on a `setpoint` variable. Refer to Program 5.1 following.

```
'Program 5.1: Simple ON/OFF Control with the StampPlot Interface

'This program establishes simple ON/OFF control of the model incubator.
'Program I/O is based on the circuitry of Figure 5.2.
'Zero and Span voltages: Digital 0 = Vin(-) = .70V and Span = Vref = .50V.

'Configure Plot
Pause 500                                'Allow buffer to clear
DEBUG "!RSET",CR                          'Reset plot to clear data
DEBUG "!TITL Simple ON/OFF Control",CR    'Caption form
DEBUG "!PNTS 60000",CR                    '60000 sample data points
DEBUG "!TMAX 300",CR                       'Max 300 seconds
DEBUG "!SPAN 70,120",CR                    '70-120 degrees
DEBUG "!AMUL .1",CR                        'Multiply data by .1
DEBUG "!DELD",CR                           'Delete Data File
DEBUG "!CLMM",CR                           'Clear Min/Max
DEBUG "!CLRM",CR                           'Clear Messages
DEBUG "!USRS ",CR                          'Clear User status bar
DEBUG "!SAVD ON",CR                        'Save Data
DEBUG "!TSMP ON",CR                        'Time Stamp On
DEBUG "!SHFT ON",CR                        'Enable plot shift
DEBUG "!PLOT ON",CR                        'Start Plotting
DEBUG "!RSET",CR                           'Reset plot to time 0

' Define constants & variables

CS CON 3                                  ' 0831 chip select active low from BS2 (P3)
CLK CON 4                                  ' Clock pulse from BS2 (P4) to 0831
Dout CON 5                                  ' Serial data output from 0831 to BS2 (P5)
Datain VAR byte                             ' Variable to hold incoming number (0 to 255)
Temp VAR word                                ' Hold the converted value representing temp

TempSpan VAR word                           ' Full Scale input span in tenths of degrees
TempSpan = 5000                             ' Declare span 50 (1/100ths degrees)

Offset VAR word                             'Minimum temp. Offset, ADC = 0
Offset = 700                                'Declare zero Temp. Set Vin(-) to .7 and
                                             'Offset will be 700 tenths degrees. At these
                                             'settings, ADC output will be 0 - 255 for
                                             'temps
                                             'of 700 to 1200 tenths of degrees.

Setpoint VAR word                           ' Initialize setpoint to 101.5 degrees
Setpoint = 1015

MMFlag VAR bit                               '
MMFlag = 0

LOW 8                                        ' Initialize heater OFF

Main:
```

## Experiment #5: Closed-Loop Control

---

```
GOSUB Getdata
GOSUB Calc Temp
GOSUB Control
GOSUB Display
GOTO Main

Getdata:                                'Acquire conversion from 0831
  LOW CS                                'Select the chip
  LOW CLK                                'Ready the clock line.
  PULSOUT CLK,10                         'Send a 10 uS clock pulse to the 0831
  SHIFTFIN Dout, CLK, MSBPOST,[Datain\8] 'Shift in data
  HIGH CS                                 'Stop conversion
RETURN

Calc Temp:                               'Convert digital value to
  Temp = TempSpan/255 * Datain/10 + Offset 'temp based on Span &
RETURN                                  'Offset variables.

Control:                                 'ON/OFF control
  IF Temp > Setpoint THEN OFF
  HIGH 8                                  'Heater ON
RETURN

OFF:                                      'Heater OFF
  LOW 8
RETURN

Display:                                 'Plot Temp and heater status
  IF OUT8 = 0 AND MMFlag = 0 THEN MMClear ' Clear Min/Max
  DEBUG DEC Temp,CR
  DEBUG IBIN OUT8,CR
RETURN

MMClear:                                 'Clear Min/Max When setpoint is first reached.
  DEBUG "!CLMM",CR
  DEBUG "!USRS Overshoot/Undershoot Test Ready",CR
  MMFlag = 1
RETURN
```

Run the program and observe the behavior of the system. StampPlot Lite will graphically plot the temperature response of the system and the on-off status of Output 8. Follow the StampPlot Lite procedures of running the program, closing the debug window, opening StampPlot Lite, and pressing the “reset” button.

When you start your system, the heater will be on as indicated by the LED. The heater/resistor becomes quite hot when full power is applied. This heat transfers through the environment and warms the temperature sensor. When the sensor has heated to 101.5, the BASIC Stamp will turn off the heater. For a period after the heater is turned off, the temperature continues to rise. This is called overshoot. At this point, it is important to understand the dynamics of your system. The heat held within the mass of the resistor will continue to dissipate into the air, the air becomes warmer, and the LM34 reports that overshoot has occurred. Similar to the mechanical inertia of a moving object, this phenomenon is called thermal inertia. Overshoot becomes large when the heat energy contained in the mass of the resistor is large, relative to the heat already in the canister. The 35-mm canister is small, but the mass of the half-watt resistor also is small. As a result, the overshoot of your system will probably be less than one degree.



When the temperature does turn around and begin to fall, as it passes the setpoint, the heater is once again turned on. Undershoot will occur for similar reasons as did the overshoot. During the time the heater is coming up in temperature, the ambient temperature has continued downward. Continuous cycling above and below the desired setpoint is typical of on-off control. The rate of this cycling and the degree of the overshoot depend on the characteristics of the system. On-off control is suitable for processes that have large capacity, can tolerate sluggish response, and sustain a relatively constant level of disturbance. If our incubator were large, well insulated, and kept in a constant room environment, on-off control would be acceptable. After the process has had a chance to cycle a few times, record the minimum and maximum overshoot values.

Maximum overshoot \_\_\_\_\_ Minimum Overshoot \_\_\_\_\_

Using your cursor, investigate time between cycles. Record these times below.

Time at which the heater first turned OFF ( $T_{1off}$ ): \_\_\_\_\_

Time at which the heater turned back ON ( $T_{1on}$ ): \_\_\_\_\_

Time at which the heater turned OFF again ( $T_{2off}$ ): \_\_\_\_\_

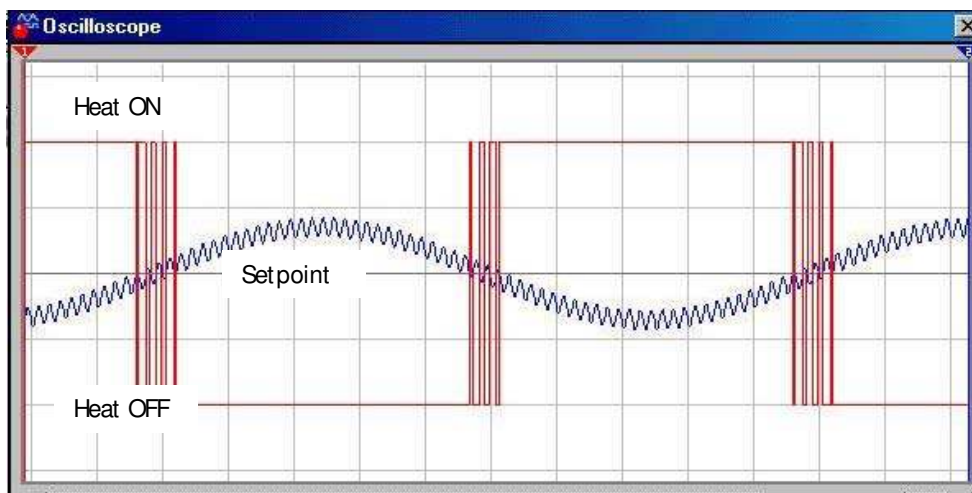
Cycle time = ( $T_{2off}$ ) - ( $T_{1on}$ ): \_\_\_\_\_

## Experiment #5: Closed-Loop Control

---

A major problem with on-off control is that the output drive may cycle rapidly as the measurement hovers about the setpoint. Noise riding on the analog sensor measurement would be interpreted as rapid fluctuation above and below the setpoint. The timing diagram in Figure 5.3 represents this problem

Figure 5.3: On-Off Control When Noise Rides on the Data



The slow-moving data that is cycling through the setpoint has a high-frequency noise component riding on it. As you can see, the coupled effects of the noise results in the data passing above and below the setpoint several times. The microcontroller would attempt to turn the heating element on and off accordingly.

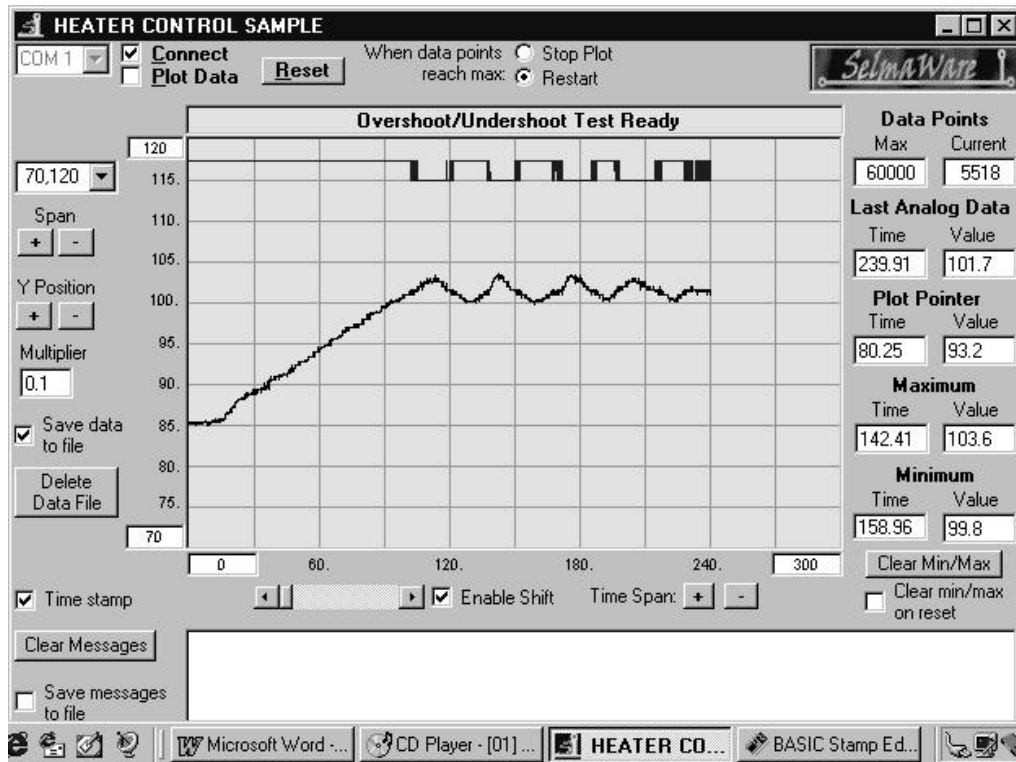
In an actual incubator application where larger amounts of power are controlled, this rapid switching could cause unwanted RF noise. This rapid cycling could also be damaging to electromechanical output elements such as motors, relays, and solenoids.

Do you observe the rapid cycling of the LED as the temperature approaches the setpoint? \_\_\_\_\_

Remove the 10- $\mu$ F capacitor from across the sensor output. Does this increase the cycling problem? Why? Figure 5.4 is a typical screen shot resulting from this experiment. Overshoot and rapid cycling are a problem. Is this similar to your system's response?



Figure 5.4: Simple ON/OFF Control



5

## Experiment #5: Closed-Loop Control

---

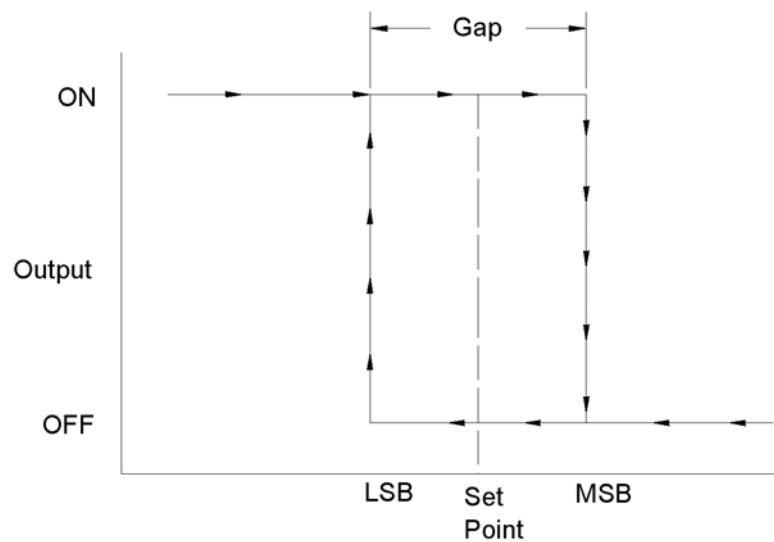
### Challenge! Change the Dynamics of Your System

1. Connect your brushless fan to the Vcc supply and aim it toward the canister. Reset the program and watch the control action. Describe any effect that the new level of disturbance has on the overshoot and/or the cycling. Summarize how you have changed the dynamics of your system.
2. Place a single glass marble in your canister. Reset the program and watch the control action. Describe any effect that changing the mass of your system has on the overshoot and/or the cycling. Summarize how you have changed the dynamics of your system.

### Exercise #2: Differential-Gap Control

The rapid cycling resulting from noise or the measurement hovering around a single setpoint is the biggest disadvantage of simple on-off control. Most practical on-off control systems lend themselves to allowing a minimum and maximum value of measurement. An incubator is a good example. Although the desired temperature is 101.5 degrees, it allows  $\pm 1$  degree of variance about the setpoint. Differential-gap control is a mode of control that takes action based on the measurement crossing a defined upper and lower limit. When the measured value goes beyond one limit, full appropriate action is taken to drive the temperature to the opposite limit. Full opposite action is then taken to drive the process back again. Figure 5.5 graphically diagrams the action taken by differential-gap control. When the system is started and its temperature is below the Lower limit, the heater will come on and the temperature rise. When the temperature passes the upper limit, the heater is turned OFF, heat will begin to leave the process, and the temperature will begin to drop to below the Lower limit. The heat then is turned back on and the cycle begins again.

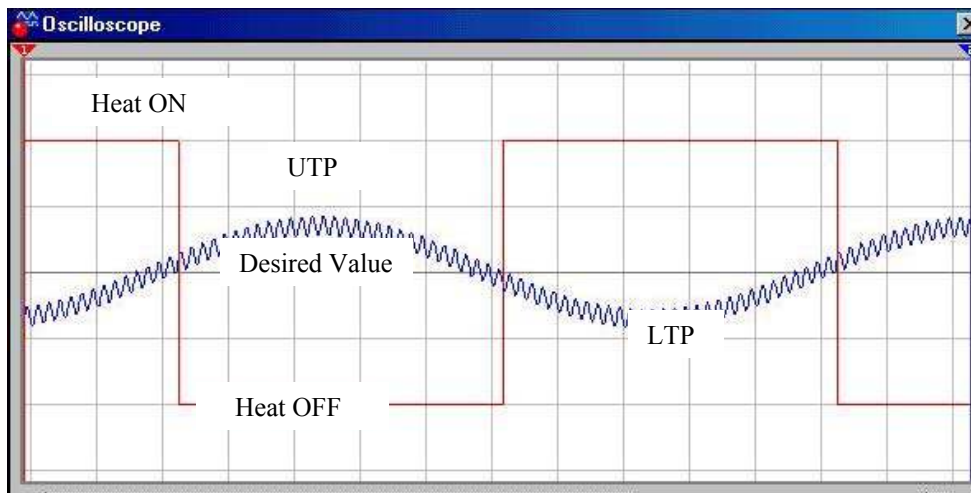
Figure 5.5: Differential-Gap Control Action

**5**

The result is a slower on-off cycle time and no cycling resulting from noisy data. Notice how the timing diagram in Figure 5.6 differs from the earlier one depicting noisy data in a simple on-off control mode.

Whenever the measurement is anywhere between these limits, the heater state is not changed. The result is a slower ON/OFF cycle time and no cycling from noisy data. The heater is switched when the data (+noise) passes a limit. After that, the data (+noise) has to exceed the other limit before switching will occur again. Because the differential gap is wider than the effect of the noise, rapid cycling is eliminated.

Figure 5.6: Differential-Gap Control Action when Noise is Riding on the Data



These advantages come at the compromise of allowing the measured variable to drift further from the desired “average” value. The thermal inertia of our system will still result in some amount of overshoot and undershoot. We are accepting a wider variance in temperature. When processes allow this variance, differential-gap control is usually preferred over simple on-off control.

The `control` subroutine can be easily changed to accommodate differential-gap control. Make the following modifications to Program 5.1.

Declare the new variables of `Upper_limit` and `Lower_limit` at the beginning of the program and initialize them to 102 °F and 101 °F respectively.

```
Upper_limit VAR word
Lower_limit VAR word
Upper_limit = 1020           '102 degrees in 1/10ths
Lower_limit = 1010          '101 degrees in 1/10ths
```

Next, replace the on-off `Control1` subroutine with the following code.

```
Control:
  IF Temp > Upper_limit THEN OFF      ' Over upper limit then Heat OFF
  IF Temp < Lower_limit THEN ON       ' Under lower limit then Heat ON
RETURN                                 ' return leaving heat in last state

OFF:
  Out8 = 0                            ' Heater Off
  MMFlag = 1
RETURN

ON:
  Out8 = 1                            ' Heater On
RETURN
```



Run the program and observe the behavior of your system.

Challenge! Observe and Evaluate Differential-Gap Control

Allow the program to cycle a few times. Report on the following:

Record the minimum and maximum overshoot temperatures.

Maximum overshoot \_\_\_\_\_ Minimum Over shoot \_\_\_\_\_

With your cursor, investigate time between cycles. Record these times below.

Time at which the heater first turned OFF ( $T_{1off}$ ) \_\_\_\_\_

Time at which the heater turned back ON ( $T_{1on}$ ) \_\_\_\_\_

Time at which the heater turned OFF again ( $T_{2off}$ ) \_\_\_\_\_

Cycle time =  $(T_{2off}) - (T_{1off})$  \_\_\_\_\_

## Experiment #5: Closed-Loop Control

---

Project the switching point of the digital output down to the plotted temperature. Can you determine the temperature at which switching occurred? Momentarily remove the 10- $\mu$ F filter capacitor. Does the increase in noise cause rapid cycling about the limits?

Use the fan to change the disturbances to the process. Reset the program and watch the control action. Describe any effect the new level of disturbance has on the overshoot and/or the cycling. Summarize how you have changed the dynamics of your system.

Place a single glass marble in the canister. Restart the program and summarize how increasing the mass has affected the process control action. Investigate the cycle time and overshoot.

### Simple ON/OFF or Differential Gap?

Hopefully, the data that you have observed and recorded will reveal some important characteristics of these two control modes. They both have advantages and disadvantages. Simple on-off control results in rapid cycling of the heating element. Reported cycle times of less than one second could easily result if your system has fast recovery or there is noise on the analog line. Rapid cycle time would not be acceptable if our heater were being controlled by an electromechanical relay.

Notice, however, that the overshoot is approximately a half-degree and our average temperature is at the desired setpoint. Compare this control response to that observed when Differential Gap has been added to the On/OFF control.

With Differential-Gap control, you will notice fundamental differences in the control action.

1. Rapid cycling about the setpoint no longer occurs.
2. The minimum and maximum values still overshoot, but now beyond the limits.
3. Total cycle time between ON/OFF conditions is longer.

Increased cycle time and noise immunity about the setpoint are definite improvements over simple on-off control. The tradeoff, however, is allowing the process to vary further from the desired temperature setpoint. Obviously, an understanding of your process and its hardware will determine the appropriate control mode.

Both modes took appropriate control action to maintain temperature under changing disturbance levels and load conditions. The drawback to either mode of ON/OFF control is that the controlled variable is constantly on the move. The fully-ON and fully-OFF conditions of the final control element are continually forcing the measurement past the limits.

If you recall, in the Open-Loop Control exercise of Experiment #3, a value of drive between off and fully on was found to be appropriate to hold the temperature at the setpoint. If all disturbances to the process remained constant, the temperature would stay at the setpoint when the right percentage of drive was applied. We also saw that as conditions changed, so did the measurement. Experiment #6 will investigate controls that take an appropriate amount of action based on an evaluation of the measurement. Applying Proportional, Integral, and Derivative control theory can be employed to maximize the effectiveness of the control system.



#### Programming Challenges

1. Alter your program so a  $\pm 1$  degree differential gap can be calculated automatically, based on the desired setpoint.
2. Add a variable called Differential\_gap so the operator needs only to enter the Setpoint and the amount of Differential gap and the program automatically performs the desired action.



## Questions and Challenge

1. Write one complete sentence that clearly states the fundamental difference between Open-loop and Closed-loop controls.
2. Simple ON/OFF control compares the measurement of the process variable to a \_\_\_\_\_.
3. If the final control element to our model incubator were an air conditioner instead of a heater, what would change about Program 5.1?
4. When the process variable continues to increase after the final control element is turned OFF, it is termed \_\_\_\_\_.
5. Rapid cycling about the setpoint of an ON/OFF control system is a result of \_\_\_\_\_ riding on the measurement data.
6. List three devices that could not withstand rapid cycling of power.  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
7. Process cycle time is directly affected by the amount of \_\_\_\_\_ in the system.



8. Differential-gap control takes full action when the process variable crosses the \_\_\_\_\_ points.
9. Cycle time will be \_\_\_\_\_ if differential-gap control is used in a system rather than simple ON/OFF control.
10. Rapid cycling is not a problem in differential-gap control if the measurement data plus the \_\_\_\_\_ is less than the differential gap.
11. When the measurement is between the limits, the output will be in the mode determined by which limit was exceeded \_\_\_\_\_.
12. Adding more mass to a Differential-gap control system will \_\_\_\_\_ the amount of overshoot and \_\_\_\_\_ the cycle time of the process.



Control Challenge: Reverse the Scenario

Reverse the positions of the output devices in Figure 5.2. Replace the heater in figure 5.2b with the brushless fan and place the heater across the +9V supply as shown in Figure 5.2c. Control the temperature by turning the fan on and off while the heater is on all of the time. You will have to change the code to such that the output turns on to drive temperature down and off as temperature falls below the setpoint. You may want to place the heater and sensor closer together and open the end of the canister to be able to blow air directly onto the pair. This will change the dynamics of the system. It may cycle faster, perhaps overshoot less, and be more susceptible to chatter. Experiment with it and note your observations. Realize that every system that you work with will have its own unique dynamic characteristics.

## Experiment #5: Closed-Loop Control

---



## Experiment #6: Proportional-Integral- Derivative Control

PID is an acronym for Proportional-Integral-Derivative Control. In this section, we will explore each of these methods and how they work together to efficiently control a system.

### Overview of PID Control

One objective of a process control is to hold a system constant. In the previous exercise, we used various means of cycling the heater of our incubator to maintain a desired temperature. Using differential gap, we created an allowable band in which the heater would cycle, causing the temperature to cycle above and below the setpoint. In Experiment #4, we saw how we could use pulse-width modulation (PWM) to add energy to our system in duty cycles between 0% (fully off) and 100% (fully on). While these types of control had their advantages and disadvantages, PID control allows the greatest control of a system but can be more difficult to implement and tune (or adjust) for optimum performance.

6

Holding a process constant involves continually adding energy that equals the system's losses exactly. If the system's losses were constant, the process control would be as simple as applying one steady state level of drive. However, the factors that affect a process do change. They change in unpredictable magnitudes and at unpredictable rates. Compounding this problem is that a system has reaction delays that must be understood. An instant change in losses due to a disturbance is not felt immediately, and the change in drive to a system is not either. Process control can be as much an art form as it is a science.

The first step to understanding PID control is that every system has both gains and losses of energy.

$$E_{\text{system}} = E_{\text{in}} - E_{\text{out}}$$

A system is said to be in equilibrium when the energy gained equals the energy lost.

$$\text{Equilibrium: } E_{\text{in}} = E_{\text{out}}$$

When in equilibrium our incubator would maintain a constant temperature. But this is seldom, if ever, the case. Depending on the heater drive and conditions surrounding the incubator, temperature will either be increasing or decreasing. As conditions change, such as room temperature changing, air movement changes on the canister, sunlight falling on the canister, or the resistor aging, the amount of heat added and removed is seldom constant.

## Experiment #6: Proportional – Integral – Derivative Control

---

Consider other examples of systems, such as an oil-flow system. The drive element, the pump, is controlled to maintain a desired oil flowrate. A sudden change in the system may be a valve being shut blocking one path for oil flow. Slow change in the system may be corrosion of the piping causing friction or the changing of the oil temperature. The pump needs to adjust in order to compensate for these losses.

One other system to consider is an automobile. Typically we want the car to maintain a constant speed on the highway. The engine makes up for friction losses from the tires on the pavement and wind losses. When the car is maintaining a constant speed, the system is in equilibrium and our foot keeps the accelerator in constant position. When conditions change, such as the car climbing a hill, the cars velocity changes and it begins to slow. The force of gravity increases on the car and these increased losses remove more energy than the engine is supplying and the car begins to slow. Without depressing the accelerator more will the car eventually come to a stop on the hill? No, it will slow to a lower constant speed where once again losses = gains and a new equilibrium is reached.

Conditions, or disturbances on our system, can change very rapidly, such as when a gust of air suddenly blows over the incubator or very slowly such as the heating element aging. PID control can measure and take action on:

- 1) How far from the setpoint a system is, or the magnitude of the error.
- 2) The duration that an error remains.
- 3) How quickly an error occurs in the system, or the rate of change.

The sums of these three evaluations comprise the output drive in an attempt to maintain a system in equilibrium. Figure 6.1 illustrates the evaluation and control of a system for PID control.

The classic PID formula for calculating the controller output is as follows:

$$Co_{pid} = (Kp * E) + (Ki * \int Et) + (Kd * \frac{\Delta E}{\Delta t})$$

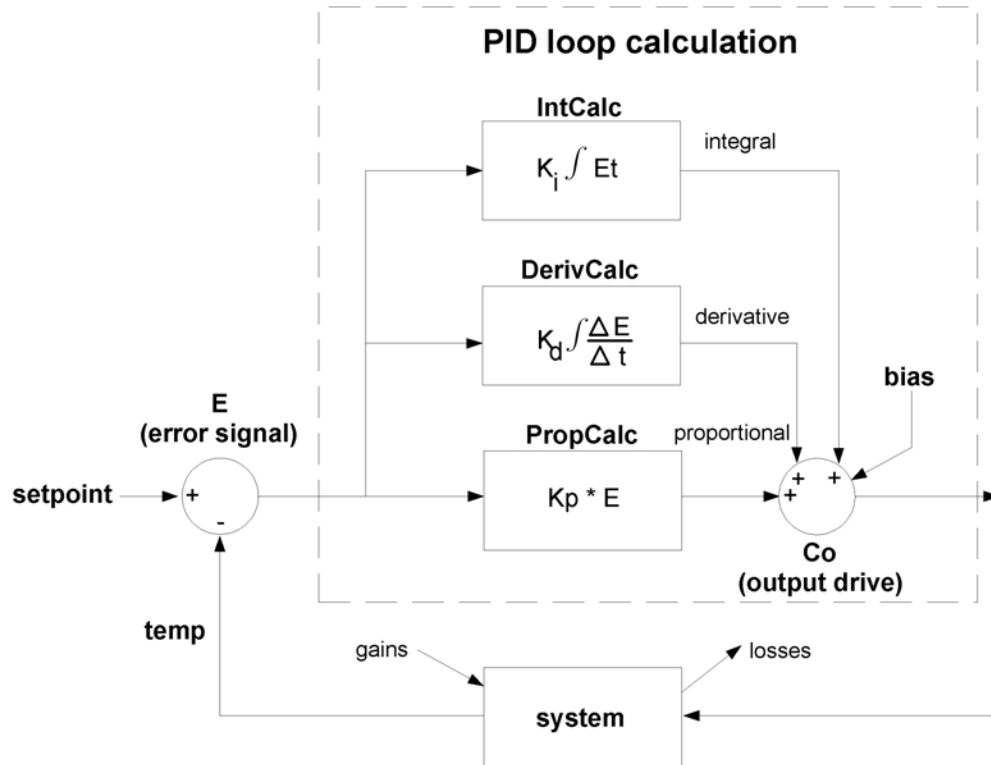
Where:

Co = controller output or drive  
Kp = proportional gain  
E = error signal  
Ki = integral gain  
Kd = derivative gain

The drive equation based on the above is:

$$\%Drive_{Total} = Drive_{PROP} + Drive_{INT} + Drive_{DERIV}$$

Figure 6.1: PID Control Block Diagram



**6**

In this section, the incubator will be controlled using PID control and the PID equation will be explored and illustrated.

Circuit Construction

We will use the same circuit from Exercise #5 (Figure.), but you will manually connect the fan to Pin 19 Vin power or regulated 5V when needed for a disturbance.

The following is the full program for this section. We will change values in the PID Control Settings in testing the different areas of control.

