Process Control

Student Guide

VERSION 1.0

WARRANTY

Parallax Inc. warrants its products against defects in materials and workmanship for a period of 90 days from receipt of product. If you discover a defect, Parallax Inc. will, at its option, repair or replace the merchandise, or refund the purchase price. Before returning the product to Parallax, call for a Return Merchandise Authorization (RMA) number. Write the RMA number on the outside of the box used to return the merchandise to Parallax. Please enclose the following along with the returned merchandise: your name, telephone number, shipping address, and a description of the problem. Parallax 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 Inc. will refund the purchase price of the product, excluding shipping/handling costs. This guarantee is void if the product has been altered or damaged. See the Warranty section above for instructions on returning a product to Parallax.

COPYRIGHTS AND TRADEMARKS

This documentation is copyright 2006 by Parallax Inc. By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used exclusively with Parallax products. Any other uses are not permitted and may represent a violation of Parallax copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by Parallax Inc. Duplication for educational use is permitted, subject to the following Conditions of Duplication: Parallax Inc. grants the user a conditional right to download, duplicate, and distribute this text without Parallax's permission. This right is based on the following conditions: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with Parallax products, and the user may recover from the student only the cost of duplication.

This text is available in printed format from Parallax Inc. Because we print the text in volume, the consumer price is often less than typical retail duplication charges.

BASIC Stamp, Stamps in Class, Board of Education, Boe-Bot SumoBot, SX-Key and Toddler are registered trademarks of Parallax, Inc. If you decide to use registered trademarks of Parallax Inc. on your web page or in printed material, you must state that "(registered trademark) is a registered trademark of Parallax Inc." upon the first appearance of the trademark name in each printed document or web page. HomeWork Board, Propeller, Parallax, and the Parallax logo are trademarks of Parallax Inc. If you decide to use trademarks of Parallax Inc. on your web page or in printed material, you must state that "(trademark) is a trademark of Parallax Inc.", "upon the first appearance of the trademark name in each printed document or web page. Other brand and product names are trademarks or registered trademarks of their respective holders.

ISBN 1-928982-36-0

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, or any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax Inc. 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 DISCUSSION LISTS

We maintain active web-based discussion forums for people interested in Parallax products. These lists are accessible from www.parallax.com via the Support → Discussion Forums menu. These are the forums that we operate from our web site:

- BASIC Stamp – This list is widely utilized by engineers, hobbyists and students who share their BASIC Stamp projects and ask questions.
- Stamps In Class® – Created for educators and students, subscribers discuss the use of the Stamps in Class educational products in their courses. The list provides an opportunity for both students and educators to ask questions and get answers.
- Parallax Educators –Exclusively for educators and those who contribute to the development of Stamps in Class. Parallax created this group to obtain feedback on our educational products and to provide a forum for educators to develop and obtain Teacher's Guides and other resources.
- Translators – The purpose of this private list is to provide a conduit between Parallax and those who translate our documentation to languages other than English. Parallax provides editable Word documents to our translating partners and attempts to time the translations to coordinate with our publications. To join, email aalvarez@parallax.com.
- Robotics – Designed for Parallax robots, this forum is intended to be an open dialogue for robotics enthusiasts. Topics include assembly, source code, expansion, and manual updates. The Boe-Bot®, Toddler®, SumoBot®, HexCrawler and QuadCrawler robots are discussed here.
- SX Microcontrollers and SX-Key – Discussion of programming the SX microcontroller with Parallax assembly language SX – Key® tools and 3rd party BASIC and C compilers.
- Javelin Stamp – Discussion of application and design using the Javelin Stamp, a Parallax module that is programmed using a subset of Sun Microsystems' Java® programming language.
- ParallaxEFX – For animators, theatre prop builders, and those who create Halloween and other holiday displays using the ParallaxEFX product line.
- Propeller Chip – Forum for those using the Parallax Propeller™ chip.

ERRATA

While great effort is made to assure the accuracy of our texts, errors may still exist. If you find an error, please let us know by sending an email to editor@parallax.com. We continually strive to improve all of our educational materials and documentation, and frequently revise our texts. Occasionally, an errata sheet with a list of known errors and corrections for a given text will be posted to our web site, www.parallax.com. Please check the individual product page's free downloads for an errata file.

Table of Contents

Preface

Industrial process control (PC) 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. In this respect, you may hear that microcontrollers are embedded as the "brains" in much of our manufacturing equipment and consumer electronic devices. But in reality, the real "brains" of the system is the process control technician.

Although embedded process control centers around the microcontroller, it is only one piece of the total control system. The process control technician must be part control engineer, electronics technician, and computer programmer. This Process Control text uses its experiment-based chapters to build a good foundation from which to analyze and understand the many facets of embedded control technology.

The text builds this foundation through hands-on laboratory circuits and experiments that reinforce short, relative discussions of control theory. You will experiment with event-based and time-based sequential control as well as various open-loop and closed-loop continuous control modes. You will understand the characteristics of these modes of control and how they lend themselves to different types of control applications. Converting the control scenario and the mode of control chosen into a program flowchart is the first major step toward bringing automated intelligence into the system. Clear, well-commented PBASIC programs demonstrate how the Basic Stamp can be programmed to provide the control action.

An exciting and powerful software application comes with this text to help you visually understand the dynamics of a system as well as allow you to develop computer-based monitoring and control of your Basic Stamp. StampPlot's multiple-channel graphing feature is used throughout the text to allow you to monitor and compare input and output relationships to better understand the dynamics of the control system. You will also see how virtual controls, such as gauges, pushbuttons, sliders, textboxes, etc. can be used to build interactive visual interfaces for supervisory control and data acquisition of your Basic Stamp projects.

The hardware needed in the experiments to simulate the process has been kept to a bare minimum. While the microcontroller is programmed to be 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. The experiments include information on proper I/O signal conditioning. The process control technician must have a good understanding of the electronics required to get proper input voltages into the microcontroller from switches, contacts, and sensors as well as understand how to interface it to high-power output elements through the use of relays and power semiconductors.

After working with the sample control scenarios in the book, students quickly find themselves considering the countless automated control applications all around them. The most exciting aspect of this Process Control text is its ability to give you the tools to apply control theory, flowchart diagramming, and input/output signal conditioning to your own real-world applications.