## Experiment #6: Proportional – Integral – Derivative Control

```
'Program 6.1: PID Control with the StampPlot Interface

'***** PID CONTROL SETTING *****
SP      CON      990      ' Initialize setpoint to YOUR bias Temp in TENTHS
Range  CON      20      ' Allowable temperature range in TENTHS (20=2F)
B       CON      50      ' Bias drive setting
Kp      CON      0       ' Proportional Gain Setting in TENTHS (10=Gain of 1)
Ki      CON      0       ' Integral gain constant in TENTHS (1=Gain of .001)
Ti      CON      24      ' Interl Reset time (1=~5 seconds)
Kd      CON      0       ' Derivative gain constant
MinA    CON      75      ' Minimum analog Y axis value
MaxA    CON      120     ' Maximum analog X axis value
MaxT    CON      600     ' Maximum time in seconds X Axis
'*****

'***** Configure Plot
PAUSE 2000
DEBUG  "!TITL PID Control",CR      ' Title Plot
      DEBUG  "!RSET",CR           ' Reset Plot
DEBUG  "!PNTS 1000",CR           ' 1000 data points
DEBUG  "!TMAX ",DEC MaxT,CR      ' Set maximum time
DEBUG  "!AMAX ",DEC MaxA,CR      ' Set analog max
DEBUG  "!AMIN ",DEC MinA,CR      ' Set analog min
DEBUG  "!AMUL .1",CR            ' Analog multiplier of .1
DEBUG  "!TSMP ON",CR            ' Enable time-stamping
DEBUG  "!SAVM ON",CR            ' Save message to file
DEBUG  "!CLMM",CR               ' Clear min/max on reset
DEBUG  "!SHFT ON",CR            ' Enable plot shifts
DEBUG  "!PLOT ON",CR            ' Enable plotting
      DEBUG  "Display drive settings
DEBUG  "!USRS SP=",dec SP," Kp=",dec Kp," Ki=",dec Ki," Ti=",dec Ti," Kd=",dec Kd,CR
DEBUG  "!RSET",CR               'Reset Plot

' ***** Define constants & variables
CS      CON      3       ' 0831 chip select active low from BS2 (P3)
CLK     CON      4       ' Clock pulse from BS2 (P4) to 0831
Dout    CON      5       ' Serial data output from 0831 to BS2 (P5)
Heater  CON      8       ' Output pin to heater
Datain  VAR      BYTE    ' Incoming Data (0 to 255)
Temp    VAR      WORD    ' Hold the converted value representing temp
TempSpan CON      5000   ' Full Scale input span in tenths of degrees.
Offset  CON      700     ' Minimum temp. Offset, ADC = 0
Sign    VAR      WORD    ' Used to hold sign for calculations

Drive   VAR      WORD    ' Amount of total drive
Err     VAR      WORD    ' Amount of error present
P       VAR      WORD    ' Amount of Proportional drive
I       VAR      WORD    ' Amount of Integral Drive
D       VAR      WORD    ' Amount of Derivative Drive

PWMCnt  VAR      BYTE    ' Counter for amount of time to apply PWM
```

## Experiment #6: Proportional – Integral – Derivative Control

```
LastErr      VAR      WORD      ' Holds last temperature for derivative drive
LastErr = 0

IntCount     VAR      BYTE      ' Variable for counting cycles for integral drive
PWMTime     CON      20          ' Variable defining how long PWM drive should last
                                         ' V (20~5 seconds)

Ei           VAR      WORD      ' Cumulative error for integral calculations
Ei = 0       ' Clear cumulative error

'***** Main loop
Main:
  GOSUB Getdata
  GOSUB Calc Temp
  GOSUB Calc Drive
  GOSUB Plot Data
  GOSUB Drive Heater
  GOTO Main

Getdata:           'Acquire conversion from 0831
  LOW CS           'Select the chip
  LOW CLK          'Ready the clock line.
  SHIFTFIN Dout, CLK, msbpost,[Datain\9]      'Shift in data
  HIGH CS          'conversion
  RETURN

Calc Temp:         'Convert digital value to
  Temp = TempSpan/255 * Datain/10 + Offset      'temp based on Span &
  RETURN          'Offset variables.

Calc_Drive:
  GOSUB ErrorCalc      'Error Calcs
  GOSUB PropCalc       'Perform proportional error calcs
  GOSUB IntCalc        'Perform Integral Calcs
  GOSUB DerivCalc      'Perform Derivative calcs
  Drive = (B + P + I + D) 'calculate total drive
  Sign = Drive         'Sign adjust to max of 100 min 0
  GOSUB SetSign
  Drive = ABS Drive MAX 100
  IF Sign = 1 THEN DriveDone
  Drive = 0
  DriveDone:
  RETURN

'***** Drive the heater
Drive Heater:
  FOR PWMCount = 1 TO PWMTime 'Apply pwm at 220 mSec for each PWMTime repetition
    PWM Heater,drive * 255/100,220
  NEXT
  RETURN

'***** Plot Data
Plot_Data:
```

6

## Experiment #6: Proportional – Integral – Derivative Control

---

```
DEBUG DEC Temp,CR

'*** Nicely formatted message output for reading (4 lines)
DEBUG "Set:", DEC SP," Temp:", DEC Temp
DEBUG " %Err:",SDEC Err," %B=", DEC B, " %P=", SDEC P
DEBUG " %I=",SDEC I," %D=", SDEC D
DEBUG " %Drive:",SDEC Drive, CR
'*** Comma-separated message output for import into spreadsheet
' DEBUG ",",DEC Temp,",",SDEC Err,",",SDEC P,",",SDEC I,",",SDEC D,",",SDEC Drive,CR
RETURN

'***** Calculate %Error - Sign adjusted
ErrorCalc:
    Err = (SP - Temp)                'Calculate temperature error
    Sign = Err
    GOSUB SetSign
    Err = ABS Err*100/Range          'Calculate % error
    Err = Err * Sign
    RETURN

'***** Proportional Drive - Sign adjusted
PropCalc:
    Sign = Err
    GOSUB SetSign
    P = ABS Err * Kp + 5/10          'Prop err = %Err * Kp /10 to scale, +5 to round
    P = P * Sign
    RETURN

'***** Integral Drive - Sign Adjusted
IntCalc:
    Ei = Ei + Err                    'Accumulate %err each time
    IntCount = IntCount + 1          'Add to counter for reset time
    IF IntCount < Ti Then IntDone    'Not at reset count? -- done
    Sign = Ei
    Gosub SetSign
    Ei = ABS Ei / Ti                 'Find average error over time
    Ei = Ei * Ki + 5 /10             'Int err = int. err * Ki
    Ei = Ei * Sign
    I = I + Ei                       'Add error to total int. error
    Sign = I
    GOSUB SetSign
    I = ABS I MAX 100                'Limit to 100-prevent windup
    I = I * Sign
    IntCount = 0                     'Reset int. counter and accumulator
    Ei = 0
IntDone:
    RETURN

'***** DERIVATIVE DRIVE
DerivCalc:
    D = (Err-LastErr) * KD           ' Calculate amount of derivative drive
                                     ' based on the difference of last error
DerivDone
```



```
LastErr = Err          ' Store current error for next deriv calc

RETURN

'***** Set sign of value
SetSign:
  IF Sign.bit15 = 0 THEN SignPos      'If signbit is 1, then negative
  Sign = -1
  Return
SignPos:
  Sign = 1
SignDone:
  Return
```

Figure 6.2: Main Process Flow

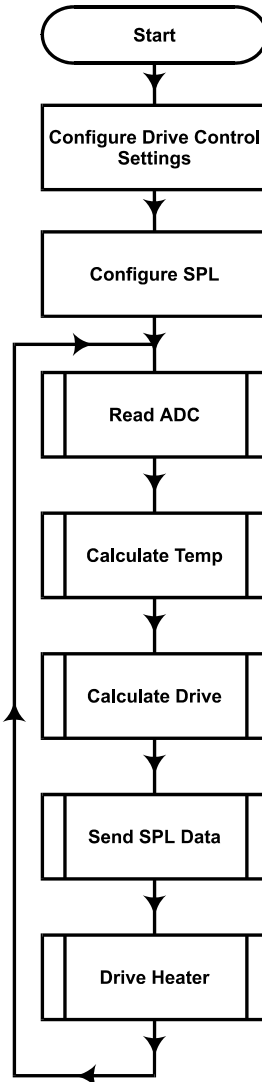


Figure 6.2 is a flowchart of the main loop for the PID program. Specifics of each of the processes will be discussed as they arise.

All microcontrollers have their limitations, as do other systems, such as Programmable Logic Controllers (PLCs). In programming complex operations such as PID, it is important to understand the limitations and finding alternative means.

We've been dealing with the restriction of integer values, such as temperature being in tenths of degrees. One other limitation we'll deal need to deal with is that of negative numbers. While the BASIC Stamp can use negative values, it cannot divide them or use the MIN and MAX instructions to set limits on their size. In this section both of these will be important. The values of drive for PID will be negative or positive depending if drive should be added to the total or subtracted. We will also need to limit the maximum values so that we do not exceed 100% in certain circumstances, such as total drive.

To perform these tasks, a routine called `setSign` is used. Several routines call it using a GOSUB. The possibly negative value to be manipulated is saved to a word variable `sign`. When `setSign` is called, the sign bit (bit 15) is examined. If the sign bit is 1, it is a negative value and the variable `sign` is set to -1. If the sign bit is 0, it is positive and `sign` is set to positive 1. Back in our calling routine, the absolute value of our possibly negative number is manipulated. The result is then multiplied by `sign` to return the value back to positive or negative. The range of signed values can be from -32,768 to +32767.

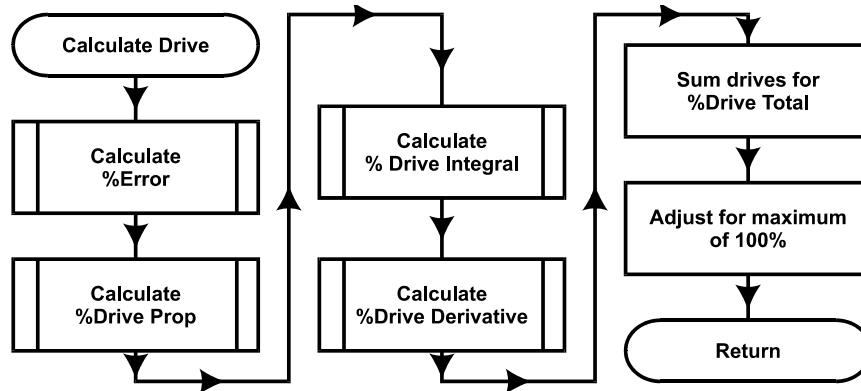
6

```

PropCalc:
  Sign = Err
  GOSUB SetSign
  P = ABS Err * Kp + 5/10          'Prop err = %Err * Kp /10 to scale, +5
to round
  P = P * Sign
  RETURN
...
SetSign:
  IF Sign.bit15 = 0 THEN SignPos  'If signbit is 1, then negative
  Sign = -1
  Return
  SignPos:
  Sign = 1
  SignDone:
  Return
    
```

The first part of program to consider is the total drive calculations. Figure 6.3 is a flowchart for these routines. As discussed, the total drive is the sum of 3 different evaluations based on the error. The total drive % will then be applied to the heater using PWM for 5 seconds. This allows a long on-time compared to a relatively short off-time when performing other operations.

Figure 6.3: Flowchart of Drive Routine



```

Drive Heater:
  FOR PWMCount = 1 TO PWMTime      'Apply pwm at 220 mSec for each PWMTime repetition
    PWM Heater,drive * 255/100,220
  NEXT
RETURN
  
```

```

Calc Drive:
  GOSUB ErrorCalc      'Error calcs
  GOSUB PropCalc      'Perform proportional error calcs
  GOSUB IntCalc      'Perform Integral Calcs
  GOSUB DerivCalc      'Perform Derivative calcs

  Drive = (B + P + I + D)      'calculate total drive
  Sign = Drive      'Sign adjust to max of 100 min 0
  GOSUB SetSign
  Drive = ABS Drive MAX 100
  IF Sign = 1 THEN DriveDone
  Drive = 0
DriveDone:
RETURN
  
```



## Exercises

### Exercise #1: Bias Drive

As discussed, a system in equilibrium is where the energy gains in a system equal the energy losses. When a system is designed typically the engineers will have determined anticipated average losses on the system. The Bias Drive is the drive needed to compensate for average losses. By designing the system so that a 50% drive is sufficient for average losses, provides the ability to add or subtract up to 50% due to other losses or large disturbances to the system. This affords the maximum amount of control.

Modifying our equation slightly:

$$C_{o_{pid}} = B + (K_p * E) + (K_i * \int Et) + (K_d * \frac{\Delta E}{\Delta t})$$

B = Bias Drive

Or:

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{INT} + \%Drive_{DERIV}$$

For the eggs in our incubator to hatch healthy chicks, 50% drive on our heater would provide sufficient energy to maintain temperature in the incubator near 101.5F with average losses. Unfortunately, our system is not well engineered. Being a non-insulated plastic canister with a small resistor for a heater, chances are that 50% drive will not be sufficient under most circumstances for our desired setpoint. Some limitations on our experimental incubator are:

- A bias of 50% PWM drive is insufficient to provide a temperature of 101.5F.
- Every incubator will have a different stable temperature for 50% bias drive due to many factors.
- Our incubator is a fragile, steady state system. Factors such as room temperature and vents or fans blowing on the canister will shift the temperature.
- Moving or bumping the incubator will cause air mixing and affect the measured temperature.

6

## Experiment #6: Proportional – Integral – Derivative Control

---

We will deviate from our optimum incubator setpoint in order to operate the system around a 50% bias temperature. Note that this bias temperature may fluctuate due to room conditions. It is recommended that the student read through all of the Chapter 6 material first and then try to perform as many hands-on portions consecutively as possible.

$$Co_{pid} = B$$

$$\%Drive_{TOTAL} = \%Drive_{BIAS}$$

Determining the Bias Temperature:

- 1) De-energize your system and allow it to cool to room temperature.
- 2) Verify that the Drive Control Setting matches the following:

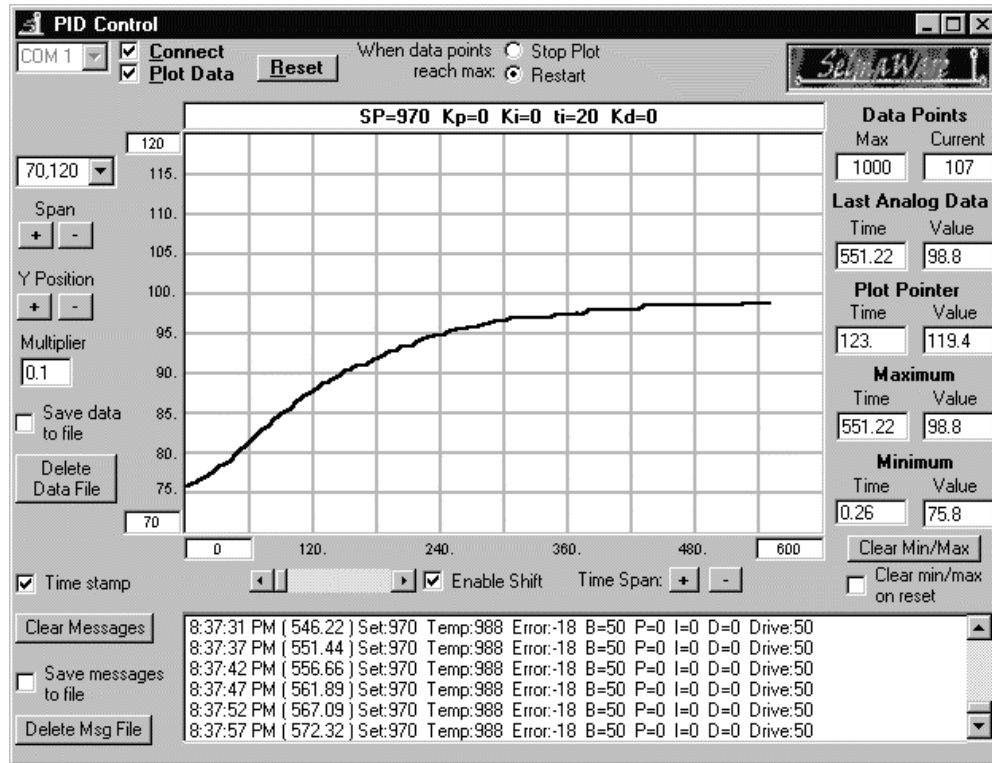
***** PID CONTROL SETTING *****			
SP	CON	990	' Initialize setpoint to YOUR bias Temp in TENTHS
Range	CON	20	' Allowable temperature range in TENTHS (20=2F)
B	CON	50	' Bias drive setting
Kp	CON	0	' Proportional Gain Setting in TENTHS (10=Gain of 1)
Ki	CON	0	' Integral gain constant in TENTHS (1=Gain of .001)
Ti	CON	24	' Integral Reset time (1=~5 seconds)
Kd	CON	0	' Derivative gain constant
MinA	CON	75	' Minimum analog Y axis value
MaxA	CON	120	' Maximum analog X axis value
MaxT	CON	600	' Maximum time in seconds X Axis

Note that Kp, Ki and Kd are all 0 providing 0 drive from these evaluations allowing only bias drive of 50% to the heater.

- 3) Energize your system and download the program to the BASIC Stamp 2.
- 4) Close the debug window and connect on StampPlot Lite.
- 5) Reset your BASIC Stamp 2 by pressing "reset" on the Board of Education.
- 6) Allow the incubator temperature to stabilize to find the 50% bias temperature.
- 7) Round your stabilized temperature to the nearest whole degree and set it as your SP (setpoint) constant (98.8 = 990).

Figure 6.4 is the plot of our test incubator stabilizing.

Figure 6.4: Bias Temperature at 50% Drive



6

Note the amount of time it took the temperature to stabilize. In our test, approximately 450 seconds. In a system with a first-order response, such as our incubator, the system requires 5 time-constants (5TCs) to stabilize (or reach 99% of the new value) following a step-change in conditions, in this case going from 0% to 50% drive. 1 time-constant (1TC) for our system would be 450/5 or 90 seconds. The response time of the system is important in tuning a system as you will see later.

Record for your system: 50% Bias Temperature: \_\_\_\_\_  
 Time for 5TCs: \_\_\_\_\_  
 Time for 1TC: \_\_\_\_\_

## Experiment #6: Proportional – Integral – Derivative Control

---

### Exercise #2: Proportional Control

$$Co_{pid} = B + (Kp * E)$$
$$\%Drive_{Total} = \%Drive_{BIAS} + \% Drive_{PROP}$$

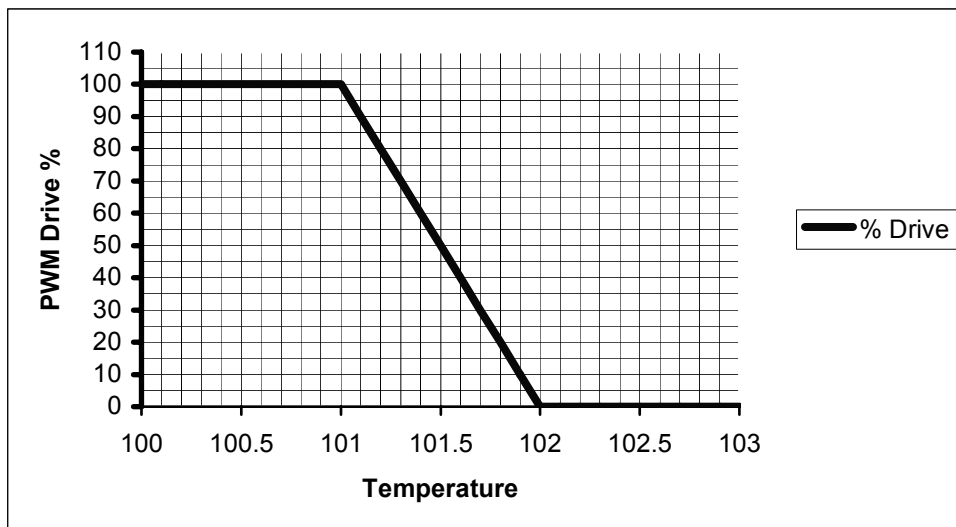
The next evaluation in the PID equation is proportional control of the system. The amount of drive from proportional control is a direct relationship to how much error exists in the system. The greater the error the greater the amount of proportional drive.

Let's say our setpoint is 101.5F, and a bias of 50% PWM is sufficient to maintain this temperature. If the temperature drops to 101.0F, giving a -0.5F error, do we want to drive at 100% PWM, or something lower, to bring the temperature back to the setpoint?

For our incubator example, we want to maintain a temperature of 101.5F. The steadier we maintain this temperature, the healthier our eggs will be. It is allowable, though, to go 0.5F above or below this setpoint. So our setpoint will be 101.5 with an allowable band of +/- 0.5F, or 101.0F to 102.0F degrees.

If any error correction reaches the upper or lower limits, we want to take full action to return temperature back to the setpoint. Any error between the setpoint and the limits will provide a proportional amount of drive action. Figure 6.5 is a graphical representation of the incubator temperature verses the amount of drive needed for the given band.

Figure 6.5: Temperature vs. Drive





From Figure 6.5, if the temperature is at 101.5F, 50% drive will be used to add heat energy to our system. If the temperature drops to 101 F, 100% drive will be used to increase the temperature, and any temperature in between will result in a proportional amount of drive.

If the temperature rises above 101.5F, the drive will proportionally decrease to lower the temperature until, at 102F, %drive has decreased to 0%.

From the graph, what would the drive be at 101.2 degrees? \_\_\_\_\_; at 101.7 degrees? \_\_\_\_\_.

Now let's look at it mathematically. The error of our system is calculated by subtracting the current temperature from the setpoint:

$$\text{Error} = \text{Setpoint} - \text{Actual}$$

The percent error is found by dividing the error value by the full temperature range and multiplying by 100%:

$$\% \text{Error} = \frac{\text{Error}}{\text{FullRange}} \times 100\%$$

For our incubator the full temperature range is:

$$102.0 - 101.0 = 1.0\text{F.}$$

If the temperature is 101.2:

$$\begin{aligned} \text{Error} &= 101.5 - 101.2 = .3 \\ \% \text{Error (E)} &= .3 / 1.0 * 100\% = 30\% \end{aligned}$$

The general gain formula is:

$$\text{Gain} = \frac{\Delta \text{Output}}{\Delta \text{Input}}$$

For our example, we are changing 100% of the drive over 100% of the allowable temperature range:

$$\text{Gain}(K_p) = \frac{\Delta \text{Output}}{\Delta \text{Input}} = \frac{100\%}{100\%} = 1$$

Our system would have a proportional gain (Kp) of 1.

## Experiment #6: Proportional – Integral – Derivative Control

---

The amount of proportional drive at 101.2 degrees would be:

$$\%Drive_{PROP} = K_p * E = 1 * 30\% = 30\%$$

The total drive of our system would then be:

$$\%Drive_{TOTAL} = B + (K_p * E) = 50\% + 30\% = 80\%.$$

Does this values match what you read from the Figure 6.3 for a temperature of 101.2F?

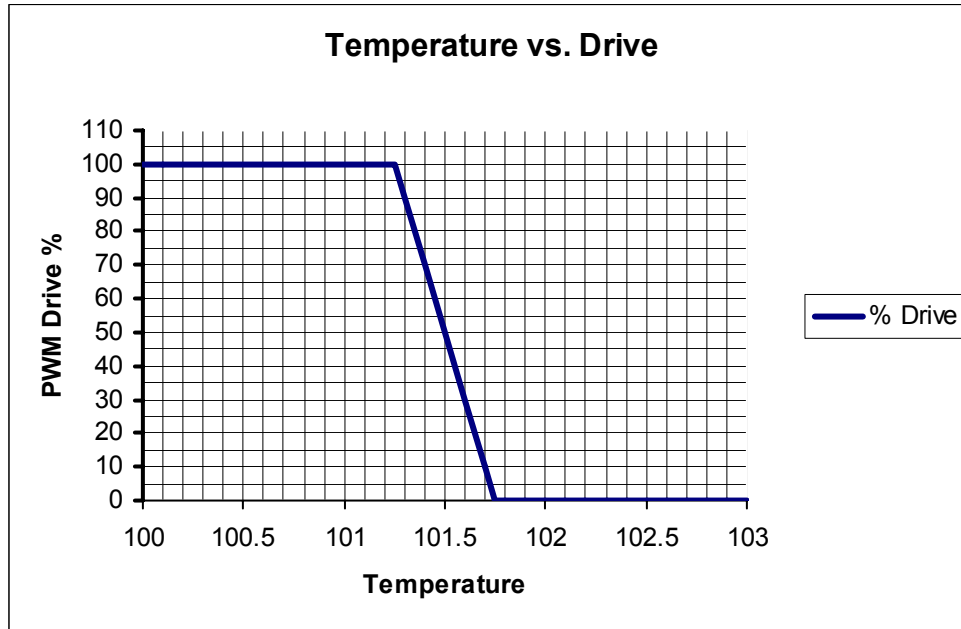
Calculate the total drive for a temperature of 101.7: \_\_\_\_\_

Compare your value to Figure 6.3. Do they match? \_\_\_\_\_.

One more term to consider is Proportional Band. Figure 6.3 is termed a 100% Proportional Band because the full range of drive covers 100% of our allowable band (101.0- 102.0). Does this mean the temperature will not exceed our high and low limits? No. This simply means that at those limits our system will be taking full action to get the temperature back to the setpoint.

What if we wanted tighter temperature control of our system? Our allowable band for healthy eggs is still 101.0 to 102.0, but we can adjust our values to take full control over half the allowable range. Figure 6.6 is a plot of this control.

Figure 6.6: Temperature vs. Drive Over 50% of Range



**6**

In Figure 6.6 the system is driving twice as hard for any deviation from the setpoint. What would be the proportional gain for this system?

$$\text{Gain}(K_p) = \frac{\Delta\text{Output}}{\Delta\text{Input}} = \frac{100\%}{50\%} = 2$$

Note from the equation the system is driving the full range of output over only 50% of the allowable temperature range. This is also known as a 50% Proportional Band because full control action happens over 50% of the allowable range.

Use the equations to calculate the following at 101.7F:

Error = \_\_\_\_\_

%Error = \_\_\_\_\_ (Hint: The allowable band is still 1.0)

Proportional Drive = \_\_\_\_\_

Total Drive = \_\_\_\_\_.

Note the relationship between proportional gain and proportional band:

Experiment #6: Proportional – Integral – Derivative Control

---

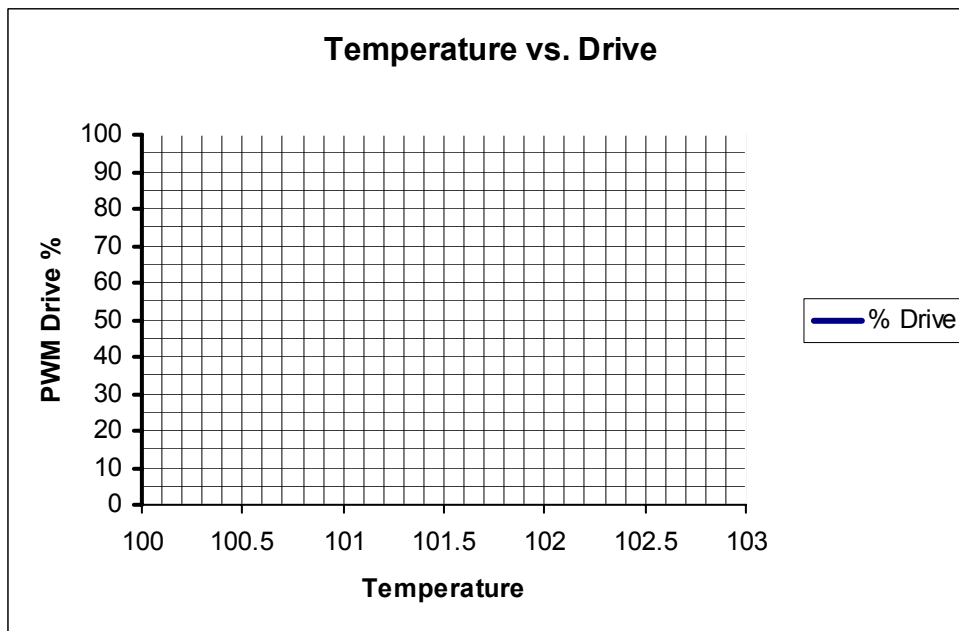
$$\text{Proportional Band} = \frac{1}{\text{Proportional Gain}} * 100\% = \frac{1}{2} * 100\% = 50\%$$

What if we wanted a slower responding system? We can define that the full range of drive will be over temperatures greater than the allowable range. If 100% drive were to cover twice as much as the allowable range:

$$\text{Gain}(K_p) = \frac{\Delta\text{Output}}{\Delta\text{Input}} = \frac{100\%}{200\%} = 0.5$$

Calculate the Proportional band: \_\_\_\_\_

Graph the Drive vs. Temperature for this gain:



Calculate the following for a temperature of 101.7F:

- Error = \_\_\_\_\_
- %Error = \_\_\_\_\_ (Hint: The allowable band is still 1.0)
- Proportional Drive = \_\_\_\_\_
- Total Drive = \_\_\_\_\_.

Controlling the Incubator:

Theory is nice, but now we need to deal with the temperature inside our actual canister using proportional control and some limitations we'll need to work with.

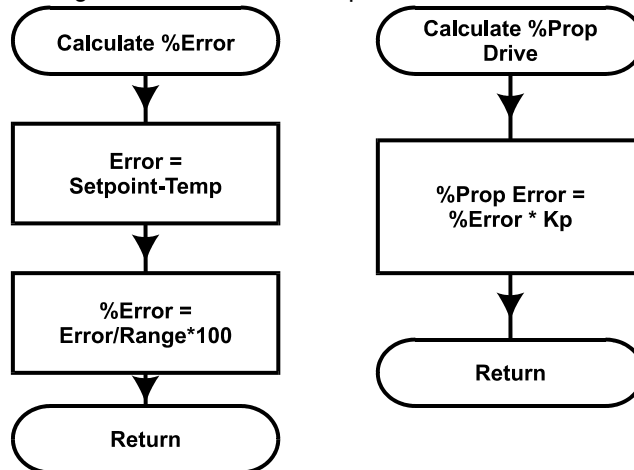
- 1) The bias temperature at 50% bias drive will not be 101.5F. The 50% bias temperature for your incubator was found in Exercise #1 of this experiment. If need be, repeat that portion to find the current bias temperature. This should be done if you think conditions in or around the canister are different. For instance if the room temperature is now higher than it was before, your stable temperature at 50% bias will stabilize at a higher value.
- 2) The BASIC Stamp works in integer math. Our temperatures are in tenths of degree (101.5=1015), and we will want to work with gain settings that are less than 1.
- 3) The resolution of the A/D is 0.19 degrees, which would not provide very fine control with an allowable temperature band of 101.0 – 102.0. For these experiments we will increase the allowable band to +/- 1.0F or 100.5F – 102.5F for a range of 2.0F.

**6**

In performing these experiments, the general sequence will be to allow the incubator to stabilize at the bias temperature. Once stable, the PID program with the appropriate settings will be downloaded. After 1 minute the fan positioned approximately 1 inch from the canister will be energized manually by connecting the positive lead to the Vin supply for 10 seconds.

Figure 6.7 is the flowchart for the error calculations and proportional drive. Notice in the code for the proportional calculations the result is divided by 10 to scale our Kp value down from tenths.

Figure 6.7: Error and Proportional Calculations



```
'***** Calculate %Error - Sign adjusted
ErrorCalc:
  Err = (SP - Temp)           'Calculate temperature error
  Sign = Err
  GOSUB SetSign
  Err = ABS Err*100/Range     'Calculate % error
  Err = Err * Sign
  Return

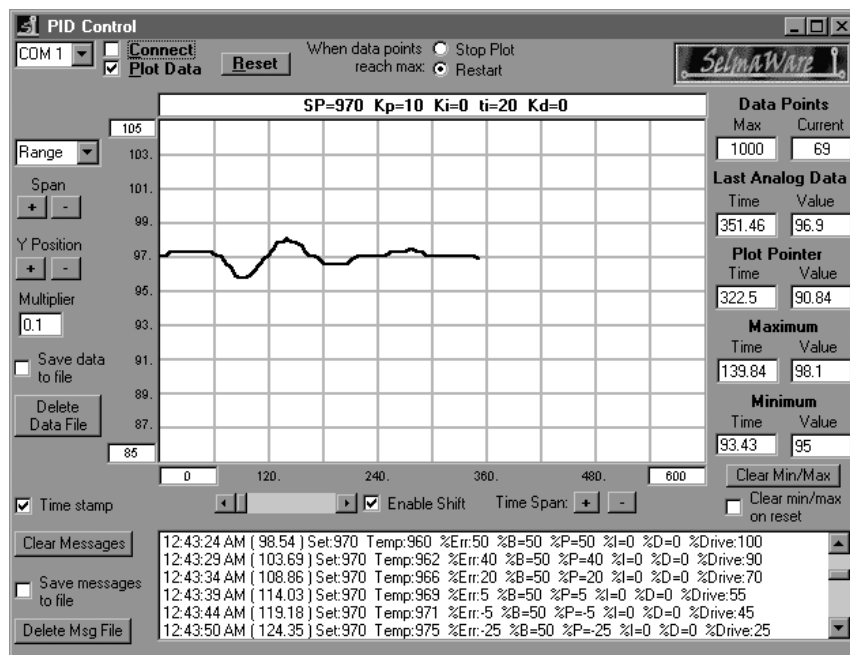
'***** Proportional Drive - Sign adjusted
PropCalc:
  Sign = Err
  GOSUB SetSign
  P = ABS Err * KP/10         'Prop err = %Err * Kp /10 to scale
  P = P * Sign
  RETURN
```

PID Control: 100% Proportional Band, Proportional Gain = 1

- 1) Find the 50% drive bias temperature if needed. Allow the temperature to stabilize at the bias temperature.
- 2) Set the PID Control settings program 6.1 accordingly, setting the value of **SP to your bias temperature in tenths** and the value of **Kp to 10 for a gain of 1**.
- 3) Disconnect on SPL, download the BS2 program, close the debug window, reconnect on SPL and reset your BS2.
- 4) Monitor temperature on SPL. When 1 minute (60 seconds) has passed, energize the fan for 10 seconds.
- 5) Monitor the incubator's response to the disturbance.

6

Figure 6.8a: Response with  $K_p = 1$ , 100% Band



Note how the temperature overshoots the setpoint then turns, goes down, overshoots, and goes up again. This is termed **hunting**. Drive is added and removed based on the response of the system to finally settle at

## Experiment #6: Proportional – Integral – Derivative Control

---

the setpoint value. Depending on the needs of your system, PID may be tuned to prevent hunting. The cruise control in our cars might make for an interesting drive if the speed hunting around the setpoint speed!

The user status box at the top of the plot shows the current settings for PID control. The message window displays the setpoint and current temperatures, %Error, and the amount of drive from each evaluation along with the total drive.

Let's do the math to analyze some of these values. The setpoint is 97.0 or 97.0F, the bias temperature for our testing. At a temperature of 96.2 (96.2F) the total drive was 90% with 50% from the Bias drive and 40% from the proportional drive.

$$\begin{aligned}\text{Error} &= \text{Setpoint} - \text{Temperature} = 97.0\text{F} - 96.2\text{F} = 0.8\text{F} \\ \%\text{Error} &= \text{Error}/\text{Range} * 100 = 0.8\text{F}/2\text{F} * 100 = 40\% \\ \%\text{Drive}_{\text{PROP}} &= K_d * E = 1 * 40\% = 40\% \\ \%\text{Drive}_{\text{INT}} \text{ and } \text{Drive}_{\text{DERIV}} &\text{ will be 0 since their gains are 0.} \\ \%\text{Drive}_{\text{Total}} &= \%\text{Drive}_{\text{BIAS}} + \%\text{Drive}_{\text{PROP}} + \%\text{Drive}_{\text{INT}} + \%\text{Drive}_{\text{DERIV}} = \\ &= 50\% + 40\% + 0\% + 0\% = 90\%\end{aligned}$$

The calculations worked!

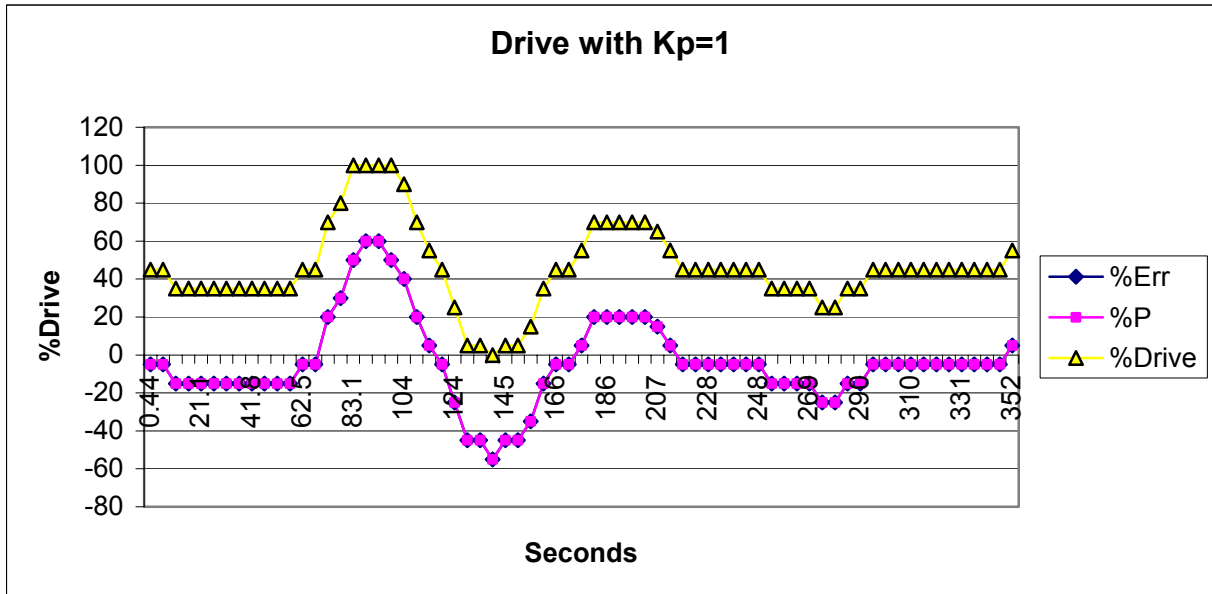
Using the alternative line for the message file logging in the Display routine, the data saved to the message file may be imported into a spreadsheet, such as Excel, for analysis. It is a simple comma-separated version of the message window data suitable for importation and graphing.

```
*** Comma-separated message output for import into spreadsheet
'   DEBUG ",", DEC Temp, ",", SDEC Err, ",", SDEC P, ",", SDEC I, ",", SDEC D, ",", SDEC Drive, CR
```

Figure 6.8b is the message data imported into Excel will show the error, proportional drive and total drive.



Figure 6.8b: Message file plotted in Excel



6

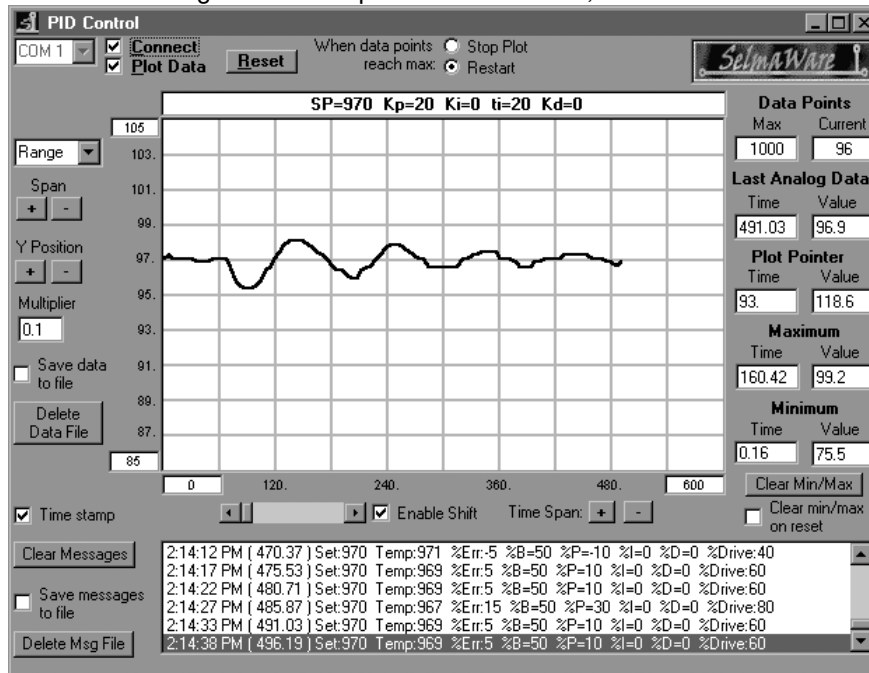
Note that with a proportional gain of 1, the %Error line is barely seen because the %P (Proportional Error) exactly overlaps it.  $\%Drive_{PROP} = \%Error$ . The amount of proportional error is added to the bias drive of 50% and follows the error.

**PID Control: 50% Proportional Band, Proportional Gain = 2**

Repeat the experiment for a proportional band of 50% at a gain setting of 2. Change  $K_p = 20$  in the control settings and run the program.

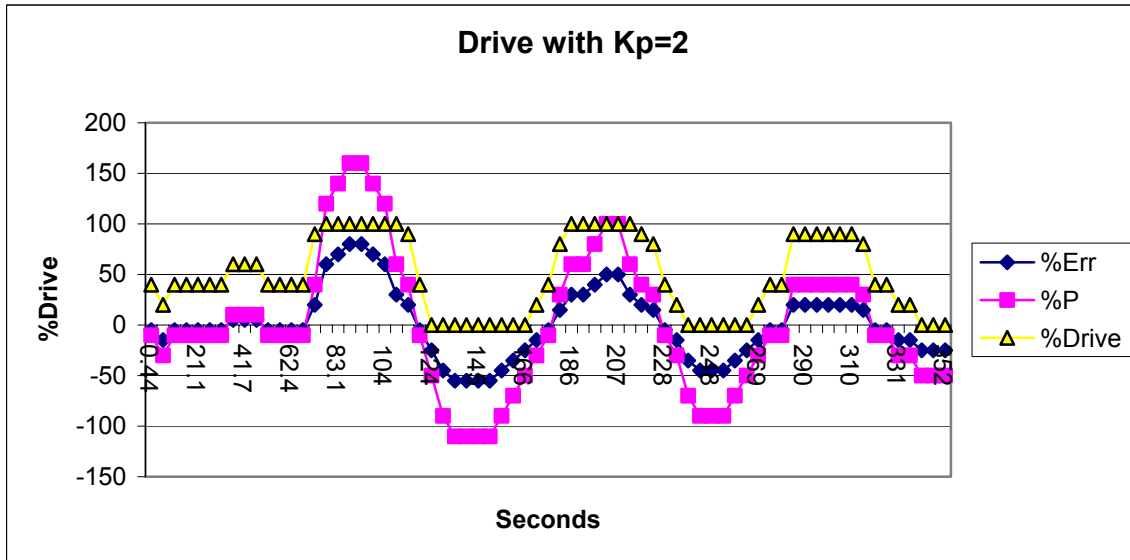
Experiment #6: Proportional – Integral – Derivative Control

Figure 6.9a: Response with Gain = 2, 50% Band



With a gain of 2, note that the response time is slightly faster though there is greater hunting.

Figure 6.9b: Data File Plotted in Excel



6

From Figure 6.9b note that at this gain setting the amount of proportional drive (%P) is twice as much as the error (%Err).

Verify the drive amounts shown in the message window of Figure 6.7a at 96.7F:

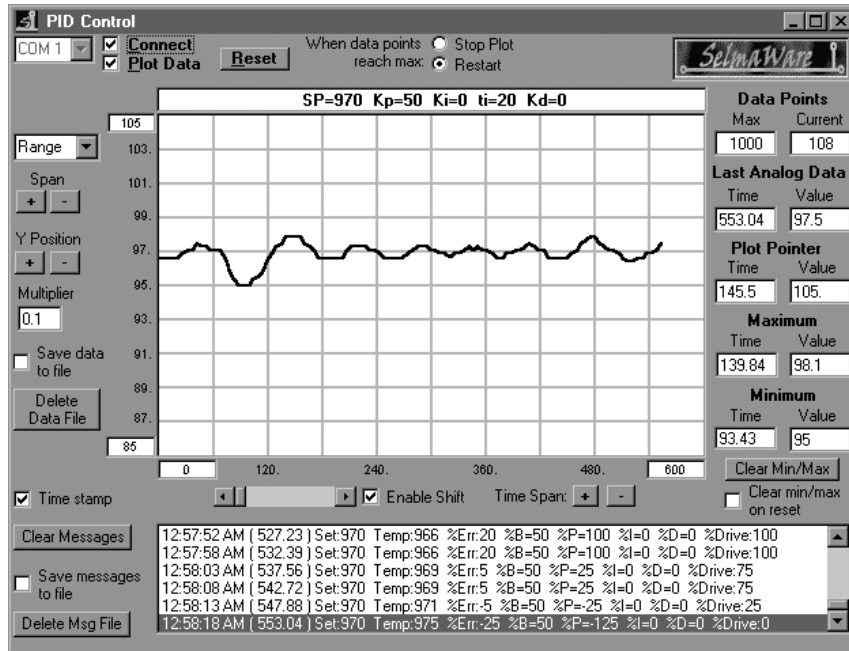
Error = \_\_\_\_\_  
 %Error = \_\_\_\_\_  
 %Drive<sub>PROP</sub> = \_\_\_\_\_  
 %Drive<sub>TOTAL</sub> = \_\_\_\_\_

PID Control: 20% Proportional Band, Proportional Gain = 5

Too much gain can be unsuitable for control. Repeat the experiment for a proportional band of 20% at a gain setting of 5.  $K_p = 50$  in the control settings. Figure 6.10 is our SPL result.

Experiment #6: Proportional – Integral – Derivative Control

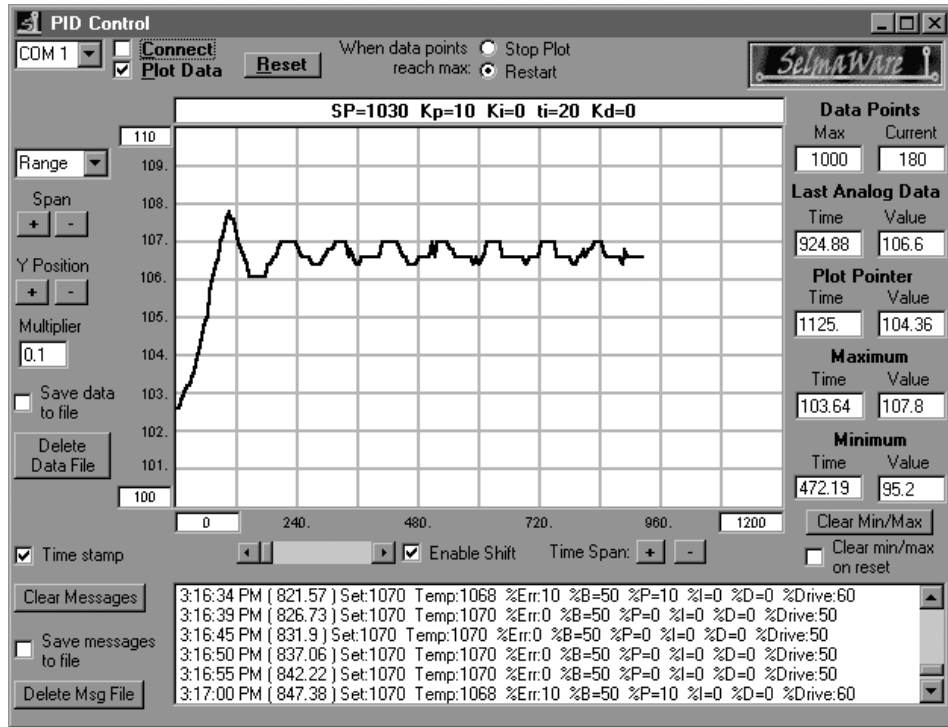
Figure 6.10: Response with Gain = 5, 20% Band



Note that the response time of the system is again slightly faster, but there is much more hunting and continued instability in the system.

As a final experiment, set the setpoint (SP) temperature substantially higher than the bias temperature, but within a controllable temperature range with a proportional gain of 1 ( $K_p=10$ ). We tested at 10F above the bias temperature (107F). Plot the results. Figure 6.11 is the result of our tests.

Figure 6.11: Setpoint 10F (107F) above Bias Temperature



6

While trying awfully hard, the temperature is not stabilizing at 107 degrees. If 107F is an achievable temperature for the system, why doesn't it stabilize there? Remember that for our system 50% drive stabilized around 97F. Additional drive is added because an error exists. If the incubator were able to stabilize at the setpoint, the error would be 0, providing 0% drive from proportional and only bias drive that is insufficient to maintain the temperature creating an error.

$$\begin{aligned} \%Drive_{TOTAL} &= \%Drive_{BIAS} + \%Drive_{PROP} \\ \%Drive_{PROP} &= K_p * E \text{ If } E = 0 \text{ then } \%Drive_{PROP} = 0\%. \\ \%Drive_{TOTAL} &= 50\% + 0\% \end{aligned}$$

If the setpoint temperature is not the bias temperature, some error MUST exist to provide additional drive from the proportional control. The higher the proportional gain, the smaller the remaining error. In the next section, we will see how Integral Error may be used to drive away this remaining error.

## Experiment #6: Proportional – Integral – Derivative Control

---

### Challenge!

1. If the proportional gain were set to .5 ( $K_p=50$ ), what type of response would you expect from the system? Why?
2. If the temperature is 0.6F below your setpoint, what would the total drive be?
3. Confirm your theory.

### Exercise #3: Proportional+Integral Control

$$Co_{pid} = B + (K_p * E) + (K_i * \int Et)$$

$$\%Drive_{Total} = \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{INT}$$

So far we've looked at what occurs when quick disturbances occur to our system in equilibrium. Proportional control may be used to drive the temperature back to the desired setpoint. But what happens when the disturbance affects the equilibrium of our system over a long period of time? At the end of the last experiment it was seen what occurs when the bias drive is not sufficient to make-up for average losses. Because some error must exist for proportional drive, the setpoint temperature cannot be maintained.

Integral control can be used to drive-away error remaining due to long lasting disturbances in the system. These may be from additional losses or gains of energy that remain for a long period of time. Consider our incubator. We found a bias temperature at which a 50% bias drive was sufficient to make up for the losses in the system maintaining it in equilibrium.

But what would happen if the fan were continuously pointed at the incubator? Continuous system losses would be higher. The 50% bias drive will be insufficient to maintain the temperature and proportional drive will respond to the error in an attempt to drive the system back toward to the setpoint. But as we've seen, because some error must remain, the setpoint is not maintained. The system will stabilize at a temperature below the desired setpoint.

Over time, integral drive can be used to drive away this error, allowing the temperature to reach the setpoint.

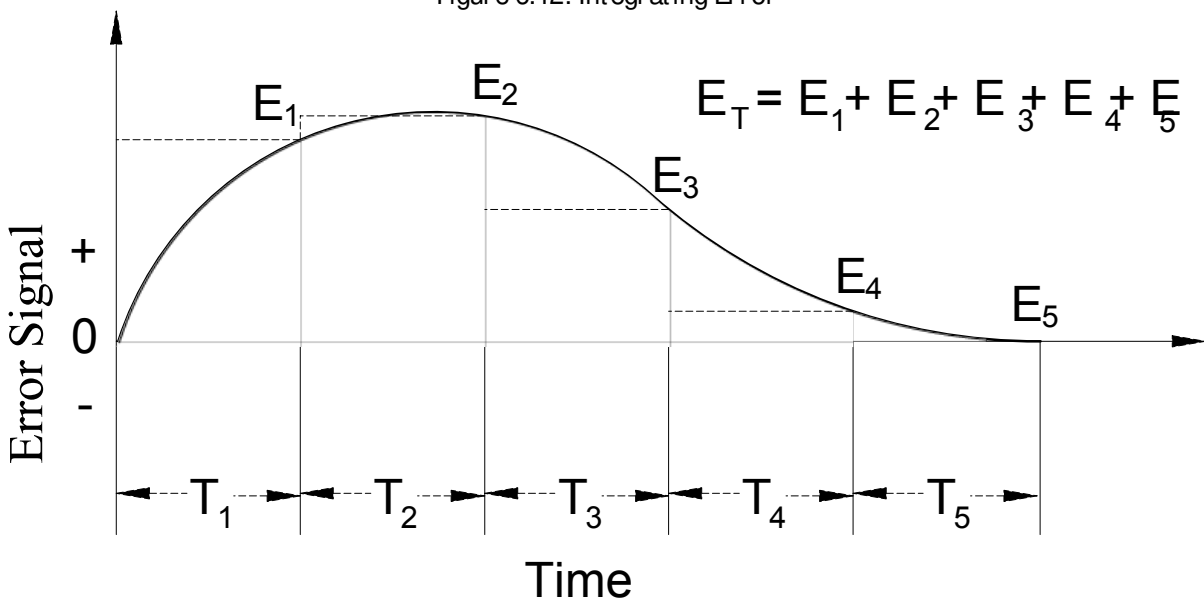
Integral drive is also used when a slow approach with long stabilization times are needed to ensure no overshoot. Consider the example of cooking soup. After cooking a bit, you taste, add an amount of salt you feel appropriate for what you would like the final taste to be. Do you taste immediately and add more? No, you wait a while to allow the salt to blend in, then taste, and add a bit more until you finally reach your desired taste. What if too much salt is added? Cutting back is a bit more difficult!

An industrial example may be that of adding pigment to paint for a desired color. Electronic circuitry monitors paint color and gradually add pigment until the desired color is reached.

In integral drive the amount of error is integrated over time. The larger the error and the longer it lasts, the greater the integral drive will be.

As seen from figure 6.12, the amount of error under the curve is added together to find the integrated error. The longer the error exists, the higher integrated total error ( $E_T$ ) will be.

Figure 6.12: Integrating Error



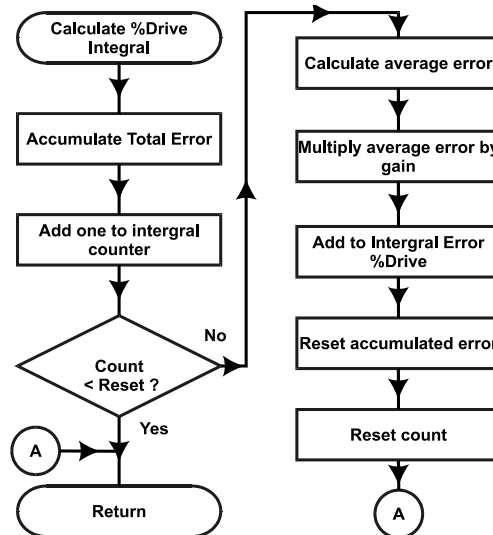
The integrated error is multiplied by the integral gain to find the integral drive.

$$E_T = \Sigma(E_1 + E_2 + E_3 + \dots)$$

$$\%Drive_{INT} = K_I * E_T / T$$

How often should the integral gain be updated or reset? Integral drive should be based on stabilized readings. Depending on the response time of the system this may be anywhere from seconds to hours or even days. Just as with adding salt to the soup, if system hasn't stabilized from the last addition, it would be easy to add too much. The stabilization time of our incubator was found back in Experiment #1 of this section and was 450 seconds for our testing. Figure 6.13 is the flowchart for the integral calculations.

Figure 6.13: Integral Flowchart



Code to accompany chart above:

```

***** Integral Drive - Sign Adjusted
IntCalc:
    Ei = Ei + Err                                'Accumulate %err each time
    IntCount = IntCount + 1                       'Add to counter for reset time
    IF IntCount < Ti Then IntDone                 'Not at reset count? -- done
    Sign = Ei
    Gosub SetSign
    Ei = ABS Ei / Ti                              'Find average error over time
    Ei = Ei * Ki + 5 / 10                         'Int err = int. err * Ki
    Ei = Ei * Sign
    I = I + Ei                                    'Add error to total int. error
    Sign = I
    GOSUB SetSign
    I = ABS I MAX 100                             'Limit to 100-prevent windup
    I = I * Sign
    IntCount = 0                                  'Reset int. counter and accumulator
    Ei = 0
IntDone:
    RETURN
    
```



### Controlling the Incubator

In this exercise, the fan will be used to produce a long lasting disturbance to the system. The fan should be placed approximately 6 inches from the incubator. It will be powered from Vdd (5V – Pin 20 or from the Vdd terminal on top of the breadboard) to provide a 'gentle' cooling to the canister (you may have to 'kick-start' the fan to start it turning). This will produce a long-term disturbance to the system instead of the strong 10 second disturbances used in the proportional testing.

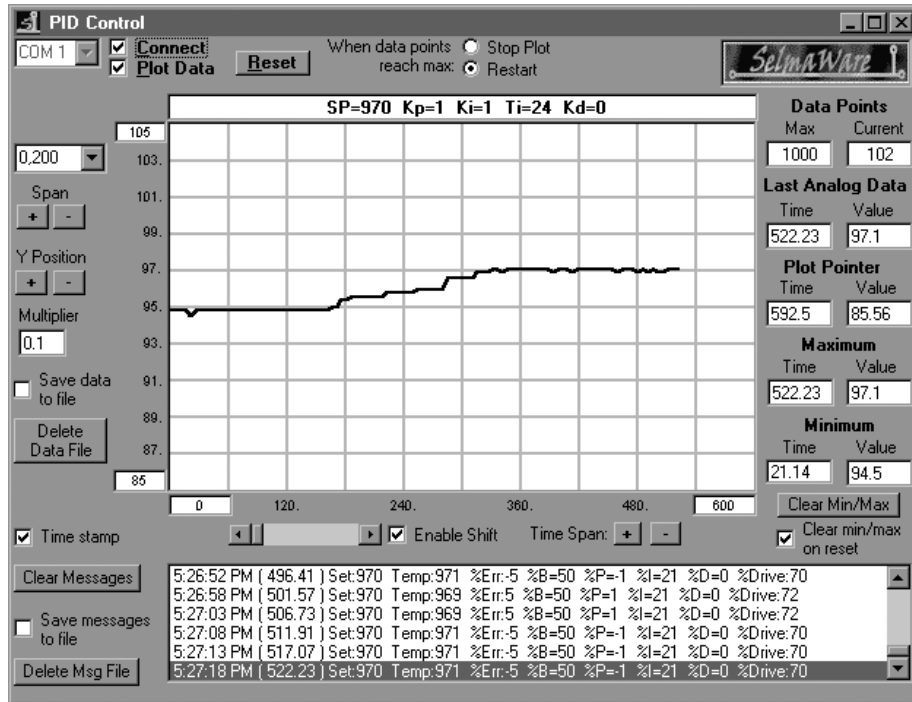
For proportional gain we will use a very small value to prevent hunting and provide a large error from the setpoint. The integral update time, or reset time, of the integral drive will be approximately 120 seconds (450 seconds would be more appropriate to allow full stabilization, but that's a long time to plot!). Integral gain will be set in tenths.

- 1) Setup Program 6.1 for a 1000% Proportional Band, Gain of 0.1 ( $K_p=1$ ) and Integral gain of 0 ( $K_i=0$ ) and derivative of 0 ( $K_d=0$ ).
- 2) Point the fan at the incubator from a distance of about 6 inches (if you see no response after 30 seconds, move it closer in one-inch increments and try again).
- 3) Energize the fan from Pin 20 (5V) and ground. Push-start the fan if needed.
- 4) Allow the system to stabilize with this new system loss.
- 5) Change the integral gain to .1 ( $K_i=1$ ),  $T_i = 24$ .
- 6) Download and plot at the new settings.

6

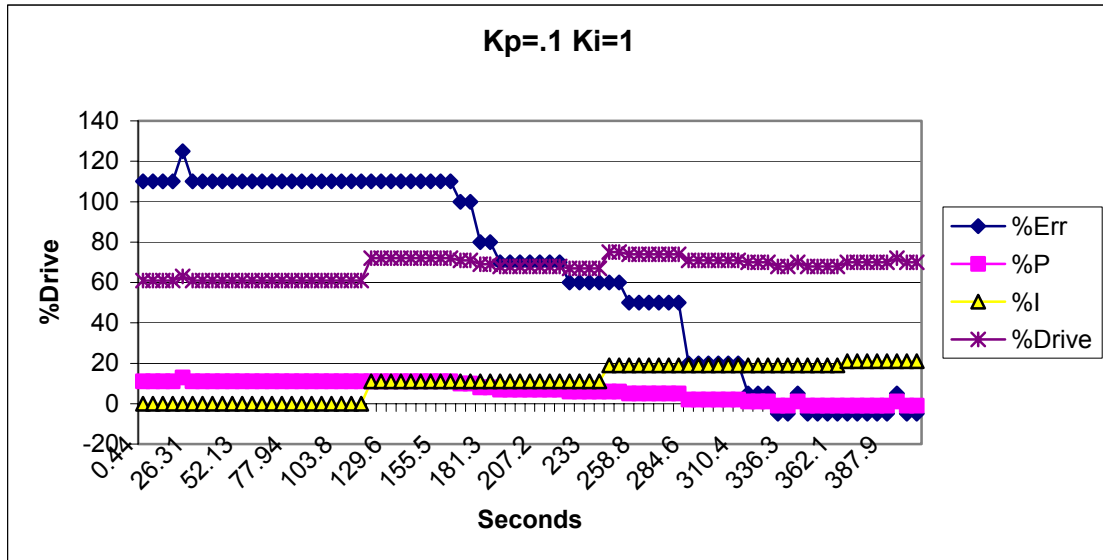
Figure 6.14a shows our results of this test with a bias setpoint of 97.0 F and Figure 6.14b is an Excel plot of captured data.

Figure 6.14a: Long-Term Disturbance Effects



Note that with a bias setpoint of 97.0F and the disturbance of the fan, the initial stable temperature was 94.8F. Over time the drive, and hence the temperature, was slowly bumped up until the actual temperature was at the setpoint.

Figure 6.14b: Data Plotted in Excel



6

Note that the initial error (%Err) was 11% at a stable temperature of 94.8F.

$$\begin{aligned} \%Drive_{Total} &= \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{INT} \\ \%Drive_{PROP} &= K_p * E_T = 0.1 * (97.0F - 94.8F) / 2F * 100 = 11\% \\ \%Drive_{INT} &= K_i * 0 \text{ until the first reset time.} \\ \%Drive_{Total} &= 50\% + 11\% + 0\% \end{aligned}$$

Around 120 seconds, the first integral reset time occurs. All the error samples prior to that time are summed and averaged over time. This is multiplied by the integral gain to find the integral drive.

$$\begin{aligned} \%Drive_{PROP} &= 11\% \text{ still since the temperature is still } 95.8F \\ \%Drive_{PROP} &= K_p * E_T = .1 * (11\% + 11\% + 11\% \dots [24 \text{ of them!}]) / 24 = 11\% \\ \%Drive_{Total} &= 50\% + 11\% + \%11\% = 72\% \end{aligned}$$

Note that with the higher total drive (%Drive), temperature eventually begins to increase, error decreases, and proportional drive decreased. Integral remains constant until the next reset time around 240 seconds when it bumps up based on the average of the errors since the last reset time.

Eventually, temperature returns to the setpoint, the error is driven away, proportional drive is virtually gone and integral drive plus the bias drive are maintaining the temperature.

## Experiment #6: Proportional – Integral – Derivative Control

$$\begin{aligned}\%Drive_{Total} &= \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{INT} \\ \%Drive_{Total} &= 50\% - 1\% + 21\% = 70\%\end{aligned}$$

But what happens when the long lasting disturbance leaves?

- 1) De-energize the fan.

Figure 6.15: Temperature Following the Removal of a Disturbance

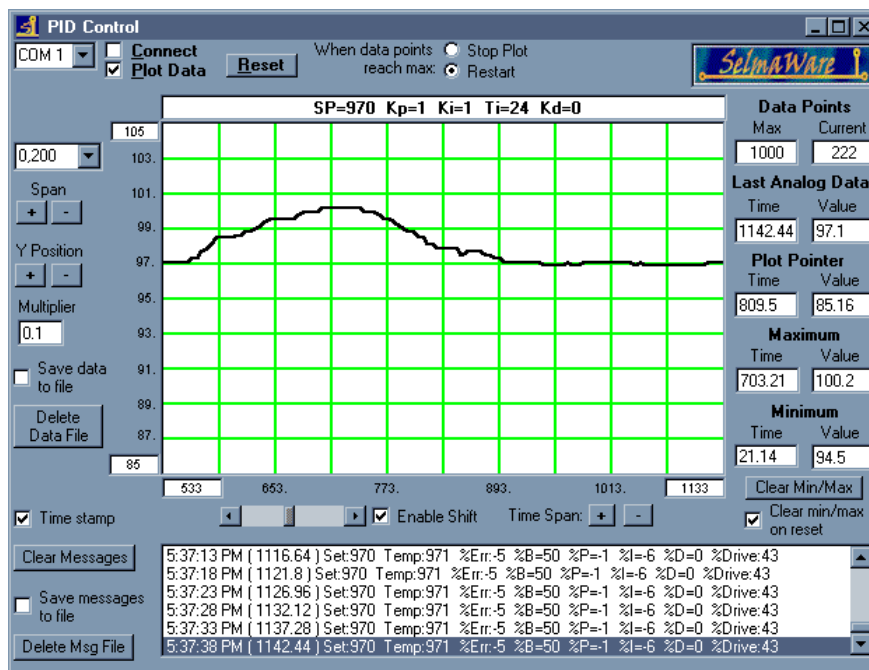


Figure 6.15 shows what happens when the disturbance is removed. The additional drive from integral drive must be slowly integrated away again.

Integral wind-up can occur if the addition of integral drive is insufficient to force the system back to the setpoint. Integral drive would continually be added. This would mean that an error would constantly persist and the output would 'wind-up' to an abnormally high values leading to an unresponsive system. If our program allowed integral drive to wind-up to 20000%, how long would it take to drive it away once a

disturbance is removed providing an error of 10%? It is wise to make a provision in your program to limit the cumulative integral drive to less than 100%

Challenge!

1. What would the system response be if Integral gain were 1 ( $K_i=10$ ) and the reset time were 60 seconds ( $K_i=12$ )? Why?
2. Test and confirm your theory.