Martin Hebel and Will Devenport
Southern Illinois University Carbondale
Electronic Systems Technologies
http://www.siu.edu/~isat/est
-- and -- SelmaWare Solutions
http://www.selmaware.com

Editor's Note: *Process Control* is a newly written text covering similar subject matter to Industrial *Control*, which it now replaces in the Stamps In Class educational series. *Process Control* is an advanced book, and we strongly recommend that students first learn the electronics and PBASIC programming concepts introduced in *What's a Microcontroller?* – the gateway to the Stamps in Class series.

EDUCATOR RESOURCES

Process Control has a supplemental set of exercises and solutions in an editable Word document that are made available only to teachers. These materials and other Stamps in Class resources can be obtained by joining the free, private Parallax Educators forum.

Both students and teachers are invited to join the public Parallax Stamps in Class forum, where they can discuss their experiences using Process Control or any other Stamps in Class text in the classroom. Students are encouraged to come here for assistance with

working through the projects in the text, and teachers are encouraged to offer support. Parallax staff moderate and participate in this forum.

To join the Stamps in Class forum, go to forums.parallax.com. After joining Stamps in Class, educators may email stampsinclass@parallax.com for instructions to join the Parallax Educators forum. Proof of status as an educator will be required.

THE STAMPS IN CLASS EDUCATIONAL SERIES

The Stamps In Class series of texts and kits provides affordable resources for electronics and engineering education. All of the books listed are available for free download from www.parallax.com. The versions cited below were current at the time of this printing. Please check our web sites www.parallax.com or www.stampsinclass.com for the latest revisions; we continually strive to improve our educational program.

Stamps in Class Student Guides:

What's a Microcontroller? is the recommended entry level text to the Stamps In Class educational series. Some students instead start with *Robotics with the Boe-Bot*, also designed for beginners.

"*What's a Microcontroller?*", Student Guide, Version 2.2, Parallax Inc., 2004
"*Robotics with the Boe-Bot*", Student Guide, Version 2.2, Parallax Inc., 2004

You may continue on with other Educational Project topics, or you may wish to explore our other Robotics Kits.

Educational Project Kits:

The following texts and kits provides a variety of activities that are useful to hobbyists, inventors and product designers interested in trying a wide range of projects.

"*Process Control*", Student Guide, Version 2.0, Parallax Inc., 2006
"*Applied Sensors*", Student Guide, Version 1.3, Parallax Inc., 2003
"*Basic Analog and Digital*", Student Guide, Version 1.3, Parallax Inc., 2004
"*Elements of Digital Logic*", Student Guide, Version 1.0, Parallax Inc., 2003
"*Experiments with Renewable Energy*", Student Guide, Version 1.0, Parallax Inc., 2004
"*Understanding Signals*", Student Guide, Version 1.0, Parallax Inc., 2003

Robotics Kits:

To gain experience with robotics, consider continuing with the following Stamps in Class student guides, each of which has a corresponding robot kit:

> **"*IR Remote for the Boe-Bot*", Student Guide, Version 1.0, Parallax Inc., 2004**
> **"*Applied Robotics with the SumoBot*", Student Guide, Version 1.0, Parallax Inc., 2005**
> **"*Advanced Robotics: with the Toddler*", Student Guide, Version 1.2, Parallax Inc., 2003**

Reference

This book is an essential reference for all Stamps in Class Student Guides. It is packed with information on the BASIC Stamp series of microcontroller modules, our BASIC Stamp Editor, and our PBASIC programming languages.

> **"*BASIC Stamp Manual*", Version 2.2, Parallax Inc., 2005**

FOREIGN TRANSLATIONS

Parallax educational texts may be translated to other languages with our permission (e-mail stampsinclass@parallax.com). If you plan on doing any translations please contact us so we can provide the correctly-formatted MS Word documents, images, etc. We also maintain a private discussion group for Parallax translators which you may join. This will ensure that you are kept current on our frequent text revisions.

SPECIAL CONTRIBUTORS

The authors would also like to thank the 2004 and 2005 EST 212 classes at Southern Illinois University Carbondale for testing much of the text during its development. Also, thanks to Barry Shahian and Clark Radcliffe for their feedback and suggestions.

Parallax Inc. would like to recognize their Education Team members: Project Manager Aristides Alvarez, Technical Illustrator Rich Allred, Graphic Designer Jen Jacobs, and Technical Editor Stephanie Lindsay. Special thanks also go to Andrew Lindsay, Chris Savage, and Kris Magri for their insightful consulting and review, and, as always, to Ken Gracey, the founder of Parallax Inc.'s Stamps in Class educational program.

Chapter 1: Process Control and Flowcharts

BEFORE YOU START

To perform the experiments in this text, you will need to have your Board of Education connected to your computer, the BASIC Stamp Editor software installed, and to have verified the communication between your computer and your BASIC Stamp. For detailed instructions, see *What's a Microcontroller?* - a free download from www.parallax.com. You will also need the parts contained in the Process Control Parts Kit. For a full listing of system, software, and hardware requirements, see Appendix B.

WHAT IS PROCESS CONTROL?

Process control refers to the control of one or more system parameters, such as temperature, flow rate or position. While most systems are a continual process, such as maintaining a temperature, other processes may be a sequence of actions, for example, the assembly of a product.

Control systems can be very simple or very complex. Figure 1-1 is a block diagram of a simple continuous control system. For control of the process, an input (such as a setpoint control or switch) is required into the controller. Based on the input, the controller will drive an actuator to cause the desired effect on the process.

Examples of actuators are heaters for temperature, pumps for flow, and servos for positioning.

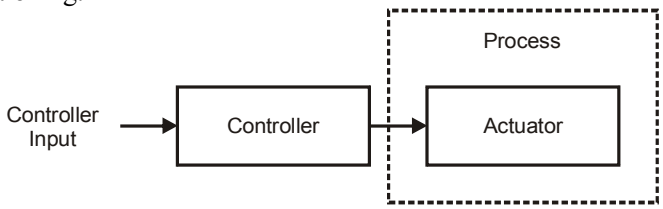


Figure 1-1
Simple Process Control Block Diagram

Consider the example of a common car heating system. The driver adjusts a temperature control to change the heat output of the vents. If the driver becomes too warm when weather conditions change, the temperature control must be adjusted to return to a comfortable temperature. This is a very simple system in that most automobiles do not monitor the cabin with temperature sensors to automatically control the heat output of the vents.