Exercise #3: Proportional-Derivative Control

$$Co_{pid} = B + (Kp * E) + (Kd * \frac{\Delta E}{\Delta t})$$

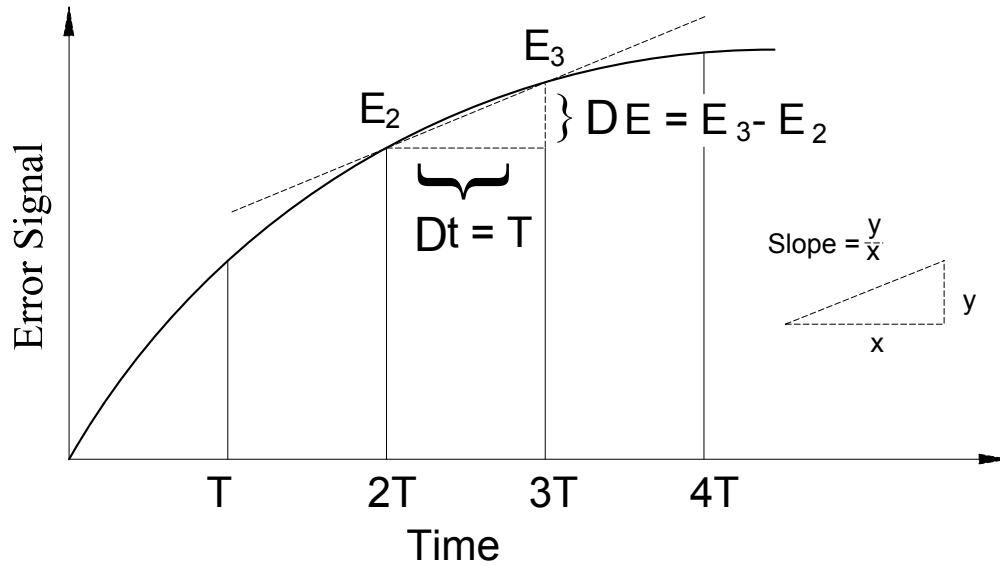
$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{DERIV}$$

6

Derivative control responds to a CHANGE in the error. The fundamental premise determining derivative drive assumes the present rate of change in the error signal will continue into the future unless action is taken. Derivative drive, when properly tuned, allows a system to rapidly respond to sudden changes and react accordingly.

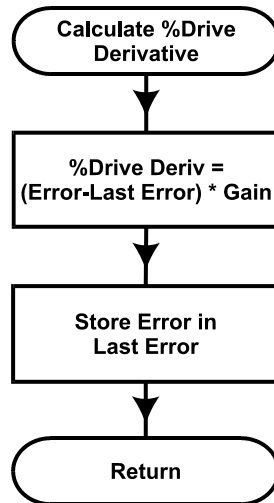
Figure 6.16 illustrates how we would evaluate the slope of an error signal. Finding the difference between error samples taken at regular time intervals reflects the rate at which the process variable is changing. The greater the difference, the greater the slope and therefore the greater the derivative drive necessary to counteract the change.

Figure 6.16: Derivative Error



Consider the act of balancing yourself on a fallen log. Typically you perform slow adjustments to your body position to maintain balance. You are responding proportionally to an error that exists in your equilibrium. Suddenly, a large gust of wind hits you causing you to rapidly lose balance. In response you quickly shift to counteract the wind. This action was based on sudden change in your equilibrium over a very short period of time. In our incubator, a rapidly changing temperature will be counteracted by an immediate reduction in drive.

Figure 6.17: Flowchart and Code for Derivative Control



6

Code for this flowchart:

```

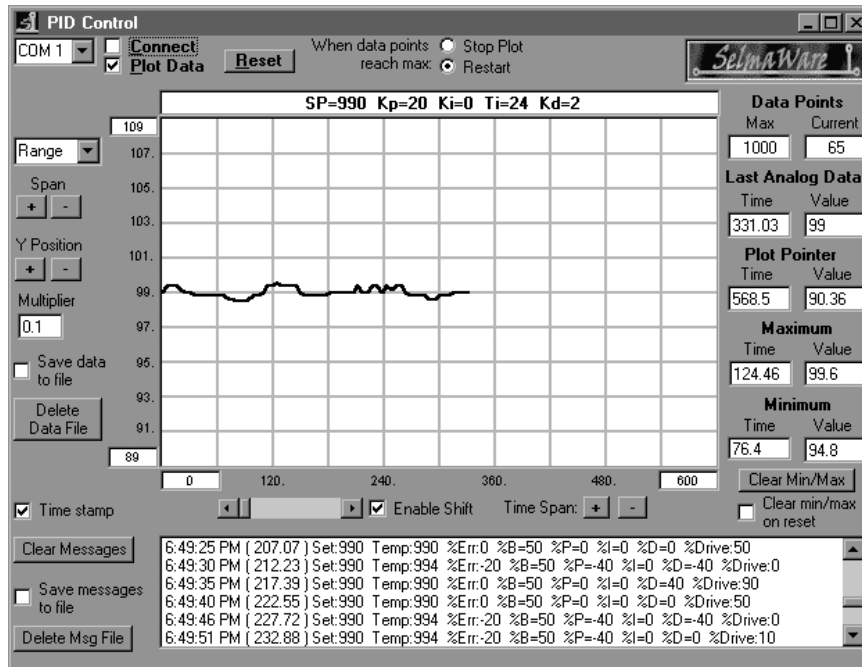
***** DERIVATIVE DRIVE
DerivCalc:
    D = (Err-LastErr) * KD ' Calculate amount of derivative drive
                          ' based on the difference of last error
    DerivDone
    LastErr = Err          ' Store current error for next deriv calc
RETURN
    
```

In this experiment, we will repeat the disturbances on a system with a proportional gain of 2 from Exercise #2. Only this time derivative drive will be used in conjunction with proportional. The fan will once again be supplied power from Vin (pin 19) for 10 seconds at 1 inch. Figure 6.18a is the results of our testing. Figure 6.18b is an Excel graph of the data.

- 1) Settings: Proportional gain of 2 (**Kp=20**), **Ki=0**, derivative gain of 1 (**Kd=2**).
- 2) Allow temperature to stabilize on SPL
- 3) Energize the fan for 10 seconds.

(For this test, our bias temperature was 99.0F)

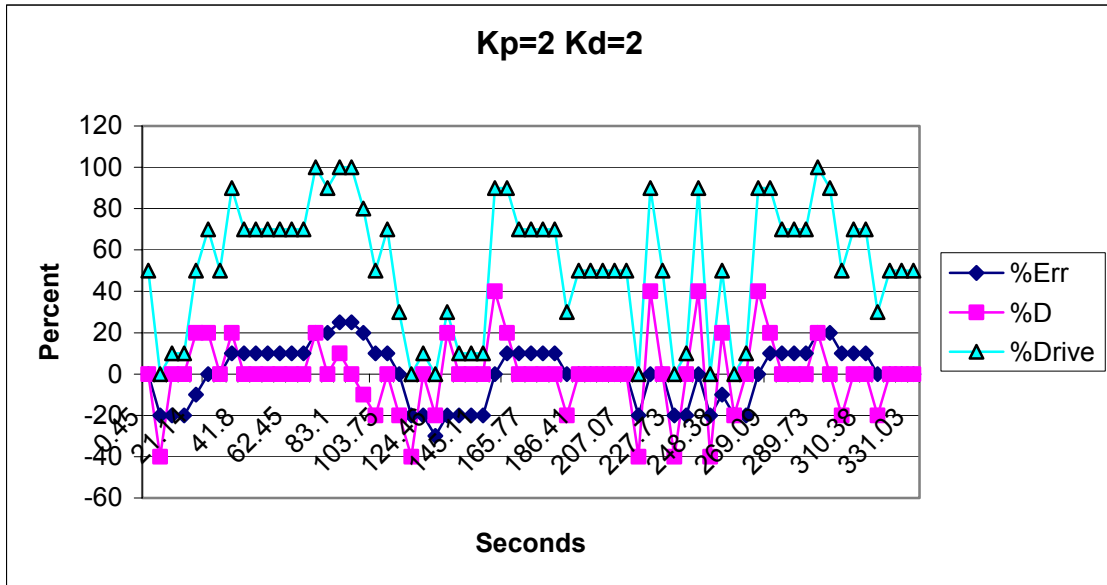
Figure 6.18a: Disturbance Response with Proportional Gain of 2 and Derivative Gain of 2



Compare this with Figure 6.9a using only proportional drive. Note that the system stabilizes much faster with fewer oscillations and overshoot. In the message section, there are 2 consecutive reading of 99.0F. First had a %D drive of 40% since there was a 20% error change (.4F) between the previous and current. The second resulted in a %D of 0% because 2 consecutive readings of 99.0F represents no change in error and a slope of zero.



Figure 6.18b: Graph of Data



6

Note in the above graph how a change in %Err results in a derivative drive. When error is constant, although high, %D is 0. The greater the change in %Err, the greater %D. A positive going %Err (temperature decreasing) results in a positive derivative drive in an effort to stop the change.

Let's work a little math for a temperature change between samples of 99.4 to 99.0 with the setpoint at 99.0F:

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{DERIV}$$

At the time of the first sample, T=99.4:

$$\begin{aligned} \text{Error} &= \text{setpoint} - \text{actual} = 99.0F - 99.4F = -4F \\ \%Error &= \text{Error}/\text{Range} * 100 = -.4F/2F * 100 = -20\% \end{aligned}$$

Temperature dropped to 99.0F at the time of the second sample:

$$\begin{aligned} \text{Error} &= \text{setpoint} - \text{actual} = 99.0F - 99.0F = 0F \\ \%Error &= \text{Error}/\text{Range} * 100 = 0F/2F * 100 = 0\% \\ \%Drive_{PROP} &= \%Error * Kp = 0\% * 2 = 0\% \\ \%Drive_{DERIV} &= (\text{This } \%Error - \text{Last } \%Error) * Kd = (0\% - 20\%) * 1 = 40\% \end{aligned}$$

## Experiment #6: Proportional – Integral – Derivative Control

---

$$\begin{aligned}\%Drive_{TOTAL} &= \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{DERIV} = \\ &50\% + 0\% + 40\% = 90\%\end{aligned}$$

Even though temperature returned to the setpoint providing a 0% drive from proportional, the temperature dropped from the last reading. Derivative control took action based on the change in error in an effort to stop the dropping temperature by adding drive.

If the next temperature reading was 99.2%, what would the final drive be?

$$\begin{aligned}\text{Error} &= \underline{\hspace{2cm}} \\ \%Error &= \underline{\hspace{2cm}} \\ \%Drive_{PROP} &= \underline{\hspace{2cm}} \\ \%Drive_{DERIV} &= \underline{\hspace{2cm}} \\ \%Drive_{TOTAL} &= \underline{\hspace{2cm}}\end{aligned}$$

### Challenge!

1. What would system response be with a derivative gain of 5? Why?
2. With a -0.3 change in temperature from the setpoint, what would total drive be?
3. Test and confirm your theory.

### Proportional-Integral-Derivative Summary

With PID control, 3 separate drive evaluations are performed to calculate the final drive to the control element. Bias drive is used to estimate the drive needed to sustain a setpoint under nominal conditions.

Proportional drive acts by adding an amount of drive in proportion to the amount of error that exists between the setpoint and the actual value. The higher the proportional gain the greater the controller's response, though overshoot and oscillations are more likely. Some error must exist for proportional drive to act, often resulting in a stable but offset condition.

Integral drive acts by integrating a long error over time and taking action based on the total error. Integral is used to drive away error conditions that persist over a period of time. Integral control is also a good choice for a very slow approach to a setpoint when long system settling times are needed and overshoot is undesirable.

Derivative control acts by taking action based on a change of error, often from one reading to the next. It evaluates the slope of the changing output and acts in opposition to the change. Derivative control can prevent hunting and oscillations, but too much drive can send a system into wild oscillations.

Each control mode has its own unique characteristic response to maintaining the desired output to it, such as response time. Volumes have been written on the subject of PID control and tuning. Tuning a PID system involves adjusting the software parameters for each factor. The goal of tuning the system is to adjust the gains so the loop will have optimal performance under dynamic conditions. As mentioned earlier, tuning is as much of an art as it is a science. The basic procedures for tuning a PID controller are as follows. This procedure assumes you can provide or simulate a quickstep change in the error signal:

1. Turn all gains to 0
2. Begin turning up the proportional gain until the system begins to oscillate.
3. Reduce the proportional gain until the oscillations stop, and then drop it by about 20 % more.
4. Increase the derivative term to improve response time and system stability.
5. Next, increase the integral term until the system reaches the point of instability, and then back it off slightly.

**6**

As you gain experience in embedded control, you will see that the characteristics of the process will determine how you should react to error. Consider the following three real-world applications.

What are the important characteristics of these processes that will determine the suitable control scheme? What mode(s) of control do you feel would work the best?

- 1) Similar to our incubator system, the first application is a home project that uses the LM34 to measure the temperature of a 20-gallon aquarium. Water temperature is maintained within  $\pm 1$  degree of 80 °F by varying the duty cycle of a 200-watt heater. Room temperature varies from 65 to 75 degrees.
- 2) The second application controls the acidity (pH) in the production of a cola soft drink. The plant's water supply has a pH of 7.2 to 7.4. The flow stream of water into a batch of soda must be maintained at a pH of 6.8. An upstream valve is opened accordingly to release phosphoric acid into the stream. The pH sensor is relatively slow. The amount of acid needed varies with incoming pH and water flow rate.
- 3) In a plant science research facility at San Diego State University, the surface temperature of a plant's leaf must be held constant. The plant is contained in a small (shoebox sized) greenhouse. A

## Experiment #6: Proportional – Integral – Derivative Control

---

very fast thermocouple sensor rests on the leaf and measures temperature. Disturbances such as changes in wind, sunlight, and plant metabolism can happen quickly and in high magnitudes.

We have just scratched the surface of process control theory through feedback. Our focus has been limited to control action based on feeding back information from the output of our process. When disturbances affect our process, changes in the output are detected and generate an error signal. PID is tuned to drive the error away as quickly as possible. Tight control of the process variable is possible with PID, but the fundamental premise of feedback control is to respond to error. Error is expected and, to a certain degree, tolerated. As we leave this chapter, consider an alternative to feedback control. That is feed-forward control. In feed-forward control you measure those factors that disturb a process. Understanding how they affect the variable we are holding constant will allow for output action to be taken before an error signal results. If you could measure changes in ambient temperature and wind speed from the fan, could you use this information to better control our incubator? Interesting concept isn't it?



## Questions and Challenge

1. Would on/off control of the system be suitable for PID control? Explain.
2. Which type of control (proportional, integral, or derivative) would be best suited for the following?
  - a. To return a system to the setpoint based on the difference between Actual temperature and the setpoint due to a short-lived disturbance: \_\_\_\_\_.
  - b. To minimize the effect that a quick disturbance has on the system: \_\_\_\_\_.
  - c. To reduce the effect that a long-term disturbance has on the system: \_\_\_\_\_.
3. A system has a setpoint of 101.5 degrees, and an allowable band of +/- 0.5 degrees. For a 50% proportional band, what would be the proportional gain? \_\_\_\_\_.
4. A system has a setpoint of 101.5 with a gain of 3. If the Actual temperature were 101.2, what would be the drive due to proportional error? \_\_\_\_\_
5. A system has a derivative gain of 2. If the temperature dropped from 101.8 to 101.3 between readings, with a setpoint of 101.5, what would be the error due to derivative drive? \_\_\_\_\_.

**6**

## Experiment #6: Proportional – Integral – Derivative Control

---

### Final Control Challenge

From a cold condition (incubator at room temperature), find the values of PID control which will bring the incubator to an operating temperature of 95 degrees the quickest with minimal overshoot and hunting. Graph and record your results (note the graph scales):

Kp= \_\_\_\_\_ Ki= \_\_\_\_\_ Ti= \_\_\_\_\_ Kd = \_\_\_\_\_

Time first reached 95.0F: \_\_\_\_\_

Maximum value reached: (Time)\_\_\_\_\_ (Value)\_\_\_\_\_

Next minimum reached: (Time)\_\_\_\_\_ (Value)\_\_\_\_\_

Capture a screen shot (ALT- Prt Scrn) and print using MS Paint.

Find a system:

Find an example of a system that either does or could employ PID control. Discuss how PID control may be implemented to control it.

Alternative Systems to maintain:

1. Use the sample and hold circuit (Figure 2.17) from Experiment #2. Physically connect the heater and sensor with an appropriate material (non-conductive and can withstand the heat). Change the PWMtime 1 because of the much smaller mass and holding of output. Find the 50% bias temperature and attempt to regulate using PID control.
2. Use the output of Pin 8 (drive output without the sample and hold) to drive a solid-state relay controlling a lamp. Place lamp and sensor in an appropriate container. Regulate temperature in this larger incubator system.



## Experiment #7: Real Time Control and Data Logging

Microcontrollers, such as the BASIC Stamp can be good at dealing with very short time periods, such as milliseconds or seconds, but there exist many processes that depend on keeping accurate track of real time (time of day) and possibly even the date or day of the week.

Some examples include heating controls for buildings to set the temperature lower after working hours to conserve power; annealing a metal by heating at different temperatures for specific time periods to temper or strengthen the metal; and logging data over long periods of time for later retrieval and analysis.

This experiment will explore taking action based on specific time of day, action taken based on time intervals, and logging and retrieving data. For this we will need to add real-time features to the circuit built in Experiment #5. Connect the DS-1302 Real Time Clock (RTC) as illustrated in Figure 7.1c and the pushbutton in Figure 7.1d. Figure 7.2 is a board layout sample for placing all the components.

The DS1302 RTC uses an external 32.767kHz crystal oscillator for a time base. Be sure to connect the crystal as close as possible to the IC to maximize time reliability (distance will create more error due to the capacitance effects of the breadboard). The RTC is similar to the BASIC Stamp in that data is stored in registers (RAM memory) that may be written and read. These registers hold the time and date as seconds, minutes, hours, month, etc. The DS1302 also has RAM available for general data storage by the user. For our purposes we will use only the time of day features of the chip. Please see the DS1302 data sheets and Parallax application notes concerning additional features.

Just as with the ADC0831 A/D converter, data is serially shifted into the BASIC Stamp 2 from the IC. This will allow us to access the current time as maintained by the DS1302. In order to set the current time, we can place the IC in a 'write-mode' and serially shift data from the BASIC Stamp 2 into the IC.

Data for the time (and date) is maintained in the DS1302 as Binary Coded Decimals (BCD). This is a subset of the Hexadecimal number base. In Hexadecimal (base 16) a byte is broken up into a high and low nibble, and each is read as a digit.

7

Figure 7.1: Complete Circuit with Real Time Clock

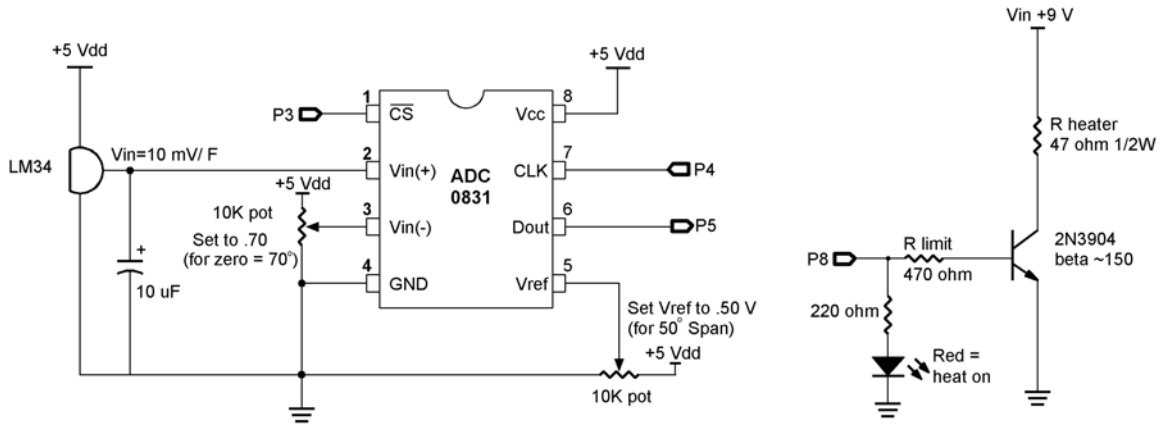


Figure 7.2a: LM34 to ADC0831 Serial A-to-D

Figure 7.2b: BASIC Stamp Driving the Heater with LED Indicator

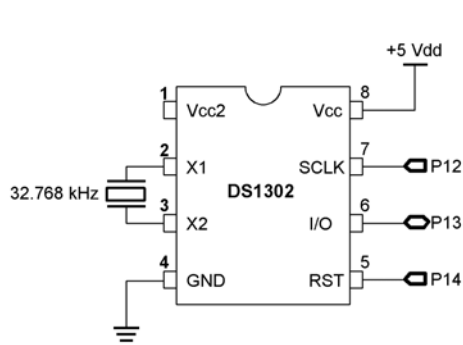


Figure 7.2c: Real Time Clock



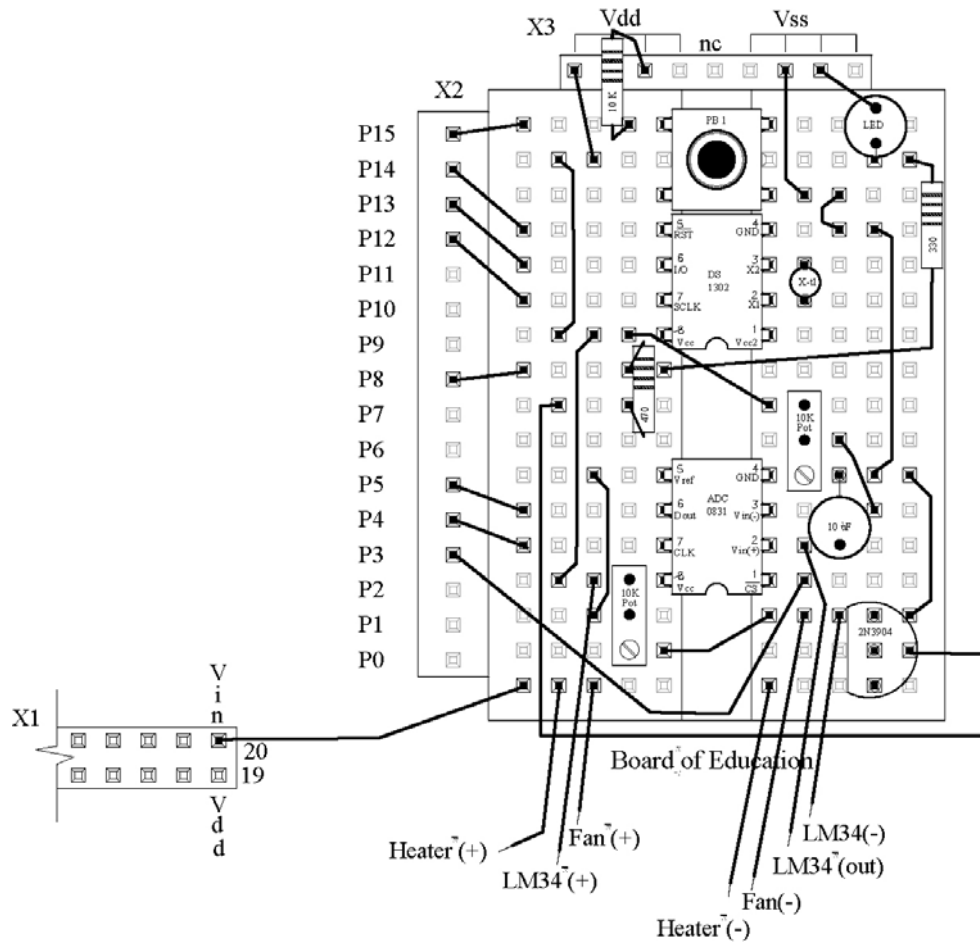
Figure 7.2d: Active-Low Pushbutton



Figure 7.2e: Brushless Fan



Figure 7.2: Sample Component Layout for Experiment #7



7

Take for example the binary number: 01000110. In binary, each place is a higher power of 2, and the decimal equivalent would be:  $64+4+2 = 70$ . With an 8-bit binary number we have a possible decimal range of 0-255.

In Hexadecimal, the byte is broken down into nibbles and converted individually:

0100 0110  
4 6

## Experiment #7: Real-time Control and Data Logging

---

This hexadecimal number is typically written as 86H (Intel format), \$86 (Motorola format) or 86<sub>16</sub> (Scientific format). Since a single unique number represents each nibble, we have a range in binary from 0000 to 1111, or decimal 0-15. In Hexadecimal the values of 10 to 15 are represented using the letters A to F. The hexadecimal full range of a byte of data is \$00 to \$FF. The binary number 10101110 is represented in Hexadecimal as \$AE.

Decimal	Binary	Hexadecimal	Decimal	Binary	Hexadecimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

In Binary Coded Decimal, each nibble is again used to represent a single digit, but since it is a coded DECIMAL number, the valid ranges can only be from 0-9 for each nibble. Our first example of 01000110 is 46<sub>BCD</sub> (a valid BCD number) and \$46 (a valid hexadecimal number). Our second example of 10101110 would be \$AE (valid hexadecimal), but an INVALID BCD value since A and E are not valid decimal numbers.

Our programs will use the hexadecimal notation in setting, or checking the times of the RTC. Additionally, the RTC is set to use a 24-hour clock, so a time of 1:00 PM will be 13:00.

### Exercise #1: Real Time Control

In this first exercise we will simulate a night-setback thermostat of a building. During normal working hours, temperature will be kept at a higher temperature than during the night when energy conservation is needed. To simplify our code, we will use On/Off control instead of the more appropriate control of differential-gap.

Since we will be using our incubator, and to ensure the temperature is above your room temperature, we will use values 100F for working hours and 90 F for nighttime hours. The times for adjusting the temperature up for the day is at 6:00 AM, or 06:00 hours. The time to turn the thermostat down will be 6:00 PM or 18:00 hours.

Program 7.1 is the code for experiment.

```
'Program 7.1 - Real Time On/Off-Control.
'This program will adjust temperture to 100F between the hours
'of 06:00 - 18:00 and 90F between 18:00 and 06:00.

'***** Initialize Settings *****
Time           =      $0553      ' Define initial time
Seconds        =      $00
CTimeLow       CON    $1800      ' Define time to go low temp.
CTimeHigh      CON    $0600      ' Define time to go high temp.

LowTempSP      CON    900        ' Define low temp in tenths of degrees.
HighTempSP     CON    1000       ' Define high temp in tenths of degrees.
'*****

GOSUB SetTime      ' Set RTC (Remark out if time ok)
Setpoint = LowTempSP ' Set initial temperature
CTime = CTimeHigh  ' Set initial change time
Seconds = $00

' Define A/D constants & variables

CS           CON    3      ' 0831 chip select active low from BS2 (P3)
CLK          CON    4      ' Clock pulse from BS2 (P4) to 0831
Dout         CON    5      ' Serial data output from 0831 to BS2 (P5)
DataIn       VAR    BYTE   ' Variable to hold incoming number (0 to 255)
Temp         VAR    WORD   ' Hold the converted value representing temp

TempSpan     CON    5000   ' Full Scale input span in tenths of degrees.
Offset       CON    700    ' Minimum temp. Offset, ADC = 0
Setpoint     VAR    WORD   ' Target Temperature

' Define RTC Constants
' Register values in the RTC
SecReg       CON    %00000
MinReg       CON    %00001
HrsReg       CON    %00010
CtrlReg      CON    %00111
BrstReg      CON    %11111

'Constant for BS2 Pin connections to RTC
RTC_CLK      CON    12     ' Clock pin
RTC_IO       CON    13     ' I/O pin
RTCReset     CON    14     ' Reset pin

'Real Time variables
RTCCmd       VAR    BYTE
RTemp        VAR    BYTE

Time         VAR    WORD   ' Word to hold full time
Hours        VAR    TIME.HIGHBYTE ' High byte is hours
```



## Experiment #7: Real-time Control and Data Logging

---

```
Minutes      VAR      TIME.LOWBYTE      ' Low byte is hours
Seconds      VAR      BYTE
'Time to change variables
CTime       VAR      WORD      ' Word to hold full time
CHours      VAR      CTime.HIGHBYTE  ' High byte is hours
CMinutes    VAR      CTime.LOWBYTE  ' Low byte is minutes
CSeconds    VAR      Byte
'Configure Plot
PAUSE 500      ' Allow buffer to clear
DEBUG "!RSET",CR      ' Reset plot to clear data
DEBUG "!TITL Timer On/Off Control",CR
DEBUG "!PNTS 4000",CR      ' 4000 sample data points
DEBUG "!TMAX 900",CR      ' Max 900 seconds
DEBUG "!SPAN 70,120",CR      ' 70-120 Degrees
DEBUG "!AMUL 0.1",cr      ' Multiply data by 0.1
DEBUG "!CLMM",CR      ' Clear Min/Max
DEBUG "!CLRM",CR      ' Clear messages
DEBUG "!TSMP ON",CR      ' Time Stamp on
DEBUG "!SHFT ON",CR      ' Enable plot shift
DEBUG "!PLOT ON",CR      ' Start plotting
DEBUG "!RSET",CR      ' Reset plot to time 0

Main:
    PAUSE 500
    GOSUB ReadRTCBurst
    GOSUB TimeControl
    GOSUB Getdata
    GOSUB Calc_Temp
    GOSUB Control
    GOSUB Display
    GOTO Main

Getdata:      ' Acquire conversion from 0831
    LOW CS      ' Select the chip
    LOW CLK      ' Ready the clock line.
    SHIFTIN Dout, CLK, MSBPOST,[Datain\9] ' Shift in data
    HIGH CS      ' Stop conversion
    RETURN

Calc_Temp:      ' Convert digital value to
    Temp = TempSpan/255 * Datain/10 + Offset' temp based on Span & Offset variables.
    RETURN

Control:      ' Maintain temperature around setpoint
    IF Temp > SetPoint THEN OFF
    HIGH 8
    RETURN

Off:
    LOW 8
    RETURN
```

```

Display:      'Display real time and time for next change and send data for plotting
              DEBUG "!USRS Current Time:", HEX2 hours,":",HEX2 Minutes,":",HEX2 seconds
              DEBUG "      Next Change at:",HEX2 Chours,":",HEX2 CMinutes,CR
              DEBUG DEC Temp,CR
              DEBUG IBIN OUT8,CR
              RETURN

SetTime:
' ***** Initialize the real time clock to start time
RTemp = $10 : RTCCmd = CtrlReg : GOSUB WriteRTC
' Clear Write Protect bit in control register
RTemp = Hours : RTCCmd = HrsReg : GOSUB WriteRTC ' Set initial hours
RTemp = Minutes : RTCCmd = MinReg : GOSUB WriteRTC ' Set initial minutes
RTemp = Seconds : RTCCmd = SecReg : GOSUB WriteRTC ' Set initial seconds
RTemp = $80 : RTCCmd = CtrlReg : GOSUB WriteRTC
'Set write-protect bit in control register
Return

WriteRTC:      'Write to DS1202 RTC
              HIGH RTCReset
              SHIFTOUT RTC IO, RTC CLK, LSBFIRST, [%0\1,RTCCmd\5,%10\2,RTemp]
              LOW RTCReset
              RETURN

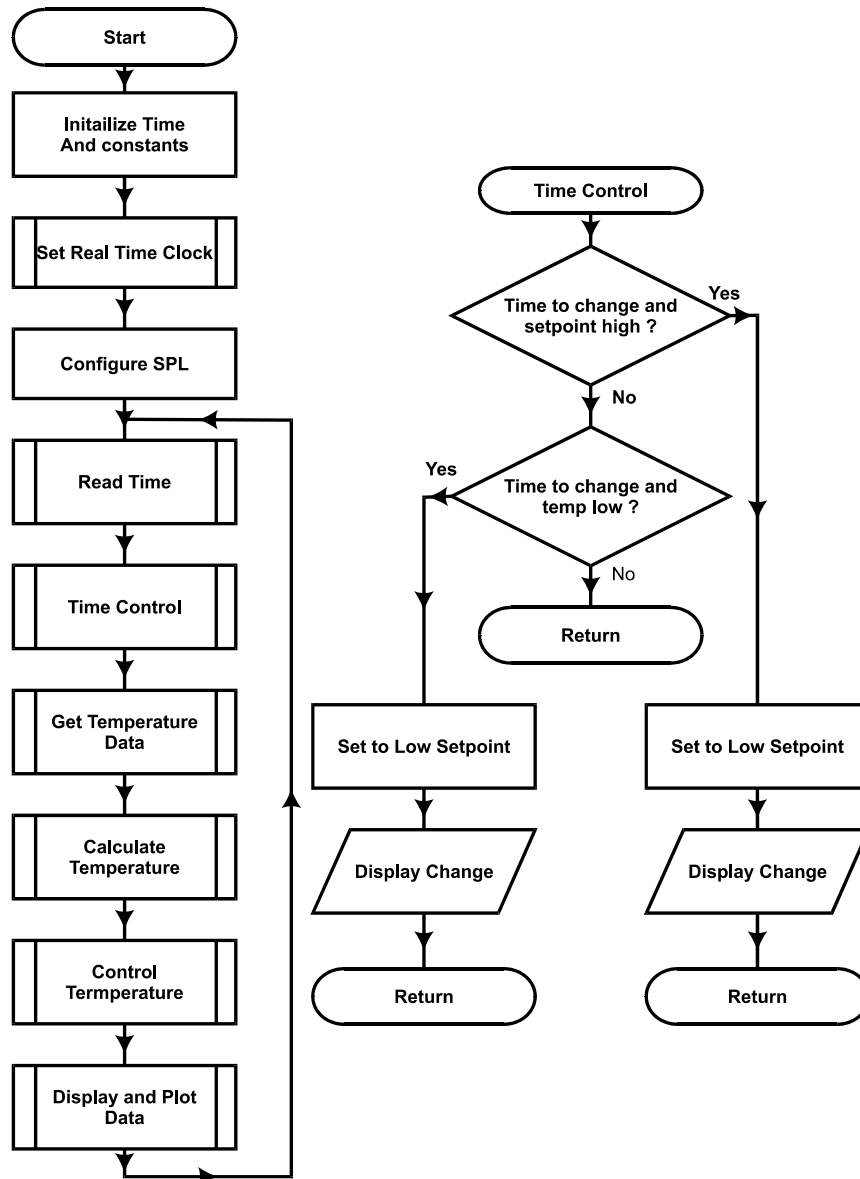
ReadRTCBurst: 'Read all data from RTC
              HIGH RTCReset
              SHIFTOUT RTC IO, RTC CLK, LSBFIRST, [%1\1,BrstReg\5,%10\2]
              SHIFTTIN RTC IO, RTC CLK, LSBPRE, [Seconds,Minutes,Hours]
              LOW RTCReset
              RETURN

TimeControl:  'See if time for a change
              IF (Time = CTimeLow) AND (Setpoint = HighTempSP) THEN LowTemp
              IF (Time = CTimeHigh) AND (Setpoint = LowTempSP) THEN HighTemp
              Return

LowTemp:     'Change to evening temperatures
              Setpoint = LowTempSP
              DEBUG "Time: ", HEX2 Hours,":",HEX2 Minutes,":",HEX2 Seconds
              DEBUG "-- Setpoint = ", DEC SetPoint/10,".0",CR
              CTime = CTimeHigh
              RETURN

HighTemp:    'Change to daytime temperatures
              Setpoint = HighTempSP
              DEBUG "Time: ", HEX2 hours,":",HEX2 minutes,":",HEX2 seconds
              DEBUG "-- Setpoint = ",DEC SetPoint/10,".0",CR
              CTime = CTimeLow
              RETURN
    
```

Figure 7.3: Night Setback Flowchart



Note that the program will reset the time to 05:53 every time the BASIC Stamp is reset. This can occur from program loading, manual Stamp reset, power supply cycling, and sometimes the COM port or computer cycling. The start time is appropriate for what we are discussing in this section, but in later sections you may want to set the values of the start-time to actual values.

```

***** Initialize Settings *****
Time           =      $0553           ' Define initial time
Seconds        =      $00
CTimeLow       CON    $1800           ' Define time to go low temp.
CTimeHigh      CON    $0600           ' Define time to go high temp.

LowTempSP     CON    900              ' Define low temp.
HighTempSP    CON    1000            ' Define high temp.
*****
GOSUB SetTime           ' Set RTC (Remark out if time ok)
    
```

*Note: If power is removed from the DS1302 Real Time Clock it will power-up with unpredictable values in the time registers with GOSUB SetTime remarked out.*

The times for changing temperature and their new values are also set here. GOSUB SetTime sets the real time clock to the specified time. Once the proper time is set, this line may be remarked out and downloaded again to prevent the time from being reset to 05:53 if the BASIC Stamp is reset.



The program uses two sets of variables for time, one to set/hold the current time, and another to hold the time we wish to change the thermostat. Note that the word variable of Time and CTime are further broken down into Hours and Minutes:

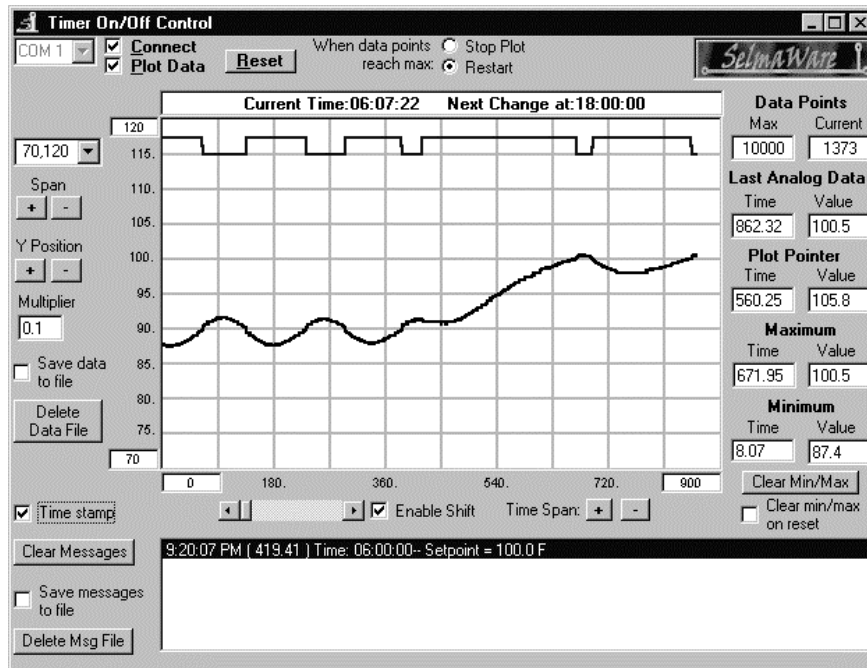
```

Time          VAR    WORD           ' Word to hold full time
Hours         VAR    TIME.HIGHBYTE  ' High byte is hours
Minutes      VAR    TIME.LOWBYTE   ' Low byte is hours
    
```

The variable Hours is assigned to be the high byte of the word variable Time, or those two BCD positions representing the hour. The same is true for minutes and the lower 2 positions. This is a very powerful tool when parts of a single variable need to be addressed individually.

Program 7.1 starts time 7 minutes before switching to the working-hours temperature. This should provide time for temperature to stabilize at the lower temperature. Figure 7.4 is a plot of the run.

Figure 7.4: Time-Controlled 'Building Heating'



The StampPlot Lite user status box displays the current time and the time that the next change is set to occur. The time in the status box may appear to be changing at irregular intervals, but that is a result of timing of the BS2 in displaying the data and not the time kept by the RTC. The message area displays both the time a change occurred and the new setpoint.

The plot illustrates On/Off control at the 90 F setpoint, and the switch to the 100 F setpoint at 06:00. Using the RTC, adding more output devices, and expanding the control section of the code, we could add numerous time-based events to occur over the course of a day.

Download and run program 7.1. Monitor with StampPlot Lite through at least the 06:00 change. You will need to wait another 12 hours to see if it switches back to the low setpoint at 18:00... Have time to wait?



Questions and Challenges:

- 1) The time data stored in the DS1302 uses the \_\_\_\_\_ number system.
- 2) Add a temperature setting of 95 F to be enabled between 4:00PM (16:00) and 6:00 PM (18:00). Modify the starting time and initial temperature to test both times.
- 3) Use the LED on P8 to simulate a house lamp. Add code to energize it at 8:00 PM (20:00) and de-energize it at 11:00 PM (23:00). Modify the starting time of RTC to test both times.

Exercise #2: Interval Timing

Instead of having events occur at defined times of the day, often a process may need to perform actions at certain intervals of time. The annealing process is one such process. In this example a metal is heated at a given temperature for a set amount of time, raised to another temperature for a set amount of time, and then cooled to yet another temperature. This tempers the metal and gives it certain desirable characteristics, such as hardness and tensile strength.



Since we are dealing with intervals of time instead of absolute times, we will need to perform calculations to find the target time that marks the end of an interval. The time interval must be added to the start time of the temperature phase. This sounds simple, but it isn't.

If you remember, our time keeping is performed in BCD, a subset of hexadecimal. When adding values together for time, the BASIC Stamp 2 is working in hexadecimal. Take the example of 38 seconds + 5 seconds. We know this should yield a result of 43 seconds, but since we are really adding \$38 + \$05 (hexadecimal), our result is \$3D (counting 5: \$39, \$3A, \$3B, \$3C, \$3D). If we compare that value to a time from the RTC, it will never occur!

We need to decimal adjust the result. This is done by checking whether the digit exceeds the legal BCD range (>9) and adding 6 if it does. Test this with the above result:

$\$3D + \$06 = \$43$  (counting 6: \$3E, \$3F, \$40, \$41, \$42, \$43). Success! We now have the result we needed for BCD values.

Some other issues we need to contend with is that either the one's or ten's place may need to be adjusted. Depending on the result we may need to carry over into our minutes or hours. Seconds and minutes need to roll over at 60, while hours needs to roll over at 24.

## Experiment #7: Real-time Control and Data Logging

---

This is the general sequence, or algorithm, our program will use:

- Check whether the one's place in seconds is legal BCD (<\$A).
  - No: Add 6 to seconds.
- Check whether the seconds exceeded <60.
  - No: Subtract 60 from seconds. Add one to minutes.
- Check whether the one's place in minutes is legal BCD (<\$A).
  - No: Add 6 to minutes

Here's the code:

```
AdjustTime:  'BCD Time adjust routine
             IF Cseconds.lownib < $A THEN HighSec
             Cseconds = Cseconds + 6
             HighSec:
             IF Cseconds < $60 THEN LowMin
             Cseconds = Cseconds - $60
             Cminutes = Cminutes + 1
             LowMin:
             IF Cminutes.lownib < $A THEN HighMin
             Cminutes = Cminutes + 6
             HighMin:
             IF Cminutes < $60 THEN LowHours
             Cminutes = Cminutes - $60
             Chours = Chours + 1
             LowHours:
             IF Chours.lownib < $A THEN HighHours
             Chours = Chours + 6
             HighHours:
             IF Chours < $24 THEN AdjustDone
             Chours = Chours - $24
             AdjustDone:
             RETURN
```

There is one case where this algorithm won't provide the correct results: When we add a value greater than 6 in any position when the place exceeds 7. Take the example of \$58 + \$08. Adding in hex we get \$60. This returns a valid BCD number, just not one that is computationally correct for BCD. An easy fix for adding 8, is to add 4, adjust, then add 4 more and adjust again. Easier yet would be to not choose timing intervals containing the digits 7, 8, or 9!

In this section we will simulate this process with our incubator, but keep in mind annealing typically heats in thousands of degrees. This is the sequence our annealing process will follow:

- Phase 1: Heat at 95.0 F for 5 minutes.
- Phase 2: Heat to 100 F for 15 minutes.

- Phase 3: Cool to 85.0 F for 10 minutes.
- Process complete, start over for next material sample.

Make the following changes/additions to program 7.1 for Program 7.2.

```
' Program 7.2: Interval Timing
' Controls temperature at 3 levels for set amount of time

'***** Initialize Settings *****
Time    =    $1200          ' Time to set clock
Seconds =    $00

PTemp1 CON    950          ' 1st phase temperature
PTemp2 CON    1000         ' 2nd phase temperature
PTemp3 CON    850          ' 3rd phase temperature

' Do not use digits 7 or above
PTime1 CON    $05          ' Length of phase 1
PTime2 CON    $15          ' Length of phase 2
PTime3 CON    $10          ' Length of phase 3
'*****

Gosub SetTime              ' Set clock (Remark out once set)
Setpoint = PTemp1          ' Initial setpoint to 1st phase temp
Gosub ReadRTCBurst        ' Get clock reading
CTime = Time               ' Set Change time to current
CMinutes = Minutes + PTime1 ' Add Phase 1 time
Gosub AdjustTime          ' BCD adjust time

' Define A/D constants & variables
```

(Middle section of code remains unchanged)

```
TimeControl:              ' Check if ready for change
  IF (Time = CTime) AND (Setpoint = PTemp3) THEN Phase1
  IF (Time = CTime) AND (Setpoint = PTemp1) THEN Phase2
  IF (Time = CTime) AND (Setpoint = PTemp2) THEN Phase3
  Return

Phase1:                   ' Phase 1 - Set for phase 1
  Debug "Phase 3 Complete - Next Sample",CR
  Debug "!BELL",CR
  Setpoint = PTemp1
  Cminutes = Cminutes + PTime1
  GOTO SetNext

Phase2:                   ' Phase 2 - Set for phase 2
  DEBUG "Phase 1 Complete",CR
  Setpoint = PTemp2
```

## Experiment #7: Real-time Control and Data Logging

---

```
Cminutes = Cminutes + PTime2
GOTO SetNext

Phase3:                                     ' Phase 3 - Set for phase 3
  DEBUG "Phase 2 Complete",CR
  Setpoint = PTemp3
  Cminutes = Cminutes + PTime3

SetNext:
  GOSUB AdjustTime                         ' BCD adjust time
  DEBUG "Time: ", hex2 hours,":",hex2 minutes,":",hex2 seconds
  DEBUG "-- Setpoint: ", dec setpoint,cr
  RETURN

AdjustTime:                                'BCD Time adjust routine
  IF Cseconds.lownib < $A THEN HighSec
  Cseconds = Cseconds + 6

  HighSec:
  IF Cseconds < $60 THEN LowMin
  Cseconds = Cseconds - $60
  Cminutes = Cminutes + 1

  LowMin:
  IF Cminutes.lownib < $A THEN HighMin
  Cminutes = Cminutes + 6

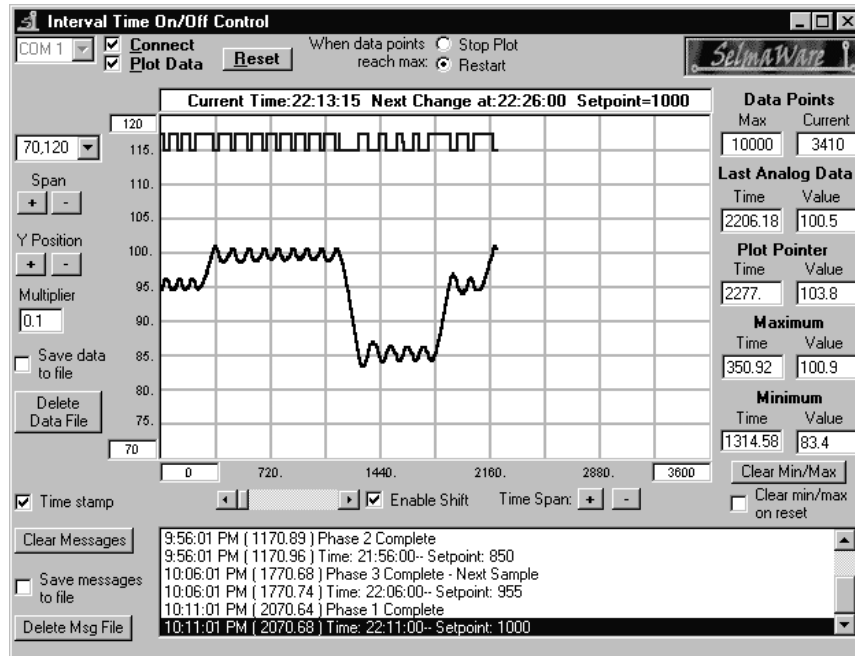
  HighMin:
  IF Cminutes < $60 THEN LowHours
  Cminutes = Cminutes - $60
  Chours = Chours + 1

  LowHours:
  IF Chours.lownib < $A THEN HighHours
  Chours = Chours + 6

  HighHours:
  IF Chours < $24 THEN AdjustDone
  Chours = Chours - $24

  AdjustDone:
  RETURN
```

Figure 7.5: Interval Timer Plot



7

Figure 7.5 is a screen shot of a sample run. Notice there is 3 distinct temperature phases, and then it repeats.

Download and run program 7.2. Use StampPlot Lite to monitor your system.

Questions and Challenges:

- 1) Why is using the RTC preferable for long-interval timing instead of PBASIC Pause commands?
- 2) Add the following hexadecimal values and decimal adjust the results (show work):  
\$15 + \$15
- 3) Modify the program to add a 5- minute phase 4 at 80.0 F.

### Exercise #3: Data Logging

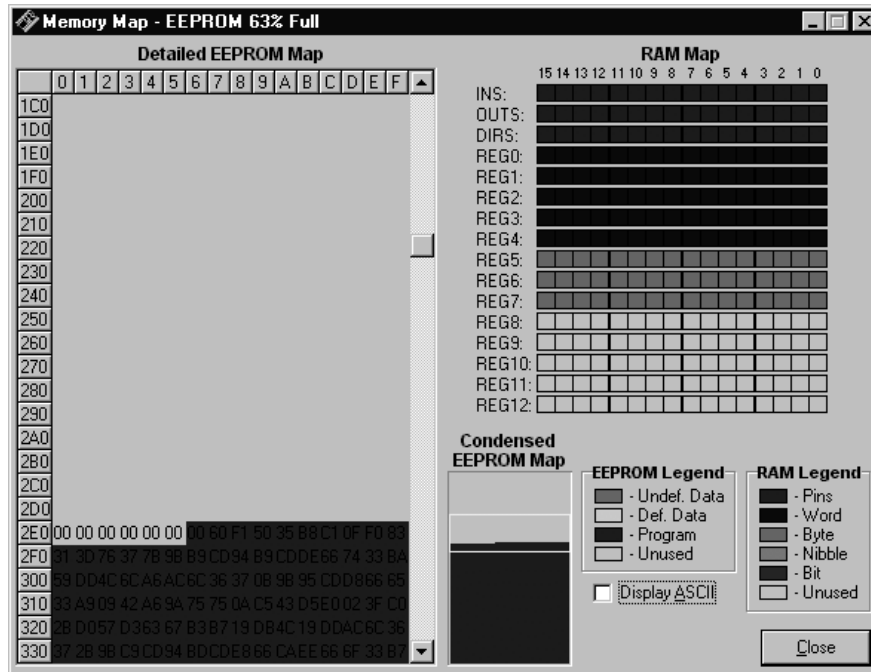
Data logging does not fall into the area of process-control, but it is an important subject, and since the RTC is connected, this is an appropriate time to discuss it. The majority of our experiments have used StampPlot Lite to graphically display current conditions in our system. Of course, one of the biggest benefits of microcontrollers are that they are self-contained and do not require a PC. All the experiments in this text would operate properly whether the data was being plotted on a PC or not. We simply wouldn't have any direct feedback of the status.

Data logging is used to collect data and store it locally by the microcontroller. This data may then be downloaded later for analysis. Some examples of this include remote weather stations and Space-Shuttle experiments. Due to location or other factors, it may not be practical to be collecting data on a PC in real time.

When data is logged to memory, it is important to make sure the hardware, programming, and time keeping is as stable as possible. The data-logger may not be accessed for very long periods of time. Unintentionally resetting the Stamp will usually lose your data and start the programming over. The Stamp is easily reset by pressing the reset button, by connecting it to a computer sometimes, or possibly even a temporary loss power.

Just as BASIC Stamp programs are stored in non-volatile memory (remains with loss of power) in EEPROM, we may also write data directly into the EEPROM. The BASIC Stamp 2 has 2048 bytes of available EEPROM memory for program and data storage. Figure 7.6 shows the BASIC Stamp 2 memory map. Programs are stored at the end of memory, allowing us to use the top of memory for data.

Figure 7.6: BASIC Stamp 2 Memory Map



7

When data is logged, we will need to retrieve from the unit both the value and time that a reading was recorded. In recording the data, the time of the data may be recorded into memory along with the value. This would require 3 bytes to be used for each measurement: Hour, Minute and Value (optionally, the second may be recorded depending on the need). Or, we can record the time that the data recording commenced or started; storing only the data at a known interval the program can then extrapolate the time of each measurement. This only requires 1 byte per measurement for the value with a one-time recording of the start time.

But what happens if the controller is inadvertently reset, such as when connecting it to the computer for the data dump? What would happen if the start time or the current log sample location were lost? What if the RTC was reset at some point so the time clock was reset? We can also use the EEPROM to keep track of important data, such as the start time and next memory location in the event the controller is reset preventing important data from being lost. The program we have developed helps to ensure data is not lost on inadvertent resets.

## Experiment #7: Real-time Control and Data Logging

---

A power outage is one eventuality our program will not deal with. Upon power loss, the BS2 will be able to recover current data, but the RTC will probably contain non-valid times. Some possible fixes for this are running the project off a battery, or adding a high-capacity capacitor to the RTC as per the data sheets. A 'super-cap', a gold-plated capacitor can maintain the RTC time for many hours or even days. One other option may be to continually write the current time to EEPROM so that in the event power is lost, the most recent time may be written back to the RTC. But EEPROMs have limited write-cycles. After several thousand writes the EEPROM will eventually fail, so we may not want to repeatedly write to the same location.

How much data can we hold? After the program is downloaded, there is approximately 700 bytes left of the original 2K of EEPROM in the BASIC Stamp 2. This will allow us to store 700 logged pieces of data. At 5-minute intervals, how long could we store data before memory is full? Some other options for storing data may be on the RTC in general user registers, or on a separate device, such as a serial EEPROM.

*Note: Do not log more data than the EEPROM has room for – overwriting code space will cause the BASIC Stamp program to fail!*

The **WRITE** command is used to write data into memory:

```
WRITE Memory Address, byte value
```

The **READ** command is used to read data from memory into a variable:

```
READ Memory Address, byte variable
```

We will use the DS1302 as an interval timer that will control when samples are taken. For this experiment we will collect outside temperature over a long period of time and then download the results to StampPlot Lite.

Program 7.3 is the code for our data logger. It is sufficiently different from our other programs to require a full listing though much of the code can be re-used.

```
'Program 7.3 - Real Time Data Logging
'This program will record in EEPROM memory the temperature
'at the specified intervals using the real time clock.

'*** Set Init. Time and Logging Interval *****
Time      =      $2246                ' Initialization Time

' Do not use digits > 6
Interval  CON    $05                ' Sample interval (in BCD minutes) for logging

Samples CON    500                ' Number of samples to acquire
Stop Reset CON    0                ' When full, 0=reset, 1 = stop logging
'*****
```



```

'Define RTC Constants
'Register values in the RTC
SecReg      CON    %00000
MinReg      CON    %00001
HrsReg      CON    %00010
CtrlReg     CON    %00111
BrstReg     CON    %11111

'Constant for BS2 Pin connections to RTC
RTC_CLK     CON    12
RTC_IO      CON    13
RTCReset    CON    14

'Real Time variables
RTCCmd      VAR    BYTE
RTemp       VAR    BYTE

' Current Time Variables
Time        VAR    WORD
Hours       VAR    Time.HIGHBYTE
Minutes     VAR    Time.LOWBYTE
Seconds     VAR    BYTE

' Log-Time variables
LTime       VAR    WORD
LHours      VAR    LTime.HIGHBYTE
LMinutes    VAR    LTime.LOWBYTE
LSeconds    VAR    BYTE

' Start time variables
STime       VAR    WORD
SHours      VAR    STime.HIGHBYTE
SMinutes    VAR    STime.LOWBYTE

MemAddr VAR    WORD           ' Current memory location for storage

' Define A/D constants & variables
CS          CON    3           ' 0831 chip select active low from BS2 (P3)
CLK         CON    4           ' Clock pulse from BS2 (P4) to 0831
Dout        CON    5           ' Serial data output from 0831 to BS2 (P5)
DataIn      VAR    BYTE       ' Variable to hold incoming number (0 to 255)
Temp        VAR    WORD       ' Hold the converted value representing temp
TempSpan    CON    5000       ' Full Scale input span (5000 = 50 degrees span)
Offset      CON    700        ' Minimum temp. Offset. (700 = 70 degrees)

DIR15 = 0
DIR8 = 1
PB          VAR    IN15       ' Pushbutton
LED         CON    8          ' LED
LOW LED     ' LED off

***** Initialize *****

```

## Experiment #7: Real-time Control and Data Logging

---

```
INIT:
  ' To set the new time, hold down PB until LED goes off
  DEBUG "Hold button now to initialize clock and logging",cr
  HIGH 8                                ' LED On
  PAUSE 2000
  IF PB = 1 THEN SkipSet                ' If PB not pressed, don't store time or restart
  Seconds = $00
  GOSUB SetTime
  GOSUB RecoveryData
  DEBUG "Release button",CR

SkipSet:
  LOW LED                                ' LED OFF
  IF PB = 0 THEN SkipSet                ' Wait for PB release
  READ 0,MemAddr.HIGHBYTE               ' Read recovery data of memory address and time
  READ 1,MemAddr.LOWBYTE
  READ 2,SHours
  Read 3,SMinutes

  ' Initialize time keeping variables
  GOSUB ReadRTCBurst                    ' Read current time
  LHours = Hours                        ' Set change hours to current
  LMinutes = Minutes                    ' Set change minutes to current
  LSeconds = $00                        ' Set change seconds to 00

  LMinutes = LMinutes + Interval        ' Add interval to get first log time
  GOSUB AdjustTime                      ' Decimal adjust new time

'***** Main Loop *****
Main:
  PAUSE 500
  GOSUB Control
  GOSUB Getdata
  GOSUB Calc Temp
  GOSUB ReadRTCBurst
  GOSUB Display
  GOSUB TimeControl
  GOTO Main

Getdata:
  LOW CS                                ' Acquire conversion from 0831
  LOW CLK                                ' Select the chip
  SHIFTLIN Dout, CLK, MSBPOST,[Datain\9] ' Ready the clock line.
  HIGH CS                                 ' Shift in data
  RETURN                                  ' conversion

Calc Temp:
  Temp = TempSpan/255 * Datain/10 + Offset ' Convert digital value to
  RETURN                                  ' temp based on Span &
                                          ' Offset variables.

Control:
  IF PB = 0 THEN DumpData                ' If PB pressed, plot recorded data
```

```

RETURN

TimeControl:                                ' Check if time for reading
  IF (Time = LTime) AND (MemAddr-4 < Samples) THEN SaveData
  RETURN

SaveData:                                    ' Write ADC reading to memory
  WRITE MemAddr, DataIn                      ' Store data into EEPROM
  HIGH 8:PAUSE 250:LOW 8                    ' Blink LED
  MemAddr = MemAddr + 1                     ' Increment memory location for next reading
  WRITE 0,MemAddr.HIGHBYTE                  ' Update recovery data
  WRITE 1,MemAddr.LOWBYTE

  LMinutes = LMinutes + Interval            ' Update for next interval
  IF MemAddr-4 < Samples THEN AdjustTime    ' If samples not full, continue
  IF Stop Reset = 1 THEN Dont Reset        ' If samples full, restart or end logging
  GOSUB RecoveryData

Dont_Reset:

AdjustTime:                                  'Decimal adjust Time
  IF LSeconds.LOWNIB < $A THEN HighSec
  LSeconds = LSeconds + 6

  HighSec:
  If LSeconds < $60 THEN LowMin
  LSeconds = LSeconds - $60
  LMinutes = LMinutes + 1

  LowMin:
  IF LMinutes.LOWNIB < $A THEN HighMin
  LMinutes = LMinutes + 6

  HighMin:
  IF LMinutes < $60 THEN LowHours
  LMinutes = LMinutes - $60
  LHours = LHours + 1

  LowHours:
  IF LHours.LOWNIB < $A THEN HighHours
  LHours = LHours + 6

  HighHours:
  IF LHours < $24 THEN AdjustDone
  LHours = LHours - $24

  AdjustDone:
  RETURN

Display:                                     'Display real time and time for next log reading
  DEBUG "!USRS Time:", HEX2 hours,":",HEX2 minutes,":",HEX2 seconds
  DEBUG " Sample Due:",HEX2 LHours,":",HEX2 LMinutes,":",HEX2 LSeconds
  DEBUG " # ",DEC MemAddr-4, " Temp now = ", DEC Temp,CR
  DEBUG DEC Temp,CR
  Return

SetTime:
' ***** Initialize the real time clock to start time
  RTemp = $10 : RTCCmd = CtrlReg : GOSUB WriteRTC
  ' Clear Write Protect bit in control register

```

## Experiment #7: Real-time Control and Data Logging

---

```
RTemp = Hours : RTCCmd = HrsReg : GOSUB WriteRTC ' Set initial hours
RTemp = Minutes : RTCCmd = MinReg : GOSUB WriteRTC ' Set initial minutes
RTemp = Seconds : RTCCmd = SecReg : GOSUB WriteRTC ' Set initial seconds
RTemp = $80 : RTCCmd = CtrlReg : GOSUB WriteRTC
' Set write-protect bit in control register
Return

WriteRTC: ' Write to DS1302 RTC
HIGH RTCReset
SHIFTOUT RTC IO, RTC CLK, LSBFIRST, [%0\1,RTCCmd\5,%10\2,RTemp]
LOW RTCReset
RETURN

ReadRTCBurst: ' Read all data from RTC
HIGH RTCReset
SHIFTOUT RTC IO, RTC CLK, LSBFIRST, [%1\1,BrstReg\5,%10\2]
SHIFTIN RTC IO, RTC CLK, LSBPRE, [Seconds,Minutes,Hours]
LOW RTCReset
RETURN

RecoveryData: ' Stores data for recovery from restart
MemAddr = 4 ' Set starting location
WRITE 0,MemAddr.HIGHBYTE ' Write to EEPROM
WRITE 1,MemAddr.LOWBYTE
WRITE 2,Hours ' Save start time in EEPROM
WRITE 3,Minutes

Return

'*** Download and sisplay logged data ***
DumpData:
'Configure Plot
PAUSE 500 ' Allow buffer to clear
DEBUG "!RSET",CR ' Reset plot to clear data
DEBUG "!TITL Interval Data Logging",CR ' Title the plot
DEBUG "!PNTS 2000",CR ' 2000 sample data points
DEBUG "!TMAX ", DEC MemAddr/7+1,CR ' Time based on number of samples

DEBUG "!SPAN ",DEC offset/10,"",DEC (TempSpan/10 + Offset) / 10,CR
DEBUG "!AMUL .1",cr ' Multiply data by 0.1
DEBUG "!CLMM",CR ' Clear Min/Max
DEBUG "!CLRM",CR ' Clear messages
DEBUG "!TSMP OFF",CR ' Time Stamping off
DEBUG "!SHFT ON",CR ' Enable plot shift
DEBUG "!DELM",CR ' Delete message file
DEBUG "!SAVM ON",CR ' Save messages (logged data) to file
DEBUG "!PLOT ON",CR ' Start plotting

PAUSE 500
DEBUG "!RSET",CR ' Reset plot to time 0

X VAR Word
LTime = STime ' Set log time = start time
DEBUG "Point,Time,Temperature",CR ' message header
```

```

FOR x = 4 to MemAddr-1           ' Loop through memory locations
  READ x,DataIn                 ' Read data stored in memory
  GOSUB Calc Temp               ' Calculate temp based on data
  LMinutes = LMinutes + Interval ' Add interval to get stored time
  GOSUB AdjustTime             ' Decimal adjust time
                                ' Display message data
  DEBUG DEC X-4," ",HEX2 LHours,":",HEX2 LMinutes," ",DEC Temp,CR
  DEBUG DEC Temp,CR            ' Plot temperature
  HIGH LED
  PAUSE 100                     ' Pause 0.1 second for spacing between data
  LOW LED
NEXT

DEBUG "!PLOT OFF",CR           ' Disable plotting
LTime = Time                    ' Set Log time to current time
LMinutes = LMinutes + Interval  ' Add interval to set for next data logging
GOSUB AdjustTime

HIGH 8                          ' LED ON
DEBUG "Hold button now to reset log",CR
PAUSE 2000
IF PB = 1 THEN SkipReset        ' If button not pressed, skip restart
GOSUB RecoveryData              ' Restart - save new recovery data
DEBUG "Release button now",CR

SkipReset:
LOW 8                          ' LED Off
IF PB = 0 THEN SkipReset        ' Wait for button release

Return

```



We'll discuss operation and major blocks in our code. At the top of the code is the initialization information:

```

'*** Set Init. Time and Logging Interval *****
Time = $2246                      ' Initialization Time

Interval CON $05                   ' Do not use digits > 6
                                ' Sample interval (in BCD minutes) for logging

Samples CON 500                    ' Number of samples to acquire
Stop Reset CON 0                   ' When full, 0=reset, 1 = stop logging
'*****

```

This data defines the time to set the RTC, how long the interval between logging should be, and how many samples should be logged. **Stop\_Reset** is defines whether to stop logging (1) or reset (0) and start over destroying the old data when the maximum samples are collected.