A more sophisticated system would have a sensor to monitor temperature and provide feedback to the controller. The controller would automatically adjust the actuator to regulate the controlled parameter - temperature. The controller would drive the heating system to maintain the temperature near the defined set point. An example of this is your home heating system.

Consider the difference between how the cabin temperature of the automobile is controlled versus the temperature in a home. In the automobile, the heat output is variable but has no sensors that directly affect the heat output and maintain temperature. In home heating a sensor is used to monitor the temperature, but the output of the heating system is not variable; it is either on or off and cycles to maintain temperature in a comfortable band. The controller itself may be very simple, such as a metallic coil that expands and contracts, or more complex, such as a microcontroller similar to the BASIC Stamp.

These are two very unique means of controlling a process. First, the types of drive employed may be variable or on/off. Second, whether feedback from the system may or may not be used in the control of the system.

INPUT, DRIVE AND MONITORING

Just as important as the type of control employed are the methods used for the input into the controller. Will the inputs provide a simple on/off input to the controller? If using an analog (variable level) input instead of a digital one (only two levels), how can it be conditioned for on/off input if needed? If analog input is required, what are methods to bring this data into the BASIC Stamp? How is the data represented in the BASIC Stamp and how can it be converted to meaningful information?

In terms of the drive of a system, there are several questions as well. Do we need to employ on/off control of the actuator, such as turning a heater or pump on or off? Does the process require variable control of the drive such as regulating heat or flow output between on or off? Does the process actuator require higher current or voltage than is provided by the BASIC Stamp? How can the BASIC Stamp outputs be used to control these actuators?

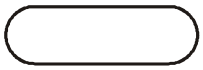
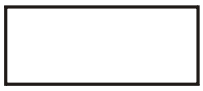
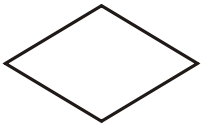
In industry, monitoring of systems is often required in order to ensure proper control action and to determine response in order to adjust this control action. Another monitoring aspect is data logging, or being able to collect real time data from the system for analysis.

This text explores these areas of process control through simple circuits using the BASIC Stamp microcontroller, and illustrates use with much larger systems.

ACTIVITY #1: FLOWCHARTS FOR REPRESENTING PROCESSES

When you hear the word 'flowchart', it may bring to mind programming, but a flowchart is often used for more than programming. A flowchart is a graphical representation of steps and decisions used to arrive at a logical outcome. It can be used to arrive at management decisions, system troubleshooting decisions, and other processes that involve well-defined steps and outcomes. Table 1-1 shows the most popular symbols used in flowcharting. These blocks, connected with flow lines, are used to describe the actions and flow of the program.

Table 1-1: Flowcharting Symbols

	Start/Stop: Indicates the beginning or end of a program or routine.
	Process: Indicates an internal process, such as calculations or delays.
	Input/Output: Indicates an input from an external source or output to an external source.
	Decision: Indicates a decision to continue flow in one of two directions based on a condition.
	Predefined Process: Indicates a predefined process, such as a subroutine, to be performed.
	Matching connectors indicate a connection between two locations in the flowchart.
	Flow lines: Indicates direction of flow between symbols.

While flowcharting has fallen out of fashion in many programming circles due to the advent of object-oriented programming (PBASIC used by the BASIC Stamp is procedural language), it is still an excellent tool when planning program flow. Flowcharting is particularly useful in process control because it can be used to visually represent the steps and decisions required to perform control of the system.

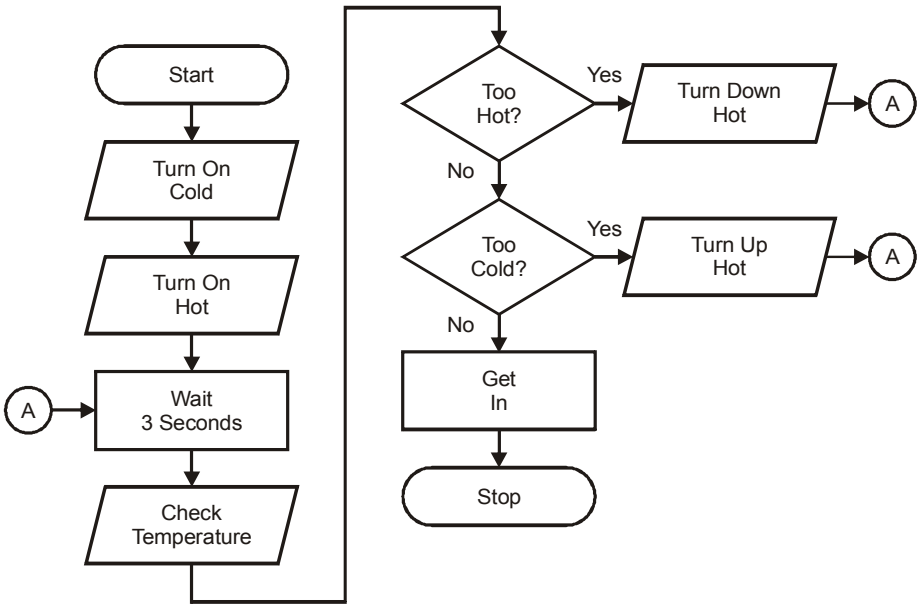
Take the everyday task of preparing the temperature of the shower before stepping into it. In pseudocode, or English statements outlining the steps to take, this is how we would proceed:

1. Turn on cold water.
2. Turn on hot water.
3. Wait 3 seconds for temperature to stabilize.
4. Test water temperature.
5. If too hot, then:
    a. Turn hot water down.
    b. Go back to step 3.
6. If too cold, then:
    a. Turn hot water up.
    b. Go back to step 3.
7. If just right then get in shower.

While it's not too difficult to read through these steps to see what actions should be taken, as a program or procedure becomes more complex it becomes more difficult to visualize the flow of the process and what actions and branches are needed. For example, how much more complex would the flow be if the hot water valve becomes fully open before the optimum temperature is reached?

As complexity increases, a flowchart makes it easier to visualize how the process will flow. Take a look at the flowchart in Figure 1-2, which describes the same process as the pseudocode above.

Figure 1-2 Adjusting Shower Temperature Flowchart



Note how each of the symbols is used.

- Typically, an input/output symbol is used when bringing data or information into the controller (in this case the person adjusting the temperature by sensing and adjusting the water actuators).
- The processing symbol is used when the controller is performing internal processing of data or a task, such as waiting or calculations.
- Finally, decision blocks are used to guide the flow of the procedure in one direction or another based on the decision results.