The pushbutton has several purposes:

## Experiment #7: Real-time Control and Data Logging

---

1. On- Power up or Reset of the BS2, a message will appear informing you to hold down the pushbutton to initialize the clock and logging (the LED will light for this also). If the button is held down, the value of time in the initialization section will be used to set the RTC and logging will be reset to the start. Recovery data will be written to EEPROM for the next reset.

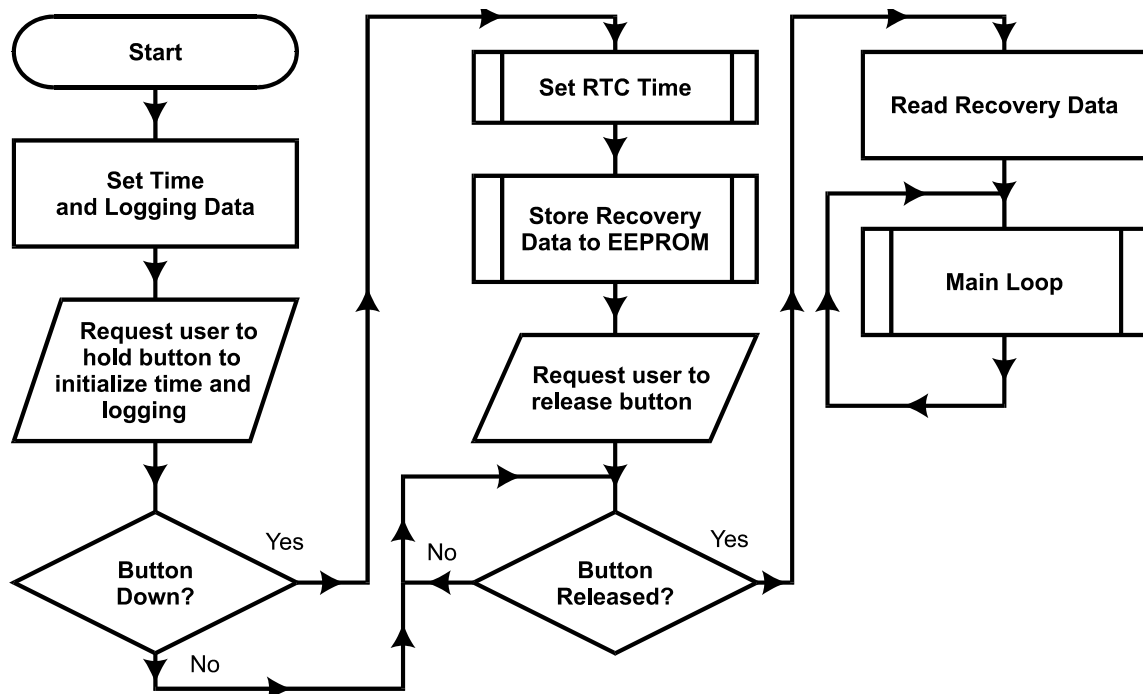
```
RecoveryData:          ' Stores data for recovery from restart
  MemAddr = 4          ' Set starting location
  WRITE 0,MemAddr.HIGHBYTE ' Write to EEPROM
  WRITE 1,MemAddr.LOWBYTE
  WRITE 2,Hours        ' Save start time in EEPROM
  WRITE 3,Minutes
Return
```

Note that a memory address is a word-sized value and must be saved as high and low byte.

2. During logging, if the pushbutton is pressed, the data will be 'dumped' or downloaded. We will be using StampPlot Lite to capture and plot the data as it is dumped. The data is NOT destroyed and the logger will continue to log new data.
3. At the end of a data dump, if the pushbutton is held down, the data logger will reset the log to the start destroying old data and resetting the start time of logging.

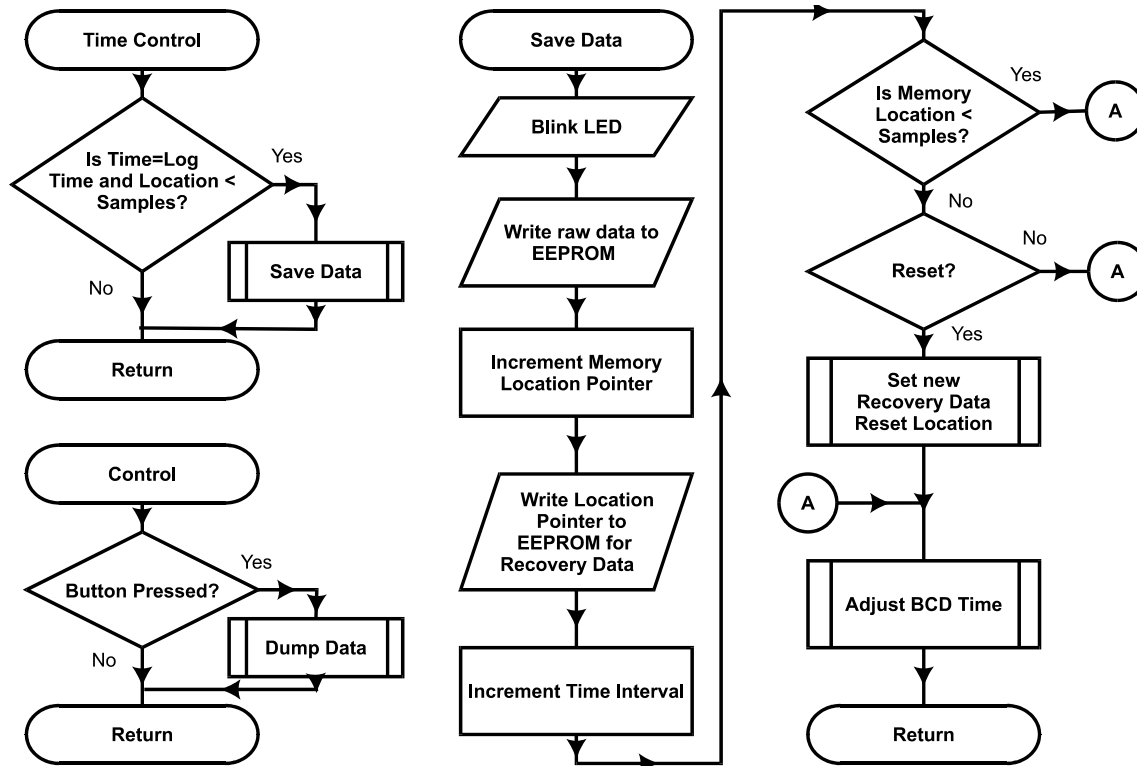
If the BASIC Stamp 2 is reset and the button is NOT held down, the program will read recovery data of current memory location and start time from the EEPROM. The RTC time will NOT be reset. It should be maintaining proper time through the reset UNLESS power was lost. Figure 7.7 is the flowchart of the initialization of the program.

Figure 7.7: Logging Initialization Routine



7

Figure 7.8: Control and Saving Routines



Time Control routine (Figure 7.8) is used to determine if it is time to save new data to memory. This is contingent on the memory location for samples being less than the number of samples specified.

```

TimeControl:                                     ' Check if time for reading
    IF (Time = LTime) AND (MemAddr-4 < Samples) THEN SaveData
    RETURN
    
```

The Save Data routine is called from Time Control when it is time to write a new sample to memory. The current `DataIn` (value read from the ADC) is stored in the current memory address, and the memory address is incremented for the next cycle. The next interval time is calculated (and later BCD adjusted). If the



maximum number of samples is reached, depending on the `Stop_Reset` value, data logging will either ceased (see Time Control) or logging will start over.

Note that the raw `DataIn` value from the ADC is stored and not the temperature-calculated value. This allows the data to be stored in one byte instead of two as a word. The stored value will be converted into temperature when it is 'dumped' to the PC.

```
SaveData:                                ' Write ADC reading to memory
    WRITE MemAddr, DataIn                 ' Store data into EEPROM
    HIGH 8:PAUSE 250:LOW 8                ' Blink LED
    MemAddr = MemAddr + 1                 ' Increment memory location for reading

    WRITE 0,MemAddr.HIGHBYTE              ' Update recovery data
    WRITE 1,MemAddr.LOWBYTE

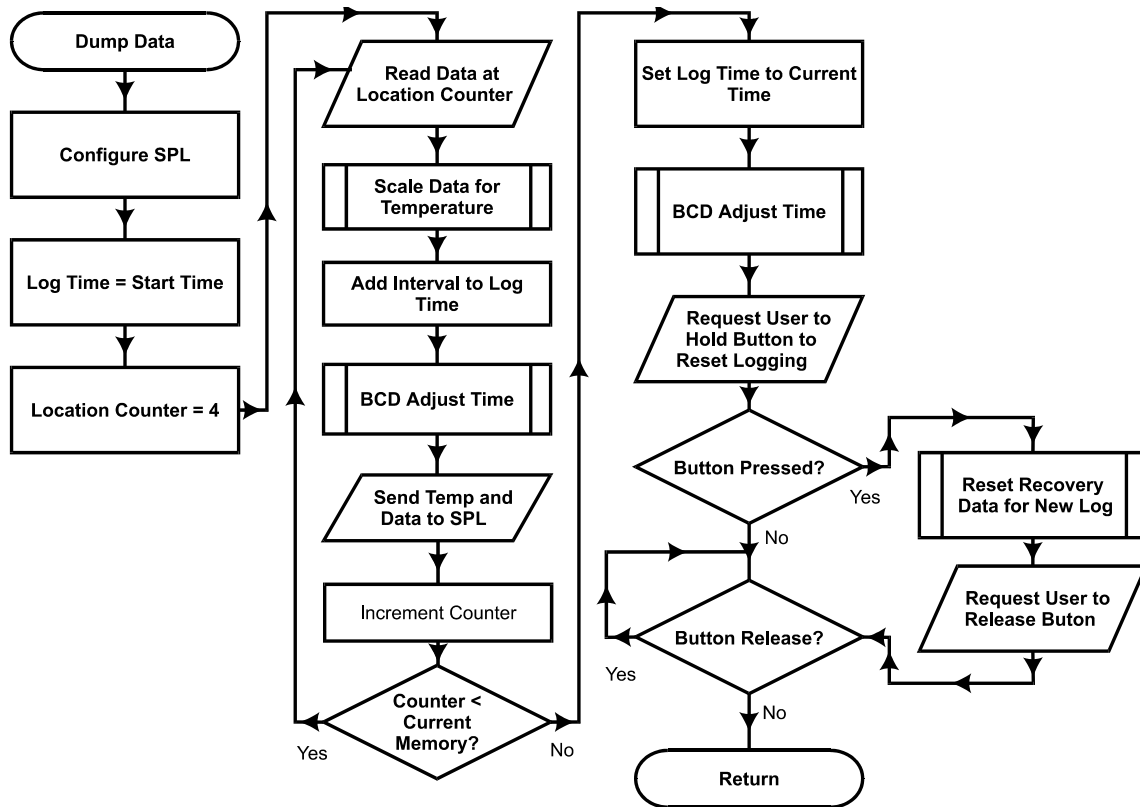
    LMinutes = LMinutes + Interval        ' Update for next interval
    IF MemAddr-4 < Samples THEN AdjustTime ' If samples not full, continue
    IF Stop_Reset = 1 THEN Dont_Reset     ' If samples full, restart or end logging
    GOSUB RecoveryData
Dont_Reset:
AdjustTime:
```

Figure 7.9 illustrates the flow of the `DumpData` routine. When the pushbutton is pressed, `Dump Data` configures `StampPlot` for plotting, and creates a loop reading through the logged values, converting them to temperatures, and sending the values for plotting and the message window. Note that the log time is set to the start time, and the timing interval is added to the log time each loop iteration to determine the original time the data was logged.

Once the loop is complete, the log time is set back to the current time and the user is requested to hold down the pushbutton to reset the logging to start (the LED will light for indication also).

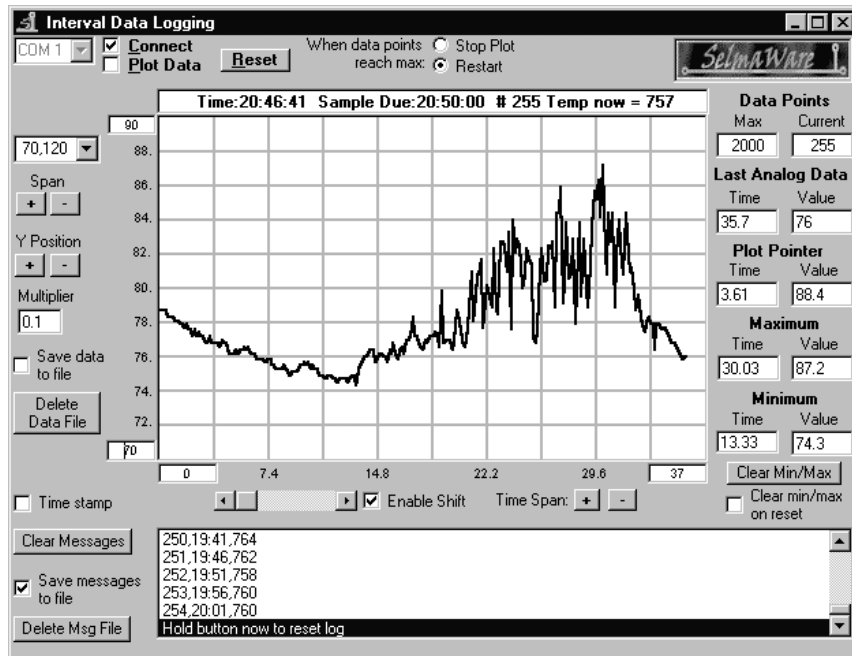


Figure 7.9: Data Dump Routine Flowchart



Once **DumpData** is complete, the control time will be updated to the time of the next sample, and logging will continue from the point it left off, allowing downloading without affecting the stored data. Figure 7.10a is a screenshot of our collected data dumped to StampPlot Lite.

Figure 7.10a: Sample Data Dump of Logged Data

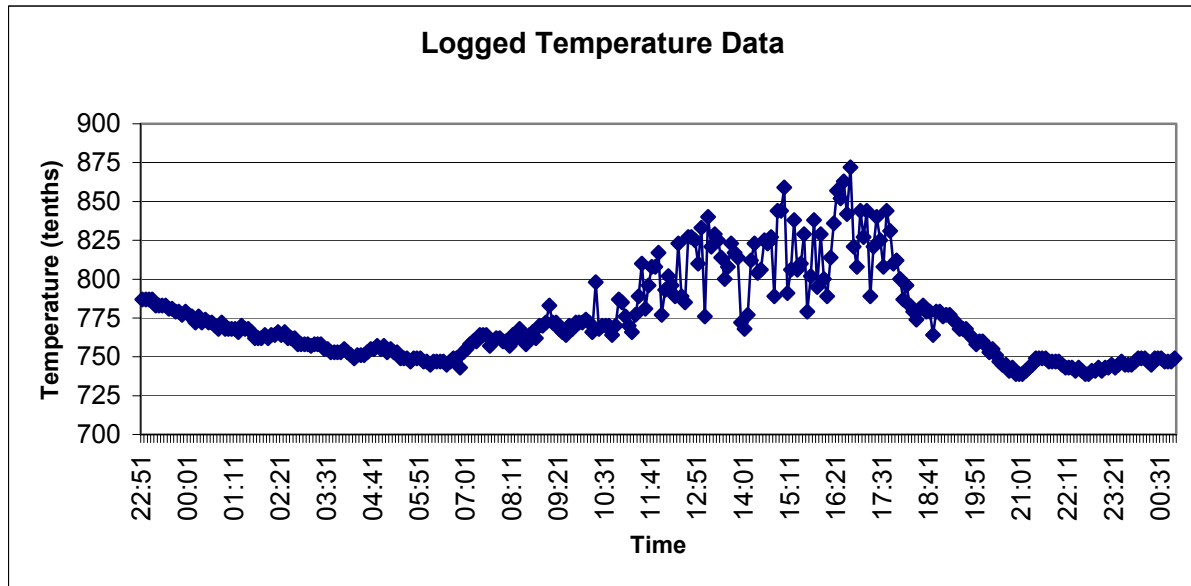


7

The format of the message data is suitable for importing into a spreadsheet for graphing, as seen in Figure 7.10b. A portion of the saved message file is as follows:

```
Point,Time,Temperature
0,22:51,787
1,22:56,787
2,23:01,787
3,23:06,787
4,23:11,783
5,23:16,783
```

Figure 7.10b: Imported Message Data to Excel



We set the incubator outside of a window and recorded outdoor temperatures.. Outdoor temperature can be tricky due to effects of wind cooling, sunlight heating, and thermal layers in the canister trapping heat. We had quite a few spikes in readings. Can you do better ?

Our plot shows temperatures from 22:51 to 22:51 the next night. Note the rise and falls in temperature during the day. The expected high for the day was 88F, and we came pretty close!

Of course, we are not restricted s temperature range 70F- 120F. Refer back to Experiment #4. Software range values are determined by the span and offset voltage settings to the ADC0831.

TempSpan	CON	5000	' Full Scale input span (5000 = 50 degrees span)
Offset	CON	700	' Minimum temp. Offset. (700 = 70 degrees)

**Questions and Challenges:**

1. Perform Logging of a System: Determine a temperature that you want to log over a long period of time. This may be outdoors, the room temperature (does the heating/cooling change during evening hours?), or maybe some other slow changing system (a water tank in the sun?).
  - 1) Determine the range of expected values for temperature. Set the span and offset variables and potentiometers appropriately.
  - 2) Determine the length of time you need to collect the data (one day? the weekend?). Based on a maximum of 50 samples, calculate the interval time needed for logging.
  - 3) Ready to program? If you are going to move your BOE, make sure it is running on a battery that will last throughout the logging!
  - 4) We also want to check the accuracy of the RTC in this experiment, so when the program is ready to download:
    - a) Open the Windows clock and pick an upcoming time.
    - b) Set the start time to this upcoming time (remember to use 24-hour time)  
Time = \$1530
    - c) 5 seconds before the initializing time, download the file to the BS2 and hold the circuit push-button down until the LED goes off or debug window instructs you to release it.
    - d) Use the Debug Window or StampPlot to note the BS2 and the PC Times.  
BS2: \_\_\_\_\_ PC: \_\_\_\_\_ Difference: \_\_\_\_\_
    - e) Let your data record! After the 1<sup>st</sup> sample is done, you may test the data dump by pressing the push-button.
    - f) After you are finished recording data, run and connect StampPlot Lite, press the push-button to dump the data.
    - g) Use the Debug Window or StampPlot to note the BS2 and the PC Time.  
BS2: \_\_\_\_\_ PC: \_\_\_\_\_ Difference: \_\_\_\_\_
    - h) Extrapolate the time-error for 24 hours: \_\_\_\_\_

Did the data conform to your expectations?

## Experiment #7: Real-time Control and Data Logging

---

2. Discuss the 'system' you monitored and conclusions of your results.
  
3. How much time error was calculated over a 24-hour period? How could this error be compensated for in software?
  
4. Why is it important to limit the amount of data that can be stored in the BS2?

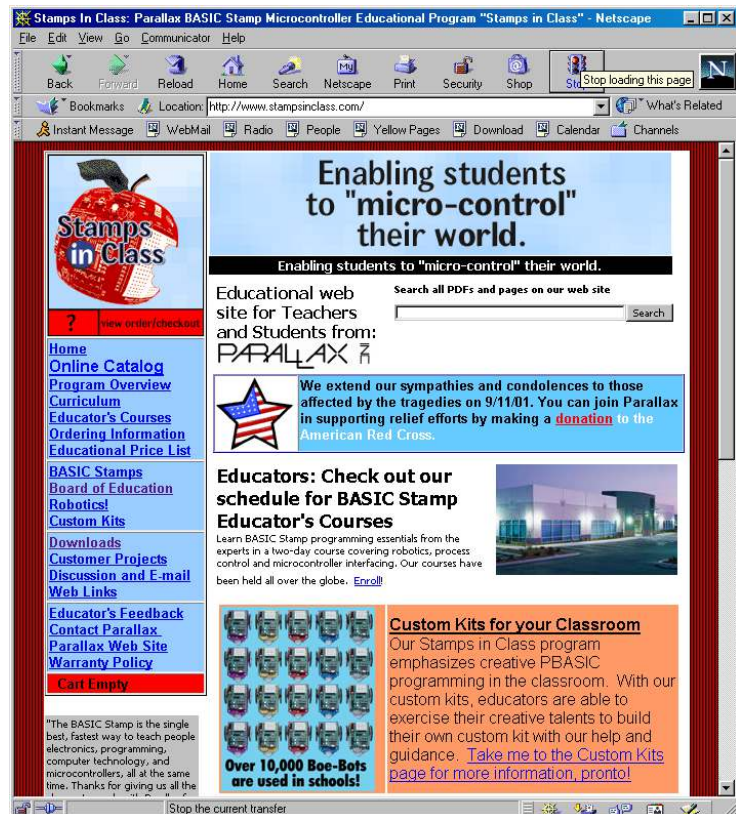
## Appendix A: StampPlot Lite

StampPlot Lite is an application developed by SelmaWare Solutions for the Industrial Control series. The application allows plotting and capture of analog, digital and general data.

### Downloading and Installing StampPlot Lite

StampPlot Lite may be downloaded from the Stamps in Class web site at <http://www.stampsinclass.com>. The program is installed by double-clicking on the setup.exe icon and accepting the default directories.

To download StampPlot Lite, click on the “Downloads” button on our web site and scroll down to the “Industrial Control” section.



## Appendix A: StampPlot Lite

---

Data from the BASIC Stamp is processed in one of four ways by the application:

### Analog Values

Any string sent beginning with a numeric value will be processed as an analog value and graphed.

```
Debug DEC 100, 13      'Plot the number 100
```

### Digital Values

Any string sent beginning with '%' will be processed as digital values. A separate digital plot will be started for each binary value in the string. For example, "%1001" will plot four digital values. Up to a 9-bit value may be sent. Once digital plots are started, caution should be used to always send the same number of bits since the plots are position-order dependent.

```
Debug IBIN4 INC, 13    'Plots 4 digital values
```

### Control Settings

Any string beginning with '!' will be processed as a control setting. The various settings of the application may be controlled from the BASIC Stamp using specified control words and values, if required.

```
Debug "!AMAX 200", 13  'Sets analog maximum for plot to 200
Debug "!RSET", 13      'Resets the plot
```

### Other Strings

All other strings simply will be added to the running message list box.

```
Debug "Hello world!", 13
```

Note that each instance of data **MUST** end with a carriage return (13 or CR).



The steps for using BASIC Stamp programs with StampPlot Lite are as follows:

1. Start StampPlot Lite through your Start/Programs/StampPlot/StampPlot Lite icon.
2. Enter and run your BASIC Stamp program from the BASIC Stamp editor.
3. Close the blue BASIC Stamp debug window by clicking on the “close” box.
4. Select the COM port and click 'Connect' checkbox.
5. Click the 'Plot Data' checkbox.
6. Some programs may require you to reset the Board of Education (BASIC Stamp) to catch initial configuration and control settings. Do this by pressing the Reset button on the board.
7. Prior to downloading (running) another program to the BASIC Stamp, be sure to uncheck the StampPlot 'Connect' checkbox or your COM port will be locked by StampPlot Lite.

The plot will acquire analog and digital data and store it temporarily so that it may be resized or shifted on the screen. The number of data points collected is adjustable. Once the data points reach maximum, the plot must either be stopped or reset.

The following program will perform some configuration settings, continually plot and display the value of X on StampPlot, and plot the four digital bits of the value of X. Enter the program and use the steps above to test it with StampPlot Lite.

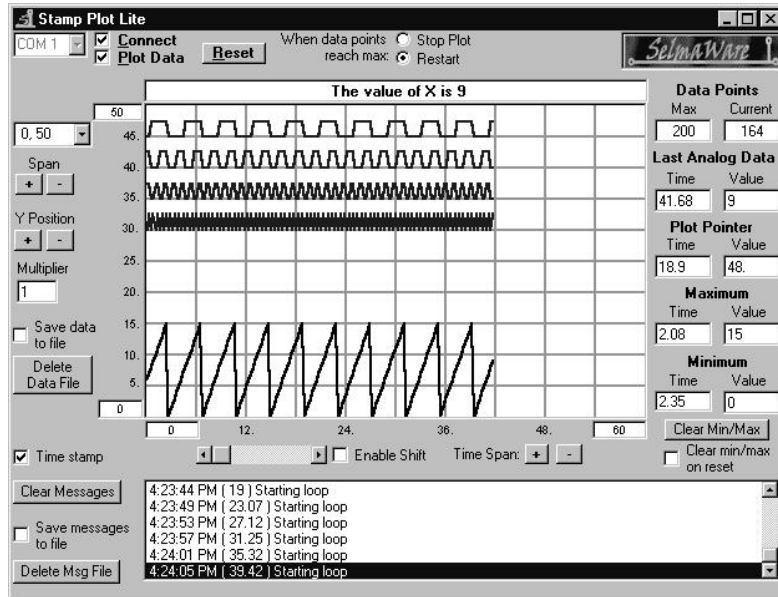
```
'Appendix A Program, StampPlot Example
'Configure StampPlot
Pause 500: Debug "!RSET",CR           ' Variable for counting
Debug "!SPAN 0, 50",CR              ' Short pause and reset
Debug "!TSMP ON",CR                 ' Span the analog range
Debug "!TMAX 60",CR                 ' Time Stamp the messages
Debug "!RSET",CR                     ' Set plot to 60 seconds max
                                     ' Reset the plot

X var Byte
Loop:
Debug "Starting loop", CR           ' Message that loop is resetting.
For X = 0 to 15                      ' For-Next loop to count to 15
Debug DEC X, CR                      ' Plot Analog value of X
Debug IBIN4 X, CR                    ' Plot digital bits of X
                                     ' Change the User Status message.
Debug "!USRS The value of X is ", DEC X, CR
Pause 200                             ' Short pause
Next
Goto Loop                             ' Restart
```

## Appendix A: StampPlot Lite

---

Below is a screen shot of StampPlot Lite showing the above program plotted.



### Tool Text Help

If a copy of StampPlot Lite is running on your computer, you may place the cursor over each control for 'Tool Text Help.' The following is a brief summary of each control. The BASIC Stamp programmable command, where applicable, is in brackets:

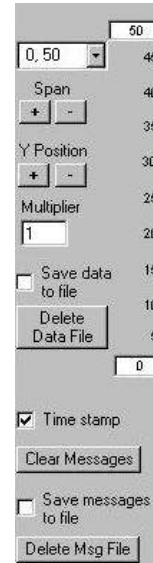
#### Top Section: General Controls



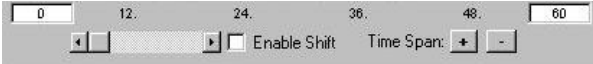
- Com 1: Drop down to select the applicable COM port.
- Connect: Connects the application to the selected COM port.
- Plot Data: Allows plotting of incoming data. Deselecting this control will stop the plotting of data but will allow messages and other actions to continue. [ !PLOT ON/OFF]
- Reset: Clears the plot, resets to time 0, clears minimum and maximum value (optional). [ !RSET]
- Stop Plot: When maximum data points are reached, the plot stops (Plot Data becomes unchecked). [!MAXS]
- Reset Plot: When maximum data points are reached, the plot resets. [ !MAXR]
- User Status: (showing "The value of X is 9") Optional status messages from the BASIC Stamp may place data here. [ !USRS message ]

Left Section: Primarily for Setting the Analog Plot

- Span Drop-Down box: Allows a selection of pre-defined plot ranges. Use of the BASIC Stamp command !SPAN will add a range to this drop-down. [ !SPAN minvalue, maxvalue ]
- + and - buttons: Respectively double or halve the span. The minimum value does not change.
- Multiplier: Defines the amount incoming BASIC Stamp analog data will be multiplied by prior to plotting or saving to file. [ !AMUL value ]
- Save Data to File: Saves the incoming data to a text file in the application directory called "stampdat.txt." If time stamping is enabled, each record will be marked with the current system time and the number of seconds since the last reset. The value of the data point for analog and digital values will also be recorded. Each record will have the following form:
  - Time of day, seconds since reset, analog data point, analog value, digital data point, digital data value. Note that each record is comma delineated for importing into a spreadsheet, if desired.
  - *Note: Data is saved ONLY when ANALOG data arrives. To force saving when no analog data is recorded, debug a value such as zero ( DEBUG DEC 0, CR).[ !SAVD ON/OFF ]*
- Delete Data File: Deletes "stampdat.txt." If data saving is enabled, the file will be re-created after deleting it. [ !DELD ]
- Analog Minimum and Maximum values: These may be manually changed. Tab off, or click another control, to set the new value. [ !AMIN value !AMAX value <or> !SPAN minvalue, maxvalue ]
- Time Stamp: Enables time stamping of messages and data to the file. It includes both the current time and seconds since the last reset. [ !TSMP ON/OFF ]
- Clear Messages: Clears the messages in the listbox. [ !CLRM ]
- Save messages to file: Saves messages to the file "stampmsg.txt" in the application directory. Messages will be saved the same way they appear in the message box. [ !SAVM ON/OFF ]
- Delete Msg file: Deletes "stampmsg.txt" in the application directory. If the "Save Messages.." is enabled, the file will be re-created. [ !DELM ]



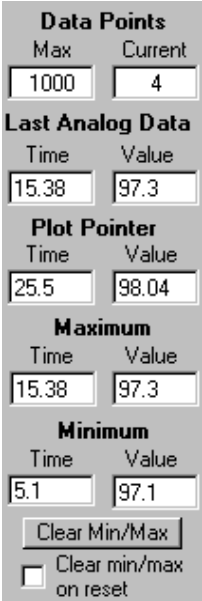
Bottom Section: Plot Shift and Time Span



- The minimum and maximum times of the plot may be set manually. Tab-off or click another control to apply the value. [ !TMIN value and !TMAX value ]
- Scroll Bar: If the plot extends beyond the current limits, the scroll bar may be used to reposition the plot (if collecting data, Enable Shift must be on).
- Enable Shift: Allows the plot to shift automatically when maximum plotted time is exceeded. Also enables operation of the scroll bar when collecting data. Note: Shifting of the plot during data collection may cause time errors in the data as the plot refreshes. [ !SHFT ON/OFF ]
- +/-: Respectively doubles or halves the time span of the plot. The minimum value of the plot will not change.

Right Section: Plot Data

- Data Points: To allow the plot to be manipulated, data is stored in memory. The maximum number of points (either analog or digital) that may be recorded is the Max. 'Current' displays the current data point being stored. Once the maximum is reached, the plot will either reset or stop, depending on the configuration. [ !PNTS 1000 ]
- Last Analog Data: Displays the time since reset and the last analog value plotted.
- Plot Pointer: Moving the plot pointer on the display shows the current analog value and time for that point on the plot.
- Maximum: Records the maximum analog value and the time it was reached.
- Minimum: Records the minimum analog value and the time it was reached.
- Clear Min/Max: Clears the recorded minimum and maximum values. [ !CLMM ]
- Clear min/max on reset: Allows a reset to clear the minimum and maximum values. [!CMMR]





## Display Control and Zoom

- Moving the cursor on the plot will set the plot pointer time and value to the current position.
- Double-clicking the plot will shift display modes from yellow to white background and thin lines to thick lines for better printing (ALT-Print Screen to capture form to the clipboard) and projected display.
- Shift-Click (hold) and Drag allows you to specify an area of the plot to zoom.