A decision can take one of two forms:

- Questions resulting in Yes or No.
- Statements resulting in True or False.

As humans, we typically work with questions resulting in yes/no. In flowcharting, it is better to use statements that result in true/false due to the logical nature of programming where conditions are checked to be true or false. Take the following example for the shower process:

- Is the water too hot? YES – Turn down the hot.
- The water is too hot. TRUE – Turn down the hot.

In programming, a typical condition may be:

```
IF (Water_Temp > 95) THEN …
```

In this example, when the condition is checked, the equality will either be true or false. Using true/false statements makes the transition from the flowchart to the programming language easier.

Challenge 1-1: Modify the Flowchart for True/False

√ Modify the flowchart in Figure 1-2 to use true/false statements instead of yes/no questions.

ACTIVITY #2: SEQUENTIAL FLOW AND CODE

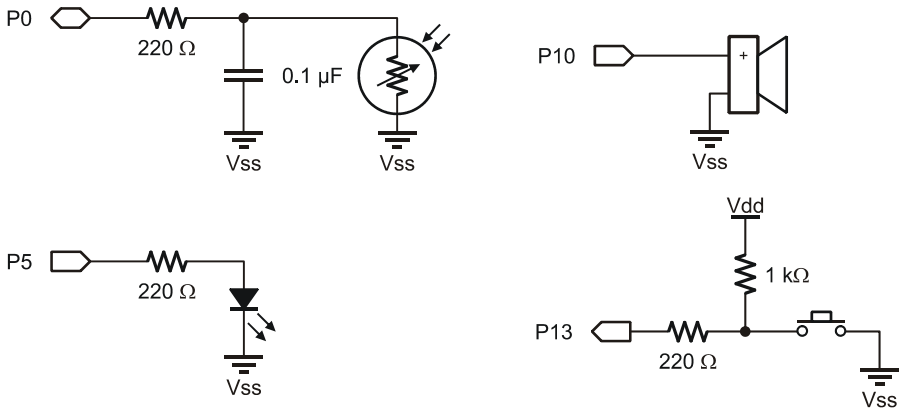
"Sequential flow" means moving from one operation to the next with no branches being made. In this activity a simple circuit will be used to illustrate principles of sequential flow and how the PBASIC language is used in programming the BASIC Stamp.

Parts Required

(3) Resistors – 220 Ω
(1) Resistor – 1 kΩ
(1) Photoresistor
(1) Pushbutton – Normally Open
(1) LED – Red
(1) Piezospeaker
(1) Capacitor – 0.1 µF

√ Construct the photoresistor, LED, piezospeaker, and pushbutton circuits shown in Figure 1-3.

Figure 1-3 Test Circuit Schematics



For an introduction to building basic circuits with these components, please see *What's a Microcontroller?*, the recommended starting point for the Stamps in Class series. It is available for free download or purchase from www.parallax.com.

Figure 1-4 is a flowchart to have the circuit continuously perform a sequence of operations. Without knowing any programming, can you determine what should occur when the program is entered and run?

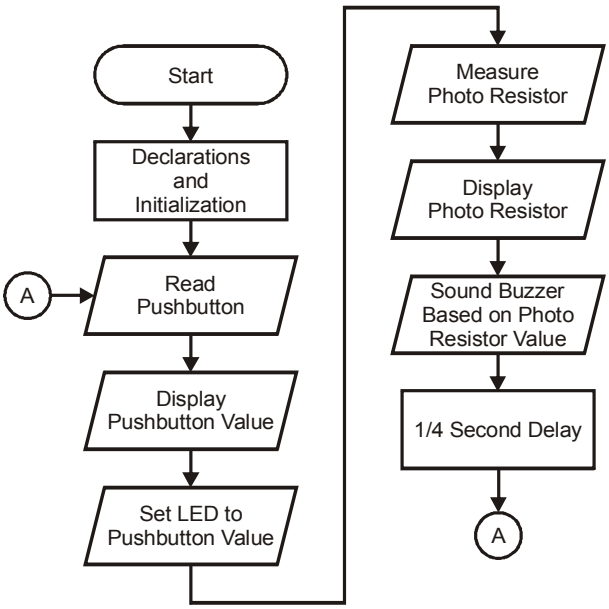


Figure 1-4
Simple Sequential Operation Flowchart

AVOID TYPOS! All of the BASIC Stamp (.bs2) programs listed in this text are available for free download from the Process Control product page at www.parallax.com.

Example Program: SimpleSequentialProgram.bs2

√ Enter and run SimpleSequentialProgram.bs2.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - SimpleSequentialProgram.bs2
' Tests and illustrates sequential flow using a simple test circuit.
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
Photo     PIN  0              ' Alias for photo resistor circuit on P0
LED       PIN  5              ' Alias for LED on P5
Buzzer    PIN  10             ' Alias for buzzer on P10
PB        PIN  13             ' Alias for pushbutton on P13
PBVal     VAR  Bit            ' Bit variable to hold pushbutton value
PhotoVal  VAR  Word           ' Word variable to hold RC Time value
```

```
BuzzerDur CON  250            ' Constant for duration of tone for buzzer


' -----[ Initialization ]---------------------------------------------------
OUTPUT LED                    ' Set LED pin to be an output
OUTPUT Buzzer                 ' Set Buzzer pin to be an output

' -----[ Main Routine ]-----------------------------------------------------
DO
  ' ******** Read Pushbutton
  PBVal = PB                  ' Read Pushbutton value and assign to PBVal
                              ' Display Pushbutton value

  ' ******** Display pushbutton value

  DEBUG CLS, "Pushbutton Value = ", DEC PBVal,CR

  ' ******** Set LED to pushbutton value
  LED = PBVal                 ' Set LED based on Pushbutton value

  ' ******** Measure Photoresistor
  HIGH Photo                  ' Charge photoresistor's RC network Capacitor
  PAUSE 10                    ' Allow 10 milliseconds to charge fully
  RCTIME Photo, 1, PhotoVal   ' Measure discharge time through photoresistor

  ' ******** Display photoresistor value
  DEBUG "Photo RC Time Value = ", DEC PhotoVal,CR

  ' ******** Sound buzzer at set duration at frequency of PhotoVal
  FREQOUT Buzzer,BuzzerDur,PhotoVal

  ' ******** 1/4 seconds delay
  PAUSE 250                   ' 1/4 second pause
LOOP                          ' Loop back to DO to repeat continuously
```

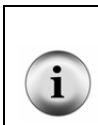
- √ Test the circuit by pressing the pushbutton and varying the light falling on the photoresistor.
  - o When the button is pressed does the state of the pushbutton change from 1 to 0 in the Debug Terminal?
  - o When the button is pressed does the LED change from on to off?
  - o When the sensor is darkened does the photoresistor RC time value change in the Debug Terminal?
  - o Does the frequency output of the buzzer change in relation to the photoresistor's RC time value? Note that the buzzer has a very limited frequency response range.
- √ If your circuit does not operate properly, verify your circuit connections and code.

Code Discussion

As you read through the program, you can see that the coding that corresponds to the various elements of the flowchart are well highlighted using comments.

The pushbutton switch is active-low, meaning that its value is 0 when pressed. This is because the pushbutton is pulled up to Vdd when not pressed and brought to Vss when pressed. (This will be explored more in Chapter 3.)

Note that the flowchart block for 'Measure Photo Resistor' takes 3 lines of code. The flowchart just describes the process and is not intended to be a line-by-line description. This flowchart could be used for coding or designing any number of devices in any number of languages.



Looking it Up: The PBASIC commands and programming techniques used here were introduced in *What's a Microcontroller?*, the recommended prerequisite to *Process Control*. If you would like a refresher about specific program elements, you can look it up quickly in the BASIC Stamp Editor's Help file. Or, refer to the *BASIC Stamp Syntax and Reference Manual*, available for purchase or free download from www.parallax.com.

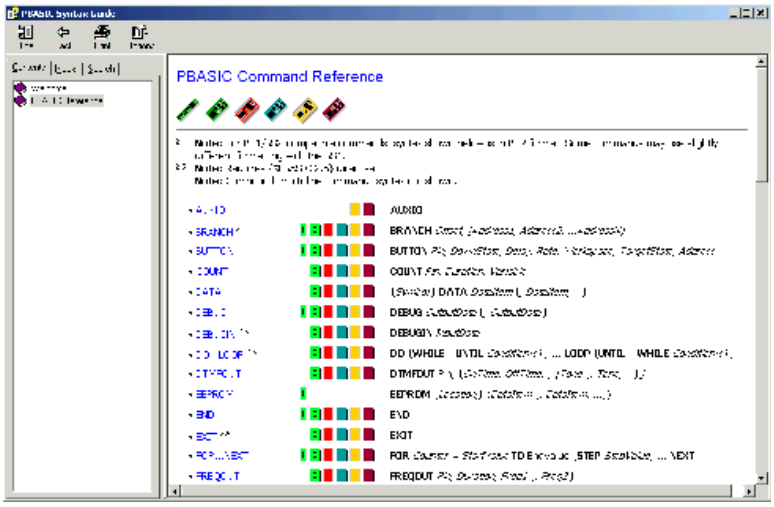


Figure 1-5
BASIC Stamp Editor's Help Files

The PBASIC Syntax Guide places information and examples for all commands at your fingertips.

Challenge 1-2: Coding from a Flowchart

Figure 1-6 is a flowchart for a different sequence of operations, using the same circuit. Code a program to match this sequence of events. Hints for coding are provided in the flow symbols.

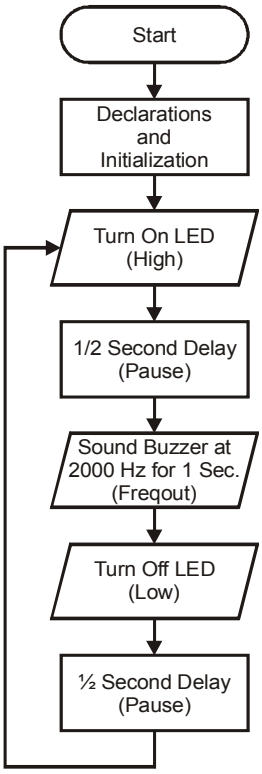


Figure 1-6
Challenge 1-2 Flowchart

ACTIVITY #3: FLOW AND CODING WITH CONDITIONAL BRANCHES

In most processes, measurements are made and decisions are then based on those measurements (such as, in the shower example, whether to turn up or down the hot water based on the current temperature). In the BASIC Stamp, there are multiple ways to code decisions and conditional branches.

Parts Required

Same as Activity #2

Consider the flowchart in Figure 1-7. What should occur when the button is pressed, and when it is not pressed?

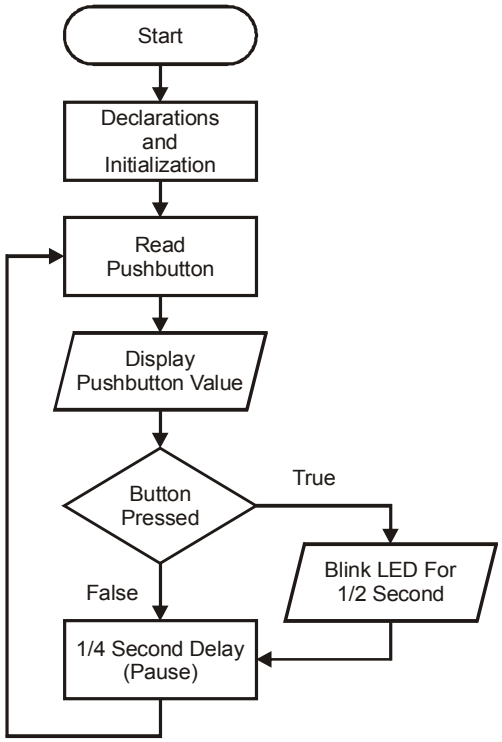


Figure 1-7
Conditional LED Blink Flowchart

If you said the LED would blink on for ½ second when the button is pressed, and not at all when not pressed, you would be correct.

Example Program: ConditionalLEDBlink.bs2

√ Enter, save and run ConditionalLEDBlink.bs2.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - ConditionalLEDBlink.bs2
' Blinks the LED based on state of Pushbutton
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
Photo     PIN  10       ' Alias for photo resistor circuit on P0
LED       PIN  5        ' Alias for LED on P5
Buzzer    PIN  10       ' Alias for buzzer on P10
PB        PIN  13       ' Alias for pushbutton on P13
PBVal     VAR  Bit      ' Bit variable to hold pushbutton value
PhotoVal  VAR  Word     ' Word variable to hold RC Time value
BuzzerDur CON  250      ' Constant for duration of tone for buzzer

' -----[ Main Routine ]-----------------------------------------------------
DO
  ' ******** Read Pushbutton
  PBVal = PB            ' Read Pushbutton Value and assign to PBVal

  ' ******** Display Pushbutton value
  DEBUG CLS,"Pushbutton value = ", DEC PBVal,CR

  ' ******** Button Pressed Conditional and Code
  IF (PBVal = 0) THEN   ' If pushbutton pressed is true then,
        HIGH LED        ' blink the LED
        PAUSE 500
        LOW LED
  ENDIF
  ' ******** 1/4 second pause
  PAUSE 250
LOOP                    ' Loop back to DO to repeat continuously
```