## BASIC Stamp Control and Configuration Commands

The majority of the plot configuration and controls may be set from within the BASIC Stamp program. To use these commands, simply debug from the BASIC Stamp. All commands must end with a CR (ASCII 13): DEBUG "!PLOT ON", CR

Command	Description
!TTL message	Sets the title of the form to the message
!USRS message	Sets the User Status box to display the message
!BELL	Sounds the bell on the PC
!AMAX value	Sets the plot maximum analog value
!AMIN value	Sets the plot minimum analog value
!SPAN minValue, maxValue	Sets the plots analog maximum and minimum as above (also adds the range to the Range Drop-Down box).
!AMUL value	Sets the value to multiply incoming data by
!TMAX value	Sets the plot maximum time (seconds)
!TMIN value	Sets the plot minimum time (seconds)
!PNTS value	Sets the number of data points to collect
!PLOT ON/OFF	Enables/disables the plotting of data
!RSET	Resets the plot and all data
!CLRM	Clears the message list
!CLMM	Clears the min/max recorded values
!CMMR ON/OFF	Enables/Disables clearing of Min/Max recorded values on reset
!MAXS	Sets the plot to STOP when data points are full
!MAXR	Sets the plot to RESET when data points are full
!SHFT ON/OFF	Enables/disables the plot from shifting when recording data (may cause a loss of data accuracy if enabled)
!TSMP ON/OFF	Enables time stamping of list messages; messages and data saved to files
!SAVD ON/OFF	Enables saving of analog and digital data to files
!SAVM ON/OFF	Enables saving of messages to a file
!DELD	Deletes the saved data file
!DELM	Deletes the saved message file

### Additional Application Notes

The amount of the plot that is used is dependent on the number of data points and the rate at which they are transmitted. For example, if you wish 60 seconds of data to fill the screen and are transmitting from the BASIC Stamp at a maximum rate of 100 msec (Pause 100 + processing time):  $60/.1 = 600$  data points.

The application needs a minimum regular pause in the reception of data for complete processing. A pause of 10 msec is typically sufficient for a fairly fast computer. If the application senses it cannot keep up with incoming data, a message box will appear and the application will disconnect. Some indications that the computer cannot keep up are: garbled data, no plotting, and the inability to affect any controls (locks up).

The greater the number of data points and the higher the current data point, the longer the plot will take to respond to plot shifts as it redraws. For faster, reliable configuration from the BASIC Stamp on initial power up or resetting, the following is recommended:

- Pause for 500msec at the start to allow StampPlot's buffer to clear.
- Perform a StampPlot **RESET** (!RSET) prior to making configuration changes to allow the data points to be cleared so redrawing is not performed.
- Reset (!RSET) at the end of the configuration resets the plot to time 0.

As with any application, the best way to learn it is to play. Have fun!

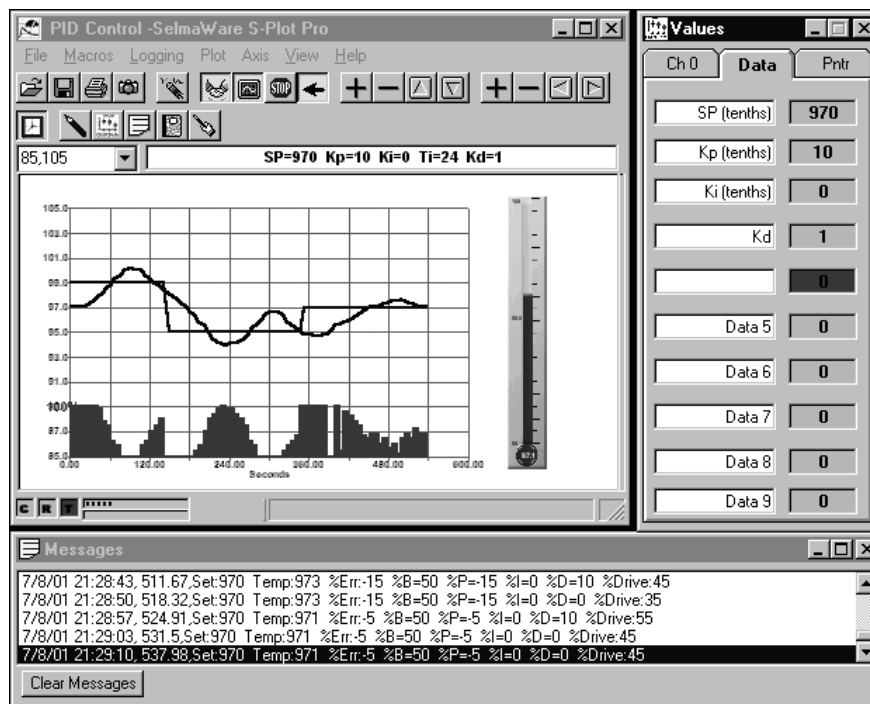


Upgrade to StampPlot Pro for your Industrial Control experiments!

A few of the features include:

- Plotting of multiple analog channels.
- Saving of plots and images.
- Advanced logging features.
- Ability to draw graphics.
- PC based macros of instructions, including data manipulation.
- Interactive capabilities for adjustments on the fly.

This screen shot of StampPlot Pro shows the PID exercise in Experiment #7 plotting actual and setpoint temperature along with the %Drive (bottom bars).



The gain values may be entered in the data window to be read by your BS2!

A PC based macro shows the temperature in the thermometer.

Examples based on the Industrial Control text are posted in the on-line tutorials. Download and evaluate free!

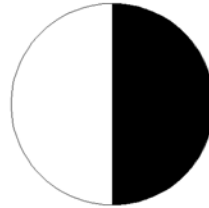
<http://www.selmaware.com/>



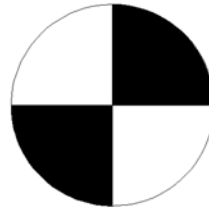


## Appendix B: Fan Encoder Printouts

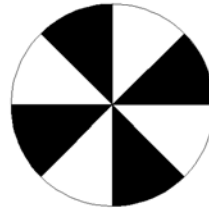
These printouts are full size, and should be an ideal fit for your fan used as a digital switch in Experiment #2.



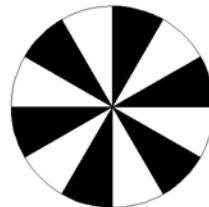
1 cycle/revolution



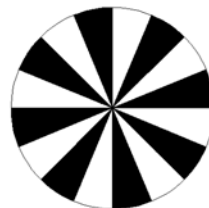
2 cycles/revolution



4 cycles/revolution



6 cycles/revolution



8 cycles/revolution





Appendix C:  
Potter Brumfield SSR  
Dat asheet

Appendix C consists of the Potter Brumfield “Hockey Puck” Solid State Relay. Their datasheets may be downloaded from <http://www.pandbrelays.com/>.



## SSRT series

### "Hockey Puck" Solid State Relay With Snubberless Triac Output

File E29244

File E29244 UL Recognized for Canada

#### Features

- Standard "hockey puck" package.
- Exposed ceramic base plate for reduced thermal resistance.
- Floating terminal design.
- Low cost snubberless triac outputs.
- 10A & 25A rms versions.
- AC & DC input versions.
- 4000V rms isolation.

#### Engineering Data

**Form:** 1 Form A (SPST-NO).

**Duty:** Continuous.

**Isolation:** 4000V rms minimum, input - output.

**Isolation Resistance:**  $10^{10}$  ohms @ 500VDC minimum.

**Capacitance:** 10 pF maximum (input to output).

**Temperature Range:**

**Storage:** -40°C to +120°C

**Operating Temperature:** -25°C to +80°C

**Case Material:** Plastic, UL rated 94V-0.

**Base Plate Material:** Ceramic.

**Case and Mounting:** Refer to outline dimension.

**Termination:** Refer to outline dimension.

**Approximate Weight:** 3.5 oz. (98g).

#### Ordering Information

Sample Part Number ▶ **SSRT -240 D 10**

1. **Basic Series:** SSRT = "hockey puck" triac output solid state relay

2. **Line Voltage:** 240 = 24-240 VAC

3. **Input Type & Voltage:** A = 90-280 VAC/VDC linear  
D = 3-32 VDC constant current

4. **Maximum Switching Rating:** 10 = .05-10A rms, mounted to heatsink  
25 = .05-25A rms, mounted to heatsink

#### Stock Items – The following items are normally maintained in stock for immediate delivery.

SSRT240A10 SSRT240D10  
SSRF240A25 SSRF240D25

#### Input Specifications

Parameter	AC/DC Input/AC Output	DC Input/AC Output
Control Voltage Range $V_{RH}$	90-280VAC/VDC	3-32VDC
Must Operate Voltage $V_{M(ON)}$ (Max.)	90VAC/VDC	3VDC
Must Release Voltage $V_{M(REL)}$ (Min.)	10VAC/VDC	1VDC
Input Current	15mA Max. @ 90VAC	15mA Max. @ 5VDC



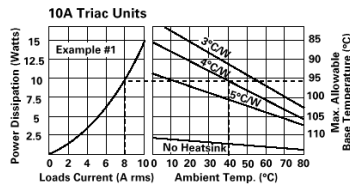
Parameter	Conditions	Units	SSRT-240A10 & SSRT-240D10	SSRT-240A25 & SSRT-240D25
Load Voltage Range $V_L$		V rms	24-240	
Repetitive Blocking Voltage (Min.)		V peak	≥600	
Load Current Range $I_L^*$	Resistive	A rms	.05-10	0.5-25
Single Cycle Surge Current (Min.)		A peak	100	200
Leakage Current (Off-State) (Max.)	$f = 60 \text{ Hz}$ , $V_L = \text{Nom.}$ (120 or 240 V rms)	mA rms	0.5	
On-State Voltage Drop (Max.)	$I_L = \text{Max.}$	V peak	1.7	
Static dv/dt (Off-State) (Min.)		V/μs	200	
Thermal Resistance, Junction to Case ( $R_{\theta j-c}$ ) (Max.)		°C/W	0.6	0.6
Turn-On Time (Max.)	$f = 60 \text{ Hz}$ .	ms	8.3	
Turn-Off Time (Max.)	$f = 60 \text{ Hz}$ .	ms	8.3	
$I^2 t$ Rating	$t = 8.3 \text{ ms}$	A <sup>2</sup> Sec.	60	260
Load Power Factor Rating	$I_L = \text{Max.}$		0.5-1.0	

\*See Derating Curves

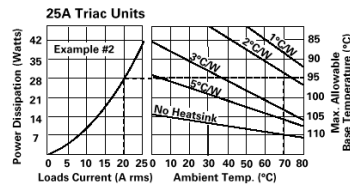
### Electrical Characteristics (Thermal Derating Curves)

#### How To Use These Curves

Knowing maximum load current and maximum ambient temperature, use derating curves to determine the minimum required heat sink and maximum allowable base plate temperature. On left hand power dissipation curve, locate the point corresponding to maximum load current. Extend a line to the right from that point to the intersection of vertical line on right hand chart corresponding to maximum ambient temperature. From heat sink curve, read directly or extrapolate required heat sink size. Extend the line farther to the right and read on the right hand scale the maximum allowable base plate temperature.



**Example #1:**  
**Given:** IL 8A rms @ 40°C  
**Find:** Heatsink required  
**Solution:** From 10A curve  
 Heatsink = 4°C/W



**Example #2:**  
**Given:** IL = 20A rms @ 70°C  
**Find:** Required heatsink  
**Solution:** Heatsink = 2°C/W

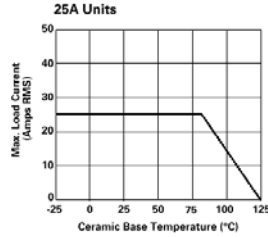
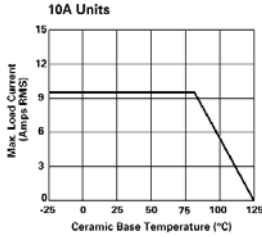
### Heatsink Dimensions

- We recommend that solid State Relay Modules be mounted to a heatsink sufficient to maintain the module's base temperature at less than 85°C under worst case ambient temperature and load conditions.
- The heatsink mounting surface should be a smooth (30-40 micro-inch finish), flat (30-40 micro-inch flatness across mating area), un-painted surface which is clean and free of oxidation.
- An even coating of thermal compound (Dow Corning DC340 or equivalent) should be applied to both the heatsink and module mounting surfaces and spread to a uniform depth of .002" to eliminate all air pockets.
- The module should be mounted to the heatsink using two#10 screws. The mounting screws should be torqued to 10 inch-pounds by alternately tightening the screws one quarter turn at a time.

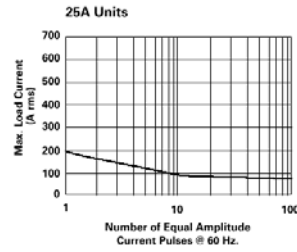
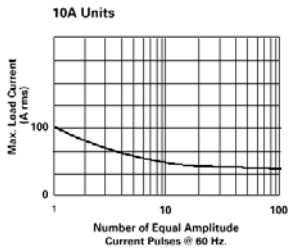
# Appendix C: Potter Brumfield SSR Datasheet



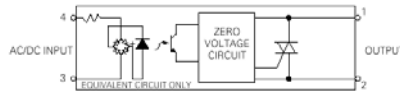
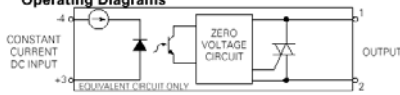
## Load Current vs. Base Temperature



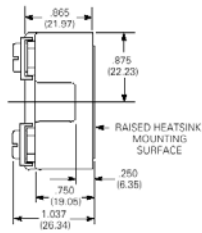
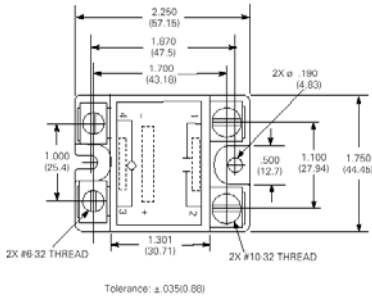
## Allowable Peak Surge vs. Duration/Expected Lifetime



## Operating Diagrams



## Outline Dimensions



Tyco Electronics  
700 Westpark Drive  
Peachtree City, GA 30269-1488

Specifications and availability subject to change without notice.  
13C777B Printed in U.S.A. IH/2-00





Appendix D:  
National Semiconductor  
LM34 Datasheet

Appendix D consists of the National Semiconductor LM34 datasheet. This appendix includes the first five (5) pages of the 12-page datasheet. Should you wish to see more applications of the LM34 than are shown in this datasheet, the entire document may be downloaded from <http://www.national.com/ds/LM/LM34.pdf>.



July 1999

## LM34 Precision Fahrenheit Temperature Sensors

### General Description

The LM34 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Fahrenheit temperature. The LM34 thus has an advantage over linear temperature sensors calibrated in degrees Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Fahrenheit scaling. The LM34 does not require any external calibration or trimming to provide typical accuracies of  $\pm 1/2^\circ\text{F}$  at room temperature and  $\pm 1 1/2^\circ\text{F}$  over a full  $-50$  to  $+300^\circ\text{F}$  temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM34's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies or with plus and minus supplies. As it draws only  $75\ \mu\text{A}$  from its supply, it has very low self-heating, less than  $0.2^\circ\text{F}$  in still air. The LM34 is rated to operate over a  $-50$  to  $+300^\circ\text{F}$  temperature range, while the LM34C is rated for a  $-40$  to  $+230^\circ\text{F}$  range ( $0^\circ\text{F}$  with improved accuracy). The LM34 series is available packaged in

hermetic TO-46 transistor packages, while the LM34C, LM34CA and LM34D are also available in the plastic TO-92 transistor package. The LM34D is also available in an 8-lead surface mount small outline package. The LM34 is a complement to the LM35 (Centigrade) temperature sensor.

### Features

- Calibrated directly in degrees Fahrenheit
- Linear  $+10.0\ \text{mV}/^\circ\text{F}$  scale factor
- $1.0^\circ\text{F}$  accuracy guaranteed (at  $+77^\circ\text{F}$ )
- Rated for full  $-50$  to  $+300^\circ\text{F}$  range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 5 to 30 volts
- Less than  $90\ \mu\text{A}$  current drain
- Low self-heating,  $0.18^\circ\text{F}$  in still air
- Nonlinearity only  $\pm 0.5^\circ\text{F}$  typical
- Low-impedance output,  $0.4\ \Omega$  for 1 mA load

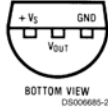
### Connection Diagrams

**TO-46  
Metal Can Package  
(Note 1)**



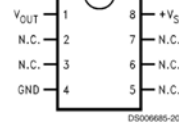
Order Numbers LM34H,  
LM34AH, LM34CH,  
LM34CAH or LM34DH  
See NS Package  
Number H03H

**TO-92  
Plastic Package**



Order Number LM34CZ,  
LM34CAZ or LM34DZ  
See NS Package  
Number Z03A

**SO-8  
Small Outline  
Molded Package**



N.C. = No Connection

Top View  
Order Number LM34DM  
See NS Package Number M08A

Note 1: Case is connected to negative pin (GND).

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

© 1999 National Semiconductor Corporation DS006685

www.national.com

LM34 Precision Fahrenheit Temperature Sensors

Typical Applications

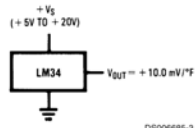


FIGURE 1. Basic Fahrenheit Temperature Sensor  
(+5° to +300°F)

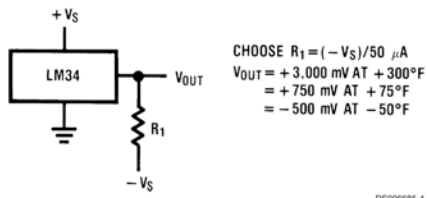


FIGURE 2. Full-Range Fahrenheit Temperature Sensor

<b>Absolute Maximum Ratings</b> (Note 11)					
If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.		TO-46 Package (Soldering, 10 seconds)			+300°C
Supply Voltage		+35V to -0.2V		TO-92 Package (Soldering, 10 seconds)	+260°C
Output Voltage		+6V to -1.0V		SO Package (Note 13)	
Output Current		10 mA		Vapor Phase (60 seconds)	215°C
Storage Temperature,				Infrared (15 seconds)	220°C
TO-46 Package		-76°F to +356°F		Specified Operating Temp. Range (Note 3)	
TO-92 Package		-76°F to +300°F			<b>T<sub>MIN</sub> to T<sub>MAX</sub></b>
SO-8 Package		-65°C to +150°C		LM34, LM34A	-50°F to +300°F
ESD Susceptibility (Note 12)		800V		LM34C, LM34CA	-40°F to +230°F
Lead Temp.				LM34D	+32°F to +212°F

<b>DC Electrical Characteristics</b> (Notes 2, 7)								
Parameter	Conditions	LM34A			LM34CA			Units (Max)
		Typical	Tested Limit (Note 5)	Design Limit (Note 6)	Typical	Tested Limit (Note 5)	Design Limit (Note 6)	
Accuracy (Note 8)	T <sub>A</sub> = +77°F	±0.4	±1.0		±0.4	±1.0		°F
	T <sub>A</sub> = 0°F	±0.6			±0.6		±2.0	°F
	T <sub>A</sub> = T <sub>MAX</sub>	±0.8	±2.0		±0.8	±2.0		°F
	T <sub>A</sub> = T <sub>MIN</sub>	±0.8	±2.0		±0.8		±3.0	°F
Nonlinearity (Note 9)	T <sub>MIN</sub> ≤ T <sub>A</sub> ≤ T <sub>MAX</sub>	<b>±0.35</b>		<b>±0.7</b>	<b>±0.30</b>		<b>±0.6</b>	°F
Sensor Gain (Average Slope)	T <sub>MIN</sub> ≤ T <sub>A</sub> ≤ T <sub>MAX</sub>	<b>+10.0</b>	<b>+9.9, +10.1</b>		<b>+10.0</b>		<b>+9.9, +10.1</b>	mV/°F, min mV/°F, max
Load Regulation (Note 4)	T <sub>A</sub> = +77°F	±0.4	±1.0		±0.4	±1.0		mV/mA
	T <sub>MIN</sub> ≤ T <sub>A</sub> ≤ T <sub>MAX</sub> 0 ≤ I <sub>L</sub> ≤ 1 mA	<b>±0.5</b>		<b>±3.0</b>	<b>±0.5</b>		<b>±3.0</b>	mV/mA
Line Regulation (Note 4)	T <sub>A</sub> = +77°F	±0.01	±0.05		±0.01	±0.05		mV/V
	5V ≤ V <sub>S</sub> ≤ 30V	<b>±0.02</b>		<b>±0.1</b>	<b>±0.02</b>		<b>±0.1</b>	mV/V
Quiescent Current (Note 10)	V <sub>S</sub> = +5V, +77°F	75	90		75	90		µA
	V <sub>S</sub> = +5V	<b>131</b>		<b>160</b>	<b>116</b>		<b>139</b>	µA
	V <sub>S</sub> = +30V, +77°F	76	92		76	92		µA
	V <sub>S</sub> = +30V	<b>132</b>		<b>163</b>	<b>117</b>		<b>142</b>	µA
Change of Quiescent Current (Note 4)	4V ≤ V <sub>S</sub> ≤ 30V, +77°F	+0.5	2.0		0.5	2.0		µA
	5V ≤ V <sub>S</sub> ≤ 30V	<b>+1.0</b>		<b>3.0</b>	<b>1.0</b>		<b>3.0</b>	µA
Temperature Coefficient of Quiescent Current		<b>+0.30</b>		<b>+0.5</b>	<b>+0.30</b>		<b>+0.5</b>	µA/°F
Minimum Temperature for Rated Accuracy	In circuit of Figure 1, I <sub>L</sub> = 0	+3.0		+5.0	+3.0		+5.0	°F
Long-Term Stability	T <sub>J</sub> = T <sub>MAX</sub> for 1000 hours	±0.16			±0.16			°F

**Note 2:** Unless otherwise noted, these specifications apply: -50°F ≤ T<sub>J</sub> ≤ +300°F for the LM34 and LM34A; -40°F ≤ T<sub>J</sub> ≤ +230°F for the LM34C and LM34CA; and +32°F ≤ T<sub>J</sub> ≤ +212°F for the LM34D. V<sub>S</sub> = +5 Vdc and I<sub>LOAD</sub> = 50 µA in the circuit of Figure 2; +6 Vdc for LM34 and LM34A for 230°F ≤ T<sub>J</sub> ≤ 300°F. These specifications also apply from +5°F to T<sub>MAX</sub> in the circuit of Figure 1.

**Note 3:** Thermal resistance of the TO-46 package is 720°F/W junction to ambient and 43°F/W junction to case. Thermal resistance of the TO-92 package is 324°F/W junction to ambient. Thermal resistance of the small outline molded package is 400°F/W junction to ambient. For additional thermal resistance information see table in the Typical Applications section.

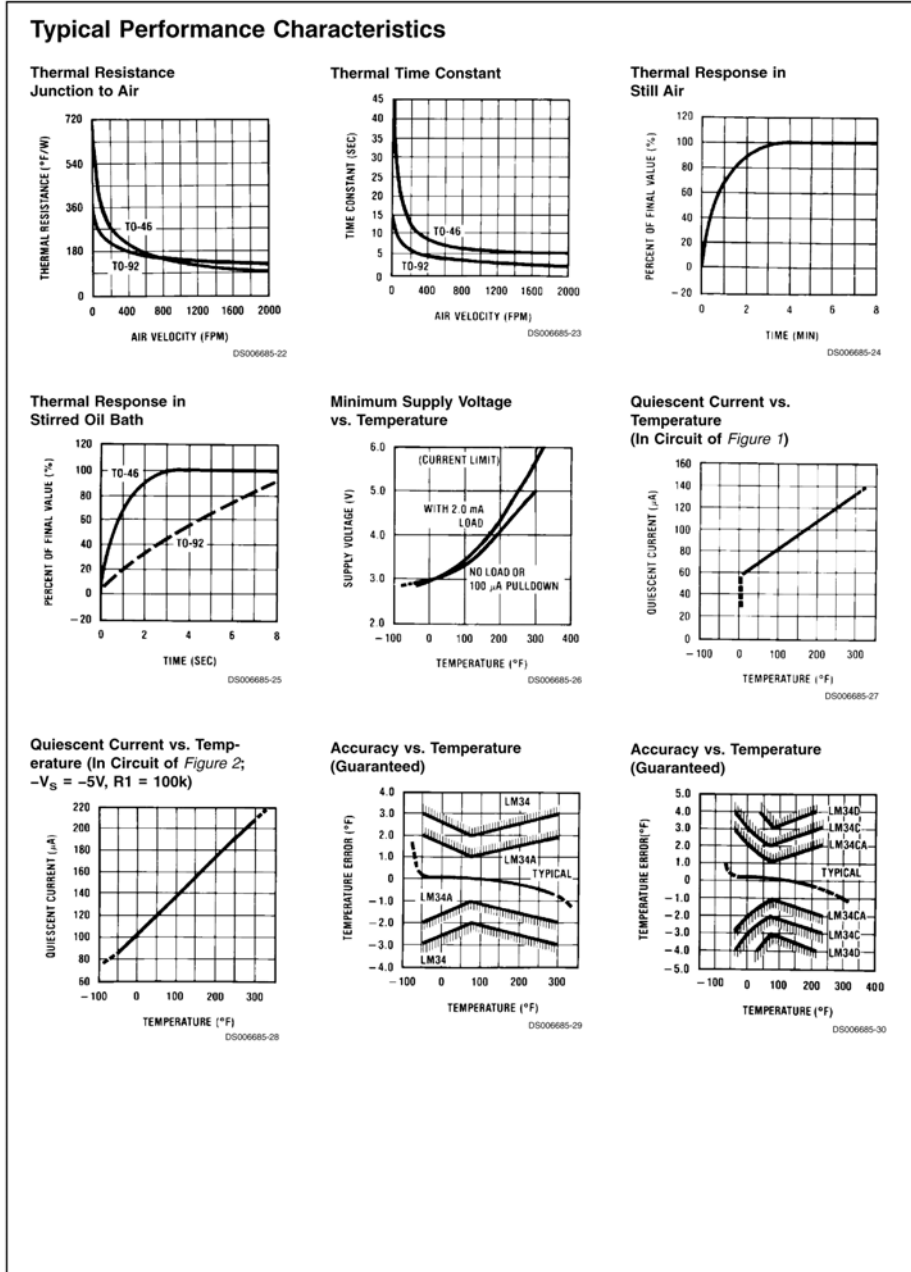
**Note 4:** Regulation is measured at constant junction temperature using pulse testing with a low duty cycle. Changes in output due to heating effects can be computed by multiplying the internal dissipation by the thermal resistance.

**Note 5:** Tested limits are guaranteed and 100% tested in production.

**Note 6:** Design limits are guaranteed (but not 100% production tested) over the indicated temperature and supply voltage ranges. These limits are not used to calculate outgoing quality levels.

**Note 7:** Specification in **BOLDFACE TYPE** apply over the full rated temperature range.

DC Electrical Characteristics (Notes 2, 7) (Continued)								
<p><b>Note 8:</b> Accuracy is defined as the error between the output voltage and 10 mV/°F times the device's case temperature at specified conditions of voltage, current, and temperature (expressed in °F).</p> <p><b>Note 9:</b> Nonlinearity is defined as the deviation of the output-voltage-versus-temperature curve from the best-fit straight line over the device's rated temperature range.</p> <p><b>Note 10:</b> Quiescent current is defined in the circuit of <i>Figure 1</i>.</p> <p><b>Note 11:</b> Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions (Note 2).</p> <p><b>Note 12:</b> Human body model, 100 pF discharged through a 1.5 kΩ resistor.</p> <p><b>Note 13:</b> See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" or the section titled "Surface Mount" found in a current National Semiconductor Linear Data Book for other methods of soldering surface mount devices.</p>								
DC Electrical Characteristics (Notes 2, 7)								
Parameter	Conditions	LM34			LM34C, LM34D			Units (Max)
		Typical	Tested Limit (Note 5)	Design Limit (Note 6)	Typical	Tested Limit (Note 5)	Design Limit (Note 6)	
Accuracy, LM34, LM34C (Note 8)	$T_A = +77^\circ\text{F}$	±0.8	±2.0		±0.8	±2.0		°F
	$T_A = 0^\circ\text{F}$	±1.0			±1.0		±3.0	°F
	$T_A = T_{\text{MAX}}$	±1.6	±3.0		±1.6		±3.0	°F
	$T_A = T_{\text{MIN}}$	±1.6		±3.0	±1.6		±4.0	°F
Accuracy, LM34D (Note 8)	$T_A = +77^\circ\text{F}$				±1.2	±3.0		°F
	$T_A = T_{\text{MAX}}$				±1.8		±4.0	°F
	$T_A = T_{\text{MIN}}$				±1.8		±4.0	°F
Nonlinearity (Note 9)	$T_{\text{MIN}} \leq T_A \leq T_{\text{MAX}}$	±0.6		±1.0	±0.4		±1.0	°F
Sensor Gain (Average Slope)	$T_{\text{MIN}} \leq T_A \leq T_{\text{MAX}}$	+10.0	+9.8, +10.2		+10.0		+9.8, +10.2	mV/°F, min mV/°F, max
Load Regulation (Note 4)	$T_A = +77^\circ\text{F}$	±0.4	±2.5		±0.4	±2.5		mV/mA
	$T_{\text{MIN}} \leq T_A \leq +150^\circ\text{F}$ $0 \leq I_L \leq 1 \text{ mA}$	±0.5		±6.0	±0.5		±6.0	mV/mA
Line Regulation (Note 4)	$T_A = +77^\circ\text{F}$	±0.01	±0.1		±0.01	±0.1		mV/V
	$5\text{V} \leq V_S \leq 30\text{V}$	±0.02		±0.2	±0.02		±0.2	mV/V
Quiescent Current (Note 10)	$V_S = +5\text{V}, +77^\circ\text{F}$	75	100		75	100		μA
	$V_S = +5\text{V}$	131		176	116		154	μA
	$V_S = +30\text{V}, +77^\circ\text{F}$	76	103		76	103		μA
	$V_S = +30\text{V}$	132		181	117		159	μA
Change of Quiescent Current (Note 4)	$4\text{V} \leq V_S \leq 30\text{V}, +77^\circ\text{F}$	+0.5	3.0		0.5	3.0		μA
	$5\text{V} \leq V_S \leq 30\text{V}$	+1.0		5.0	1.0		5.0	μA
Temperature Coefficient of Quiescent Current		+0.30		+0.7	+0.30		+0.7	μA/°F
Minimum Temperature for Rated Accuracy	In circuit of <i>Figure 1</i> , $I_L = 0$	+3.0		+5.0	+3.0		+5.0	°F
Long-Term Stability	$T_j = T_{\text{MAX}}$ for 1000 hours	±0.16			±0.16			°F





Appendix E:  
National Semiconductor  
LM358 Datasheet

Appendix E consists of the National Semiconductor LM358 datasheet. This appendix includes the first five (5) pages of the 23- page datasheet. Should you wish to see more applications of the LM358 than are shown in this datasheet, the entire document may be downloaded from <http://www.national.com/ds/LM/LM158.pdf>.



January 2000

## LM158/LM258/LM358/LM2904 Low Power Dual Operational Amplifiers

### General Description

The LM158 series consists of two independent, high gain, internally frequency compensated operational amplifiers which were designed specifically to operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

Application areas include transducer amplifiers, dc gain blocks and all the conventional op amp circuits which now can be more easily implemented in single power supply systems. For example, the LM158 series can be directly operated off of the standard +5V power supply voltage which is used in digital systems and will easily provide the required interface electronics without requiring the additional  $\pm 15V$  power supplies.

The LM358 is also available in a chip sized package (8-Bump micro SMD) using National's micro SMD package technology.

### Unique Characteristics

- In the linear mode the input common-mode voltage range includes ground and the output voltage can also swing to ground, even though operated from only a single power supply voltage.
- The unity gain cross frequency is temperature compensated.
- The input bias current is also temperature compensated.

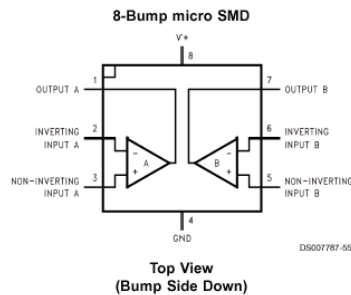
### Advantages

- Two internally compensated op amps
- Eliminates need for dual supplies
- Allows direct sensing near GND and  $V_{OUT}$  also goes to GND
- Compatible with all forms of logic
- Power drain suitable for battery operation
- Pin-out same as LM1558/LM1458 dual op amp

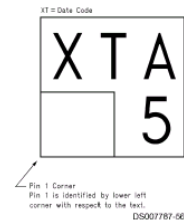
### Features

- Available in 8-Bump micro SMD chip sized package, (See AN-1112)
- Internally frequency compensated for unity gain
- Large dc voltage gain: 100 dB
- Wide bandwidth (unity gain): 1 MHz (temperature compensated)
- Wide power supply range:
  - Single supply: 3V to 32V
  - or dual supplies:  $\pm 1.5V$  to  $\pm 16V$
- Very low supply current drain (500  $\mu A$ )—essentially independent of supply voltage
- Low input offset voltage: 2 mV
- Input common-mode voltage range includes ground
- Differential input voltage range equal to the power supply voltage
- Large output voltage swing: 0V to  $V^+ - 1.5V$

### Connection Diagrams



### micro SMD Marking Orientation

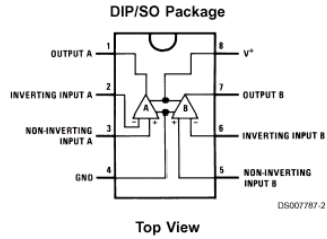
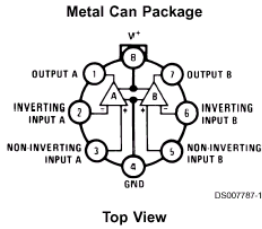


Bumps are numbered counter-clockwise.



LM158/LM258/LM358/LM2904

Connection Diagrams (Continued)



Ordering Information

Package	Temperature Range				NSC Drawing
	-55°C to 125°C	-25°C to 85°C	0°C to 70°C	-40°C to 85°C	
SO-8			LM358AM LM358M	LM2904M	M08A
8-Pin Molded DIP			LM358AN LM358N	LM2904N	N08E
8-Pin Ceramic DIP	LM158AJ/883(Note 1) LM158J/883(Note 1) LM158J LM158AJLQML(Note 2) LM158AJQMLV(Note 2)				J08A
TO-5, 8-Pin Metal Can	LM158AH/883(Note 1) LM158H/883(Note 1) LM158AH LM158H LM158AHLQML(Note 2) LM158AHLQMLV(Note 2)	LM258H	LM358H		H08C
8-Bump micro SMD			LM358BP LM358BPX		BPA08AAA

**Note 1:** LM158 is available per SMD #5962-8771001  
LM158A is available per SMD #5962-8771002  
**Note 2:** See STD Mil DWG 5962L87710 for Radiation Tolerant Devices

**Absolute Maximum Ratings** (Note 11)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

	LM158/LM258/LM358 LM158A/LM258A/LM358A	LM2904
Supply Voltage, $V^*$	32V	26V
Differential Input Voltage	32V	26V
Input Voltage	-0.3V to +32V	-0.3V to +26V
Power Dissipation (Note 3)		
Molded DIP	830 mW	830 mW
Metal Can	550 mW	
Small Outline Package (M) micro SMD	530 mW 435mW	530 mW
Output Short-Circuit to GND (One Amplifier) (Note 4) $V^* \leq 15V$ and $T_A = 25^\circ C$	Continuous	Continuous
Input Current ( $V_{IN} < -0.3V$ ) (Note 5)	50 mA	50 mA
Operating Temperature Range		
LM358	0°C to +70°C	-40°C to +85°C
LM258	-25°C to +85°C	
LM158	-55°C to +125°C	
Storage Temperature Range	-65°C to +150°C	-65°C to +150°C
Lead Temperature, DIP (Soldering, 10 seconds)	260°C	260°C
Lead Temperature, Metal Can (Soldering, 10 seconds)	300°C	300°C
Soldering Information		
Dual-In-Line Package Soldering (10 seconds)	260°C	260°C
Small Outline Package Vapor Phase (60 seconds) Infrared (15 seconds)	215°C 220°C	215°C 220°C
See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" for other methods of soldering surface mount devices.		
ESD Tolerance (Note 12)	250V	250V

**Electrical Characteristics**

$V^* = +5.0V$ , unless otherwise stated

Parameter	Conditions	LM158A			LM358A			LM158/LM258			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	(Note 7), $T_A = 25^\circ C$	1	2		2	3		2	5		mV
Input Bias Current	$I_{IN(+)}$ or $I_{IN(-)}$ , $T_A = 25^\circ C$ , $V_{CM} = 0V$ , (Note 8)	20	50		45	100		45	150		nA
Input Offset Current	$I_{IN(+)} - I_{IN(-)}$ , $V_{CM} = 0V$ , $T_A = 25^\circ C$	2	10		5	30		3	30		nA
Input Common-Mode Voltage Range	$V^* = 30V$ , (Note 9) (LM2904, $V^* = 26V$ ), $T_A = 25^\circ C$	0	$V^*-1.5$		0	$V^*-1.5$		0	$V^*-1.5$		V
Supply Current	Over Full Temperature Range $R_L = \infty$ on All Op Amps $V^* = 30V$ (LM2904 $V^* = 26V$ ) $V^* = 5V$	1	2		1	2		1	2		mA
		0.5	1.2		0.5	1.2		0.5	1.2		mA

LM158/LM258/LM358/LM2904

Electrical Characteristics								
V <sup>+</sup> = +5.0V, unless otherwise stated								
Parameter	Conditions	LM358			LM2904			Units
		Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	(Note 7), T <sub>A</sub> = 25°C		2	7		2	7	mV
Input Bias Current	I <sub>IN(+)</sub> or I <sub>IN(-)</sub> , T <sub>A</sub> = 25°C, V <sub>CM</sub> = 0V, (Note 8)		45	250		45	250	nA
Input Offset Current	I <sub>IN(+)</sub> - I <sub>IN(-)</sub> , V <sub>CM</sub> = 0V, T <sub>A</sub> = 25°C		5	50		5	50	nA
Input Common-Mode Voltage Range	V <sup>+</sup> = 30V, (Note 9) (LM2904, V <sup>+</sup> = 26V), T <sub>A</sub> = 25°C	0		V <sup>+</sup> -1.5	0		V <sup>+</sup> -1.5	V
Supply Current	Over Full Temperature Range R <sub>L</sub> = ∞ on All Op Amps V <sup>+</sup> = 30V (LM2904 V <sup>+</sup> = 26V) V <sup>+</sup> = 5V		1	2		1	2	mA
			0.5	1.2		0.5	1.2	mA

Electrical Characteristics											
V <sup>+</sup> = +5.0V, (Note 6), unless otherwise stated											
Parameter	Conditions	LM158A			LM358A			LM158/LM258			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Large Signal Voltage Gain	V <sup>+</sup> = 15V, T <sub>A</sub> = 25°C, R <sub>L</sub> ≥ 2 kΩ, (For V <sub>O</sub> = 1V to 11V)	50	100		25	100		50	100		V/mV
Common-Mode Rejection Ratio	T <sub>A</sub> = 25°C, V <sub>CM</sub> = 0V to V <sup>+</sup> -1.5V	70	85		65	85		70	85		dB
Power Supply Rejection Ratio	V <sup>+</sup> = 5V to 30V (LM2904, V <sup>+</sup> = 5V to 26V), T <sub>A</sub> = 25°C	65	100		65	100		65	100		dB
Amplifier-to-Amplifier Coupling	f = 1 kHz to 20 kHz, T <sub>A</sub> = 25°C (Input Referred), (Note 10)	-120			-120			-120			dB
Output Current	Source V <sub>IN+</sub> = 1V, V <sub>IN-</sub> = 0V, V <sup>+</sup> = 15V, V <sub>O</sub> = 2V, T <sub>A</sub> = 25°C	20	40		20	40		20	40		mA
	Sink V <sub>IN-</sub> = 1V, V <sub>IN+</sub> = 0V, V <sup>+</sup> = 15V, T <sub>A</sub> = 25°C, V <sub>O</sub> = 2V	10	20		10	20		10	20		mA
	V <sub>IN-</sub> = 1V, V <sub>IN+</sub> = 0V, T <sub>A</sub> = 25°C, V <sub>O</sub> = 200 mV, V <sup>+</sup> = 15V	12	50		12	50		12	50		μA
Short Circuit to Ground	T <sub>A</sub> = 25°C, (Note 4), V <sup>+</sup> = 15V	40	60		40	60		40	60		mA
Input Offset Voltage	(Note 7)		4			5			7		mV
Input Offset Voltage Drift	R <sub>S</sub> = 0Ω	7	15		7	20		7			μV/°C
Input Offset Current	I <sub>IN(+)</sub> - I <sub>IN(-)</sub>		30			75			100		nA
Input Offset Current Drift	R <sub>S</sub> = 0Ω	10	200		10	300		10			pA/°C
Input Bias Current	I <sub>IN(+)</sub> or I <sub>IN(-)</sub>	40	100		40	200		40	300		nA
Input Common-Mode Voltage Range	V <sup>+</sup> = 30V, (Note 9) (LM2904, V <sup>+</sup> = 26V)	0		V <sup>+</sup> -2	0		V <sup>+</sup> -2	0		V <sup>+</sup> -2	V

Electrical Characteristics (Continued)											
V* = +5.0V, (Note 6), unless otherwise stated											
Parameter	Conditions	LM158A			LM358A			LM158/LM258			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Large Signal Voltage Gain	V* = +15V (V <sub>O</sub> = 1V to 11V) R <sub>L</sub> ≥ 2 kΩ	25			15			25			V/mV
Output Voltage Swing	V <sub>OH</sub>	26			26			26			V
	V <sub>OL</sub>	5 20			5 20			5 20			mV
Output Current	Source	10 20			10 20			10 20			mA
	Sink	10 15			5 8			5 8			mA
Electrical Characteristics											
V* = +5.0V, (Note 6), unless otherwise stated											
Parameter	Conditions	LM358			LM2904			Units			
		Min	Typ	Max	Min	Typ	Max				
Large Signal Voltage Gain	V* = 15V, T <sub>A</sub> = 25°C, R <sub>L</sub> ≥ 2 kΩ, (For V <sub>O</sub> = 1V to 11V)	25 100			25 100			V/mV			
Common-Mode Rejection Ratio	T <sub>A</sub> = 25°C, V <sub>CM</sub> = 0V to V* - 1.5V	65 85			50 70			dB			
Power Supply Rejection Ratio	V* = 5V to 30V (LM2904, V* = 5V to 26V), T <sub>A</sub> = 25°C	65 100			50 100			dB			
Amplifier-to-Amplifier Coupling	f = 1 kHz to 20 kHz, T <sub>A</sub> = 25°C (Input Referred), (Note 10)	-120			-120			dB			
Output Current	Source	20 40			20 40			mA			
	Sink	10 20			10 20			mA			
		12 50			12 50			μA			
Short Circuit to Ground	T <sub>A</sub> = 25°C, (Note 4), V* = 15V	40 60			40 60			mA			
Input Offset Voltage	(Note 7)	9			10			mV			
Input Offset Voltage Drift	R <sub>S</sub> = 0Ω	7			7			μV/°C			
Input Offset Current	I <sub>IN(+)</sub> - I <sub>IN(-)</sub>	150			45 200			nA			
Input Offset Current Drift	R <sub>S</sub> = 0Ω	10			10			pA/°C			
Input Bias Current	I <sub>IN(+)</sub> or I <sub>IN(-)</sub>	40 500			40 500			nA			
Input Common-Mode Voltage Range	V* = 30 V, (Note 9) (LM2904, V* = 26V)	0			V* - 2			V			



Appendix F:  
Dallas Semiconductor  
1302 Datasheet

Appendix F consists of the Dallas Semiconductor 1302 datasheet. This appendix includes the first five (5) pages of the 23-page datasheet. Should you wish to see more applications of the DS1302 than are shown in this datasheet, the Parallax web site includes AppKit documentation for this part.



## DS1302 Trickle Charge Timekeeping Chip

[www.maxim-ic.com](http://www.maxim-ic.com)

### FEATURES

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap-year compensation valid up to 2100
- 31-byte, battery-backed, nonvolatile (NV) RAM for data storage
- Serial I/O for minimum pin count
- 2.0V to 5.5V full operation
- Uses less than 300nA at 2.0V
- Burst mode for reading/writing successive addresses in clock/RAM
- 8-pin DIP or optional 8-pin SOICs for surface mount
- Simple 3-wire interface
- TTL-compatible ( $V_{CC} = 5V$ )
- Optional industrial temperature range:  $-40^{\circ}C$  to  $+85^{\circ}C$
- DS1202 compatible
- Underwriters Laboratory (UL) recognized

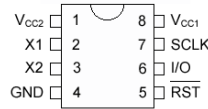
### ORDERING INFORMATION

DS1302	8-Pin DIP (300-mil)
DS1302N	8-Pin DIP (Industrial)
DS1302S	8-Pin SOIC (200-mil)
DS1302SN	8-Pin SOIC (Industrial)
DS1302Z	8-Pin SOIC (150-mil)
DS1302ZN	8-Pin SOIC (Industrial)
DS1302S-16	16-Pin SOIC (300-mil)
DS1302SN-16	16-Pin SOIC (Industrial)

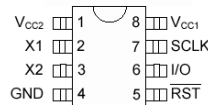
### DESCRIPTION

The DS1302 Trickle Charge Timekeeping Chip contains an RTC/calendar and 31 bytes of static RAM. It communicates with a microprocessor via a simple serial interface. The RTC/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator.

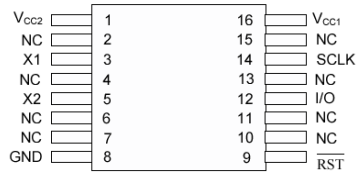
### PIN ASSIGNMENT



DS1302 8-Pin DIP (300-mil)



DS1302 8-Pin SOIC (200-mil)  
DS1302 8-Pin SOIC (150-mil)



DS1302 16-Pin SOIC (300-mil)

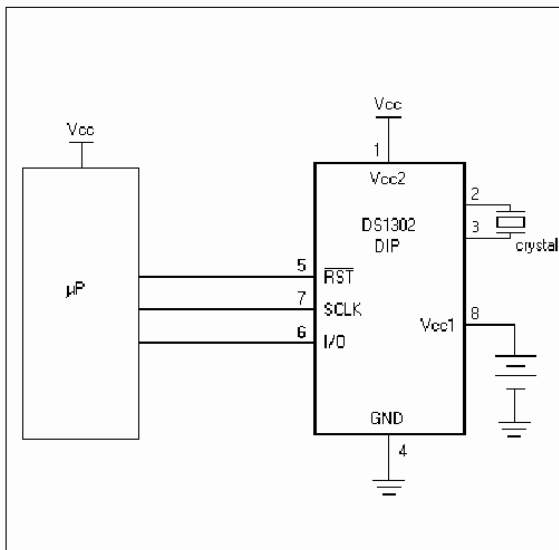
### PIN DESCRIPTION

X1, X2	- 32.768kHz Crystal Pins
GND	- Ground
RST	- Reset
I/O	- Data Input/Output
SCLK	- Serial Clock
$V_{CC1}, V_{CC2}$	- Power Supply Pins

## DS1302

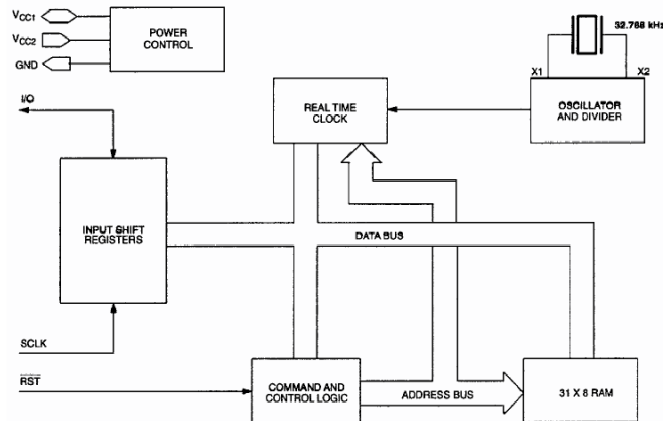
Interfacing the DS1302 with a microprocessor is simplified by using synchronous serial communication. Only three wires are required to communicate with the clock/RAM: 1)  $\overline{\text{RST}}$  (reset), 2) I/O (data line), and 3) SCLK (serial clock). Data can be transferred to and from the clock/RAM 1 byte at a time or in a burst of up to 31 bytes. The DS1302 is designed to operate on very low power and retain data and clock information on less than 1 microwatt.

The DS1302 is the successor to the DS1202. In addition to the basic timekeeping functions of the DS1202, the DS1302 has the additional features of dual-power pins for primary and back-up power supplies, programmable trickle charger for  $V_{CC1}$ , and seven additional bytes of scratchpad memory.

**TYPICAL OPERATING CIRCUIT**

**OPERATION**

The main elements of the serial timekeeper (i.e., shift register, control logic, oscillator, RTC, and RAM) are shown in Figure 1.

**DS1302 BLOCK DIAGRAM Figure 1****SIGNAL DESCRIPTIONS**

$V_{CC1}$  –  $V_{CC1}$  provides low-power operation in single supply and battery-operated systems as well as low-power battery backup. In systems using the trickle charger, the rechargeable energy source is connected to this pin. UL recognized to ensure against reverse charging current when used in conjunction with a lithium battery.

See “Conditions of Acceptability” at <http://www.maxim-ic.com/TechSupport/QA/ntrl.htm>.

$V_{CC2}$  –  $V_{CC2}$  is the primary power supply pin in a dual-supply configuration.  $V_{CC1}$  is connected to a backup source to maintain the time and date in the absence of primary power.

The DS1302 will operate from the larger of  $V_{CC1}$  or  $V_{CC2}$ . When  $V_{CC2}$  is greater than  $V_{CC1} + 0.2V$ ,  $V_{CC2}$  will power the DS1302. When  $V_{CC2}$  is less than  $V_{CC1}$ ,  $V_{CC1}$  will power the DS1302.

**SCLK (Serial Clock Input)** – SCLK is used to synchronize data movement on the serial interface. This pin has a 40k $\Omega$  internal pull-down resistor.

**I/O (Data Input/Output)** – The I/O pin is the bi-directional data pin for the 3-wire interface. This pin has a 40k $\Omega$  internal pull-down resistor.

**RST (Reset)** – The reset signal must be asserted high during a read or a write. This pin has a 40k $\Omega$  internal pull-down resistor.

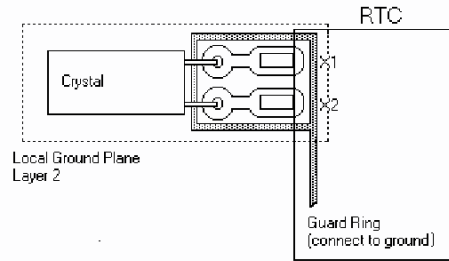
**X1, X2** – Connections for a standard 32.768kHz quartz crystal. The internal oscillator is designed for operation with a crystal having a specified load capacitance of 6pF. For more information on crystal selection and crystal layout considerations, please consult Application Note 58, “Crystal Considerations”.



DS1302

with Dallas Real-Time Clocks." The DS1302 can also be driven by an external 32.768kHz oscillator. In this configuration, the X1 pin is connected to the external oscillator signal and the X2 pin is floated.

**RECOMMENDED LAYOUT FOR CRYSTAL**



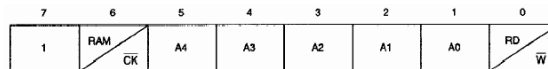
**CLOCK ACCURACY**

The accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacitive load of the oscillator circuit and the capacitive load for which the crystal was trimmed. Additional error will be added by crystal frequency drift caused by temperature shifts. External circuit noise coupled into the oscillator circuit may result in the clock running fast. See Application Note 58, "Crystal Considerations with Dallas Real-Time Clocks" for detailed information.

**COMMAND BYTE**

The command byte is shown in Figure 2. Each data transfer is initiated by a command byte. The MSB (Bit 7) must be a logic 1. If it is 0, writes to the DS1302 will be disabled. Bit 6 specifies clock/calendar data if logic 0 or RAM data if logic 1. Bits 1 through 5 specify the designated registers to be input or output, and the LSB (bit 0) specifies a write operation (input) if logic 0 or read operation (output) if logic 1. The command byte is always input starting with the LSB (bit 0).

**ADDRESS/COMMAND BYTE Figure 2**



**RESET AND CLOCK CONTROL**

All data transfers are initiated by driving the  $\overline{RST}$  input high. The  $\overline{RST}$  input serves two functions. First,  $\overline{RST}$  turns on the control logic, which allows access to the shift register for the address/command sequence. Second, the  $\overline{RST}$  signal provides a method of terminating either single byte or multiple byte data transfer.

A clock cycle is a sequence of a falling edge followed by a rising edge. For data inputs, data must be valid during the rising edge of the clock and data bits are output on the falling edge of clock. If the  $\overline{RST}$  input is low all data transfer terminates and the I/O pin goes to a high impedance state. Data transfer is illustrated in Figure 3. At power-up,  $\overline{RST}$  must be a logic 0 until  $V_{CC} > 2.0V$ . Also SCLK must be at a logic 0 when  $\overline{RST}$  is driven to a logic 1 state.

**DATA INPUT**

Following the eight SCLK cycles that input a write command byte, a data byte is input on the rising edge of the next eight SCLK cycles. Additional SCLK cycles are ignored should they inadvertently occur. Data is input starting with bit 0.

**DATA OUTPUT**

Following the eight SCLK cycles that input a read command byte, a data byte is output on the falling edge of the next eight SCLK cycles. Note that the first data bit to be transmitted occurs on the first falling edge after the last bit of the command byte is written. Additional SCLK cycles retransmit the data bytes should they inadvertently occur so long as  $\overline{\text{RST}}$  remains high. This operation permits continuous burst mode read capability. Also, the I/O pin is tri-stated upon each rising edge of SCLK. Data is output starting with bit 0.

**BURST MODE**

Burst mode may be specified for either the clock/calendar or the RAM registers by addressing location 31 decimal (address/command bits 1 through 5 = logic 1). As before, bit 6 specifies clock or RAM and bit 0 specifies read or write. There is no data storage capacity at locations 9 through 31 in the Clock/Calendar Registers or location 31 in the RAM registers. Reads or writes in burst mode start with bit 0 of address 0.

When writing to the clock registers in the burst mode, the first eight registers must be written in order for the data to be transferred. However, when writing to RAM in burst mode it is not necessary to write all 31 bytes for the data to transfer. Each byte that is written to will be transferred to RAM regardless of whether all 31 bytes are written or not.

**CLOCK/CALENDAR**

The clock/calendar is contained in seven write/read registers as shown in Figure 4. Data contained in the clock/calendar registers is in binary coded decimal format (BCD).

**CLOCK HALT FLAG**

Bit 7 of the seconds register is defined as the clock halt flag. When this bit is set to logic 1, the clock oscillator is stopped and the DS1302 is placed into a low-power standby mode with a current drain of less than 100 nanoamps. When this bit is written to logic 0, the clock will start. The initial power on state is not defined.

**AM-PM/12-24 MODE**

Bit 7 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20–23 hours).

**WRITE PROTECT BIT**

Bit 7 of the control register is the write-protect bit. The first seven bits (bits 0–6) are forced to 0 and will always read a 0 when read. Before any write operation to the clock or RAM, bit 7 must be 0. When high, the write protect bit prevents a write operation to any other register. The initial power on state is not defined. Therefore the WP bit should be cleared before attempting to write to the device.

**TRICKLE CHARGE REGISTER**

This register controls the trickle charge characteristics of the DS1302. The simplified schematic of Figure 5 shows the basic components of the trickle charger. The trickle charge select (TCS) bits (bits 4–7) control the selection of the trickle charger. In order to prevent accidental enabling, only a pattern of 1010 will enable the trickle charger. All other patterns will disable the trickle charger. The



## Appendix G: Parts Listing and Sources

All components (next page) used in the Industrial Control experiments are readily available from common electronic suppliers. Customers who would like to purchase a complete kit may also do so through Parallax. To use this curriculum you need three items: (1) a

BASIC Stamp II module (available alone, or in the Board of Education - Full Kit); (2) a Board of Education (available alone or in a Board of Education Full Kit); and 3) the Industrial Control Parts Kit.

### Board of Education Kits

The BASIC Stamp II (BS2- IC) is available separately or in the Board of Education Full Kit. If you already have a BS2- IC module, then purchase the Board of Education Kit. Individual pieces may also be ordered using the Parallax stock codes shown below.

#### Board of Education – Full Kit (#28102)

Parallax Code#	Description	Quantity
28150	Board of Education	1
800-00016	Pluggable wires	10
BS2- IC	BASIC Stamp II module	1
750-00008	300 mA 9 VDC power supply	1
800-00003	Serial cable	1

#### Board of Education Kit (#28150)

Parallax Code#	Description	Quantity
28102	Board of Education and pluggable wires	1
BS2- IC	Pluggable wires	6

This printed documentation is very useful for additional background information:

#### BASIC Stamp Documentation

Parallax Code#	Description	Internet Availability?
27949	BASIC Stamp Manual Version 2.0	<a href="http://www.stampsinclass.com">http://www.stampsinclass.com</a>
27341	Industrial Control Text	<a href="http://www.stampsinclass.com">http://www.stampsinclass.com</a>

## Appendix G: Parts Listing and Sources

---

The Industrial Control experiments require the Industrial Control Parts Kit (#27340)

Similar to all Stamps in Class curriculum, you need a Board of Education with BASIC Stamp and the Parts Kit. The contents of the Industrial Control Parts Kit is listed below, broken down by experiment. Replacement parts in the kit may be ordered from <http://www.stampsinclass.com>.

### Industrial Control Parts Kit (#27340)

Stock#	Description	#1	#2	#3	#4	#5	#6	#7	Total/Kit
150-01020	1K ¼watt resistor		1	1					2
150-01030	10K ¼watt resistor	1	2		1			1	2
150-02030	20K ¼watt resistor		1		1	1			1
150-02210	220 ohm ¼watt resistor	1	2	3	1	1	1	1	4
150-04710	470 ohm ¼watt resistor		1		1	1	1	1	4
152-01031	10K ¼watt multi-turn pot	1	2	2	2	2	2	2	2
153-00001	5.1 V .5 W Zener Diode		1		1	1			1
153-00002	BS170 MOSFET			1	1				1
200-06840	.68uF capacitor		1		1	1			1
201-01050	1uF capacitor	1							1
201-01060	10uF 10V capacitor				1	1	1	1	1
251-03230	32.768 kHz Clock Crystal							1	1
350-00001	LED green	1	1	1					1
350-00006	LED red		1	1	1	1	1	1	1
350-00007	LED yellow			1					1
350-00017	IR Led w/ shrink tube		1						1
350-00018	Infrared Phototransistor		1						1
400-00002	Pushbutton	1	2	2	1			1	2
500-00001	2N3904 Transistor		1	1	1	1	1	1	2
604-00005	DS 1302 Clock Chip							1	1
602-00015	LM358 Dual Op- Amp		1		1	1			1

Appendix G: Parts Listing and Sources

700-00039	35 mm Black Film Canister		1		1	1	1	1	1
700-00040	12 VDC Brushless Fan		1	1	1	1	1	1	1
800-00027	LM34 Temperature Probe				1	1	1	1	1
800-00028	47 Ohm Resistor Heater			1	1	1	1	1	1
ADC0831	ADC 0831 8-bit A/D converter				1	1	1	1	1

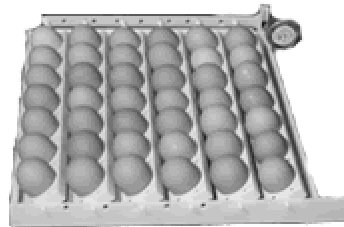




## Appendix H: Commercial Incubator Challenge

Hopefully this text has given you insight into the selection and proper interfacing of industrial field devices.

The film-canister incubator has provided a system for modeling basic process control strategies. The BASIC Stamp is well suited for monitoring and controlling applications ranging from around-the-house hobby circuits to critical industrial process control. A great project for the serious hobbyist or for a school laboratory is to develop a real microcontroller-based poultry and game bird incubator. Tabletop systems are available for as low as \$20.00. GQF Manufacturing of Savanna, Georgia (<http://www.gqfmg.com>) is only one of many manufactures that carry a full line of incubators and equipment. The tabletop Model 2362 pictured below would be an excellent choice. It contains a 25 watt 120 VAC heater and a 120 VAC – 500 mA circulating fan. Add to this the Model 1611 egg turner and you have three 120 VAC outputs to control. The total cost is around \$90.00.



Mfg #2362N

Mfg #1611

## Appendix H: Commercial Incubator Challenge

---

Experiment #3 discusses the interfacing of relays to control high voltage, high power loads. The fan and egg turner are turned on and off infrequently and could be controlled with a properly sized electromechanical relay. The heating element will be cycled quickly and should be controlled by an electronic solid-state relay.

Radio shack carries a variety of suitable relays.

An in-line bi-metal mechanical thermostat usually controls the incubator's heating element. This should be left in place as a "failsafe" cutoff and adjusted to 2 degrees higher than the desired setpoint. It would therefore remove power if your electronic thermostat or solid state relay failed. The LM34/ADC 0831 sensor circuit and current boost heater drive circuit used in Experiments #4 through #7 would work great for our real application. Instead of driving the 47-ohm resistor however you would be driving the solid-state relay. The relay, in turn, controls the high-voltage heater.

Optimum hatching temperature is dependant on the type of eggs. For instance, chicken eggs should be incubated at 101.5 while the best temperature for goose and duck eggs is 100.5. The span of the ADC0831 could be more focused around this temperature and Experiment #4 gave you the insight to do this. Doing so would give better resolution and more accurate data.

Your control strategy for the heater could be based on any of the five modes that were discussed in Experiments #5 and #6. Experimentation and plotting with StampPlot Lite gives you the information needed to analyze and evaluate the pros and cons of each method.

An audible alarm for out-of-range conditions would be a good feature to add to the incubator. Radio Shack's 108 dB Piezo Buzzer (part # 273-057) pictured below would definitely get your attention. And, it could be driven from the BASIC Stamp's power supply. The current boost BS170 MOSFET used in Figure 3.9d would be an effective drive circuit to deliver its 150 mA requirement.



Over temperature and prolonged under-temperature should be alarmed. You would want to add a couple of pushbutton switch inputs to program for alarm reset and deactivation during initial warm-up. Review Experiment #2 for proper interfacing and programming of digital inputs.



A water pan in the incubator ensures that humidity stays high enough to keep the eggs hydrated. How could the BASIC Stamp add intelligence to this aspect of our incubator? Perhaps an LED could flash to remind you to check the water. Or better yet, the moisture sensing lab in the Stamps-in-Class Earth Measurements text could be modified to detect the pan being dry. This condition could flash the lamp or beep the alarm. To automate the process, the output of a humidity sensor could be compared to a voltage reference at the inputs to your LM358. The HI/LOW output of the comparator would be an electronic digital input to the Stamp. When humidity drops below the desired value, turn on a humidifier and pipe in some water vapor.

Adding the DS1302 timekeeping chip introduced in Experiment #7 gives you all of the features of a top-of-line commercial incubator. Temperature and humidity is monitored and controlled. Daily reports of temperature variations can be time-stamped and reported. The eggs can be turned at programmed intervals. Cool down can be scheduled and a 1-degree ramp up for the 24 hours before catching can be programmed. Each Experiments 7's three exercises can be modified to accomplish these tasks. The healthy chicks pictured below are the successful result of maintaining the process for 21 days.



If you dedicate a computer to the system, StampPlot Lite could continually plot and report system conditions. If you really want to get fancy though, StampPlot Pro software adds an enormous amount of computer interface possibilities. Multiple analog channel plotting, full graphic drawing capability, two-way interaction on the fly, and the ability to display .jpg and play .wav files only limits the possibilities to the limit of your imagination. In addition, StampPlot Pro's Internet capability allows you monitor and control the incubator remotely. Stamp Plot Pro is available for evaluation through Parallax at the Stamps in Class web sites.

## Appendix H: Commercial Incubator Challenge

---

As you can see, this commercial incubator application incorporates the concepts covered in every section of this text. It is doubtful that many of you will be stop here and become chicken ranchers. As you continue to experiment with the BASIC Stamp and apply it to your hobbies and in your real industrial application, keep these concepts in mind and this textbook on your shelf as a reference.