Code Discussion

The **IF…THEN…ENDIF** block is used to test the condition. Based on the result, the program will execute the code within the block if true or skip over it if false.

```
IF (PBVal = 0) THEN    ' If condition is true then,
      HIGH LED         ' blink the LED
      PAUSE 500
      LOW LED
ENDIF
' ******** 1/4 second pause
PAUSE 250
```

When the button is not pressed, the conditional test of **`PBVal=0`** will result in false because the value of **`PBVal`** is 1. Execution will branch to after the **`ENDIF`**, executing the **`PAUSE 250`**.

When the button is pressed, **`PBVal`** will in fact equal 0; **`PBVal=0`** will be true, the code within the block will be executed, and the LED will blink.

Code Formatting Tip: While indents in lines are not required, they do help to visually represent code that is common to sections.

Challenge 1-3: Code for True and False Conditions

Many times, different code must be executed depending on whether a condition is true or false. The **`IF…THEN…ELSE...ENDIF`** structure can be used to perform this task. If the condition is false, the code in the **`ELSE`** section will be executed.

```
IF (condition) THEN
  Code to run if true
ELSE
  Code to run if false
ENDIF
```

√ Figure 1-8 is a flowchart that requires different code depending on whether the button is pressed or not. Modify ConditionalLEDBlink.bs2 to match the flowchart's operation.

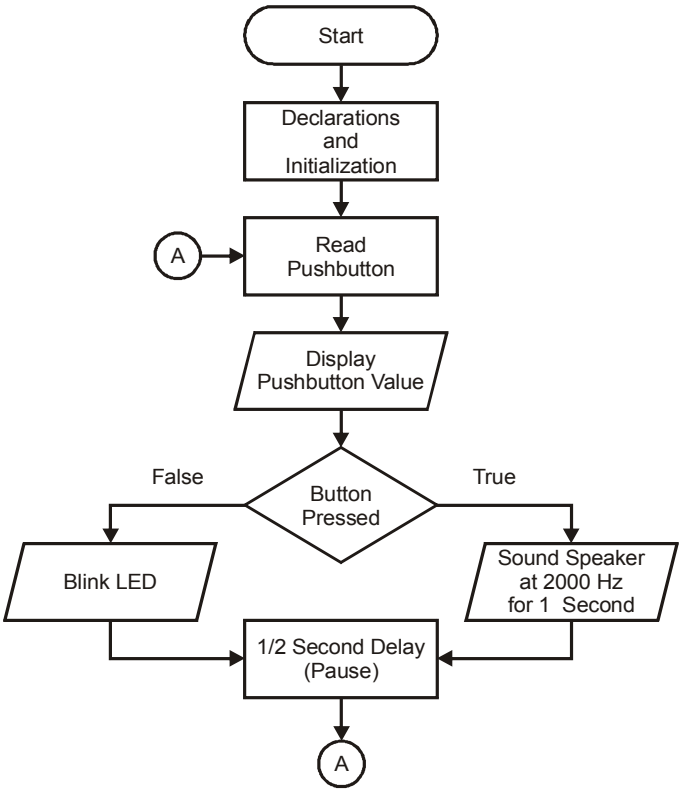


Figure 1-8
Conditional LED Blink or Tone Flowchart

ACTIVITY #4: PREDEFINED PROCESSES WITH SUBROUTINES

Parts Required

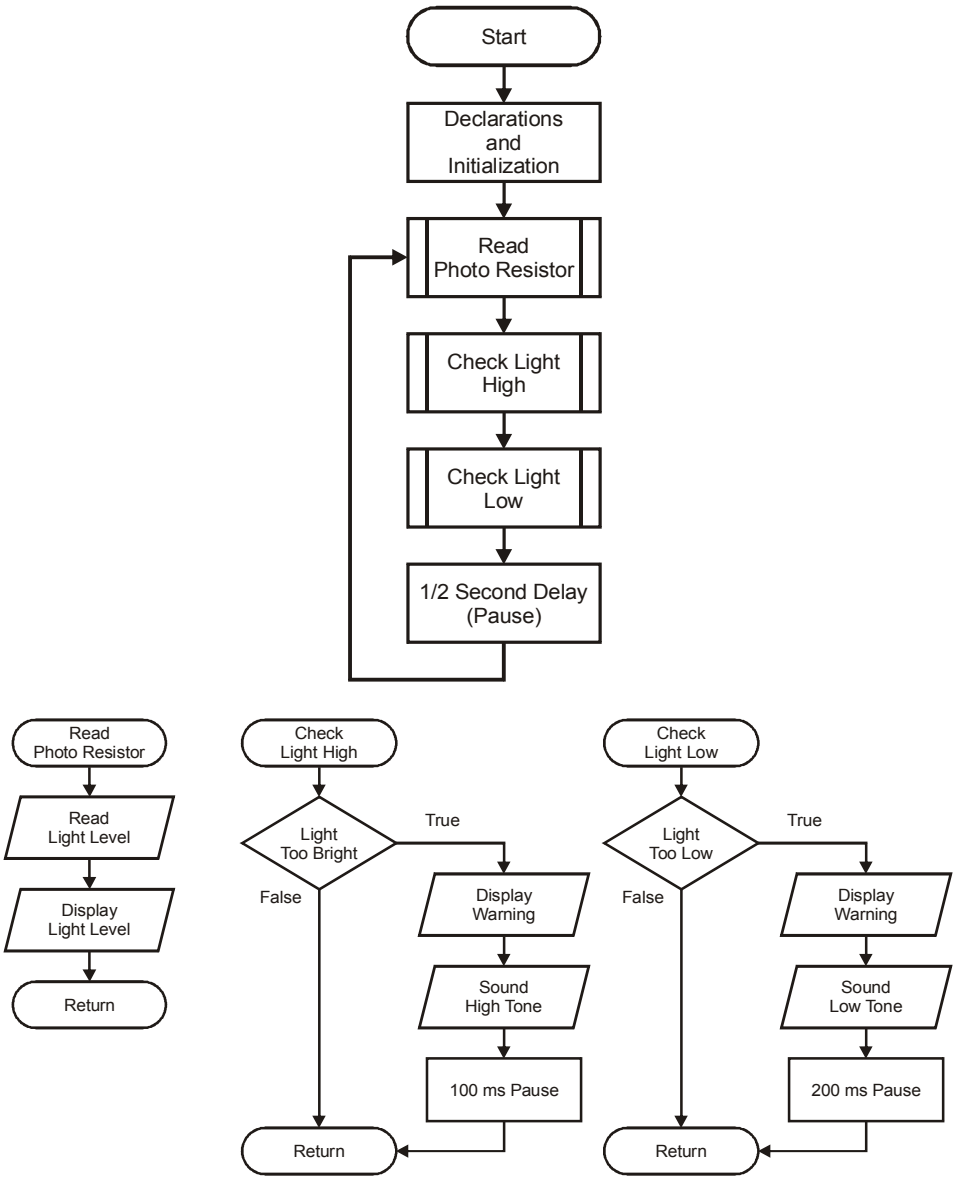
Same as Activity #2

As more operations are added to the flowchart, it can become quite large and complex. The same holds true for programs. In the previous programs, all operations were performed within the main routine, and the same held true in the flowchart.

As the process increases in size and complexity, it is best to break it down into more manageable pieces. By looking at the main loop of the flowchart or the main routine of the code, it is easy to see the overall operation of the program without being overwhelmed by the amount of code. Finally, analyzing or troubleshooting is much easier if it can be performed without having to flip between several pages or continually scroll up or down to different sections of the program. For example, consider the flowchart in Figure 1-9.

Looking at the main loop, it is easy to see the overall operation of the process. The pre-defined processes take the place of specialized code to perform these operations. Each pre-defined process has its own flowchart to define its operation. How will the process operate based on this flowchart? What occurs if the light level is low?

Figure 1-9 Light Alarms using Predefined Processes Flowchart



Example Program: LightAlarmsWithSubroutines.bs2

√ Enter and run LightAlarmWithSubroutines.bs2.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - LightAlarmWithSubroutines.bs2
' Sounds alarm based on photoresistor readings
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
Photo     PIN  0      ' Alias for photo resistor circuit on P0
LED       PIN  5      ' Alias for LED on P5
Buzzer    PIN  10     ' Alias for buzzer on P10
PhotoVal  VAR  Word   ' Variable to hold RC Time value
PhotoMin  VAR  Word   ' Holds minimum light level value
PhotoMax  VAR  Word   ' Hold maximum light level value

' ---[ Initialization ] ----------------------------------------------------
PhotoMin  =  500      ' Set minimum light value
PhotoMax  = 5000      ' Set maximum light value
PAUSE 1000            ' Allow connection to stabilize -- for Chapter 2

' -----[ Main Routine ]-----------------------------------------------------
DO
  GOSUB ReadPhoto
  GOSUB CheckLightHigh
  GOSUB CheckLightLow
  PAUSE 500
LOOP

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

ReadPhoto:           ' Read light level and plot values
  HIGH Photo
  PAUSE 10
  RCTIME Photo,1,PhotoVal
  DEBUG DEC PhotoVal, ",", DEC PhotoMin, "," ,DEC PhotoMax,CR
RETURN

CheckLightHigh:       ' Test if high light level
  IF (PhotoVal < PhotoMin) THEN
    DEBUG "LIGHT LEVEL HIGH!",CR
    FREQOUT Buzzer,100,3000
    PAUSE 100
  ENDIF
RETURN

CheckLightLow:        ' Test if low light level
  IF (PhotoVal > PhotoMax) THEN
    DEBUG "LIGHT LEVEL LOW!",CR
```

```
    FREQOUT Buzzer,200,1000
    PAUSE 200
  ENDIF
RETURN
```

√ Move your hand over the photoresistor, and watch the Debug Terminal. What occurs as the light level RC time is
  - Less than 500?
  - Between 500 and 5000?
  - Greater than 5000?

Your Debug Terminal should look similar to Figure 1-10. It displays the current level and the low- and high-level set points.

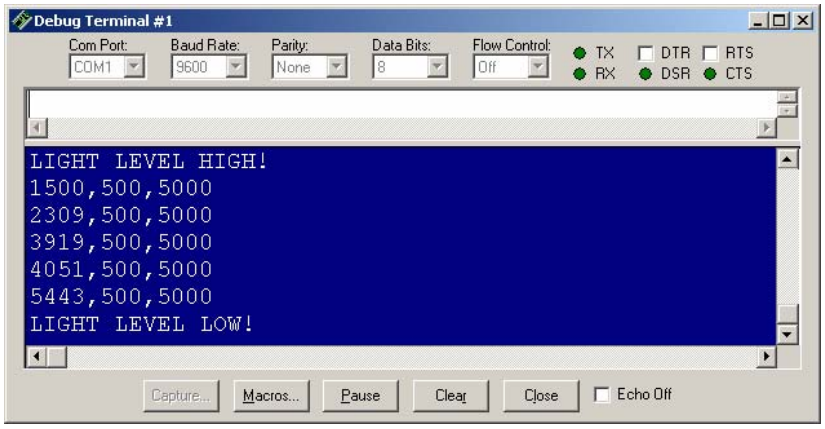


Figure 1-10 Debug Terminal Light Level Alarms

Values or tolerances of the photoresistor and capacitor may vary along with ambient light level where you are. Adjust the high and low level setpoints accordingly in the initialization section of your code.

Code Discussion

Using `GOSUB…RETURN` works well with our flowchart structure. The subroutines are the pre-defined processes. When the `GOSUB` call is run, program execution branches to the named routine. The routine code is executed. When complete, `RETURN` causes execution to branch back to the code after the `GOSUB` call.

Programming Tip: Every routine called with a `GOSUB` must exit with a `RETURN`. Internal pointers keep track of `GOSUBs` and `RETURNs`, and if not matched properly, will result in erroneous behavior of the processor.

Challenge 1-4: Add an Operational Indicator

1. Add a pre-defined process block to the main loop of the flowchart in Figure 1-9.
2. Also add a process flowchart to turn on the LED for 0.25 seconds at every pass through the main loop in order to indicate proper operation of the system.
3. Add code to LightAlarmWithSubroutines.bs2 to match the flowchart.

ACTIVITY #5: CONDITIONAL LOOPING

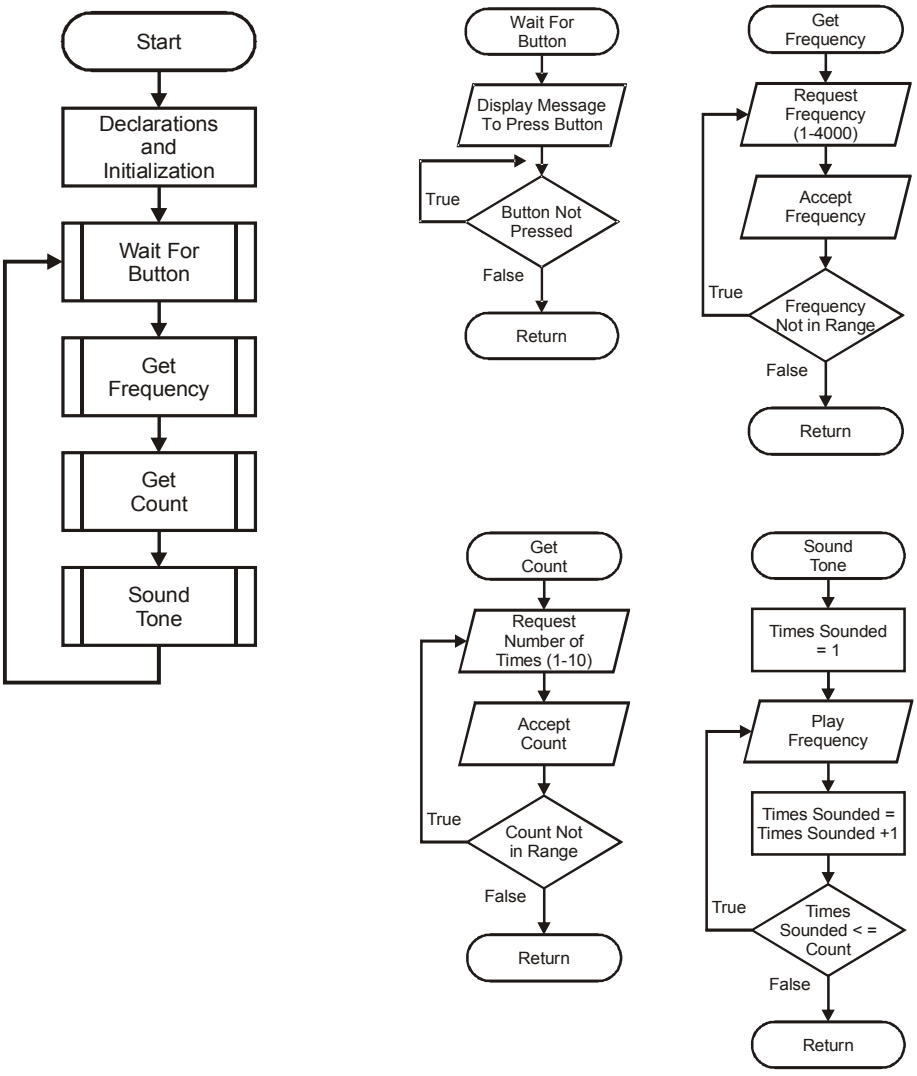
Many times in a process it is necessary to repeat a sequence based on a condition. On the other hand, halting or pausing an execution until a certain condition exists may be required. Consider the process of starting a piece of industrial machinery. Until conditions are met, such as an oil pump running, there may be no need to continue farther into the process. A conditional loop could be used to ensure that a condition exists prior to continuing with the sequence.

Parts Required

Same as Activity #2

Examine the Conditional Looping flowchart in Figure 1-11.

Figure 1-11 Conditional Looping Flowchart



Example Program: ConditionalLooping.bs2

- √ Enter, save and run ConditionalLooping.bs2.
- √ To begin, press the pushbutton as directed by the Debug Terminal.
- √ Enter a frequency to play and the number of times to play it by typing a value into the white text box at the top of the Debug Terminal, and then pressing Return or Enter.
- √ Test using valid and invalid values.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - ConditionalLooping.bs2
' Sounds tone using conditional loops
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
Photo     PIN  0        ' Alias for photo resistor circuit on P0
LED       PIN  5        ' Alias for LED on P5
Buzzer    PIN  10       ' Alias for buzzer on P10
PB        PIN  13       ' Alias for pushbutton on P13
PBVal     VAR  Bit      ' Bit variable to hold pushbutton value
PhotoVal  VAR  Word     ' Variable to hold RC Time value
FreqVal   VAR  Word     ' Frequency to sound
CountVal  VAR  Byte     ' Number of tones to sound
X         VAR  Byte     ' General Counting variable

' -----[ Main Routine ]-----------------------------------------------------
DO
  GOSUB WaitForButton
  GOSUB GetFreq
  GOSUB GetCount
  GOSUB SoundTone
  PAUSE 1000
LOOP

' -----[ Subroutines ]-----------------------------------------------------
WaitForButton:
  DEBUG CLS, "Press the pushbutton to begin",CR
  DO
  LOOP WHILE (PB=1)
RETURN

GetFreq:
  DO
    DEBUG CR,"Enter the frequency to play (1 to 4000)",CR
    DEBUGIN DEC FreqVal
  LOOP UNTIL (FreqVal <= 4000)   ' loop until within range
RETURN
```

```
GetCount:
  DO
    DEBUG CR,"Enter the number of times to play (1 to 10)",CR
    DEBUGIN DEC CountVal
  LOOP WHILE (CountVal > 10)      ' loop while out of range
RETURN

SoundTone:
  FOR X = 1 TO CountVal           ' Start X at 1 for counting up to CountVal
    FREQOUT Buzzer,500,FreqVal
    DEBUG "Buzzing ", DEC X,CR
  NEXT                            ' Add 1 to X and loop if X <= CountVal
RETURN
```

Program Discussion

The ConditionalLooping.bs2 program uses conditional loops in a variety of ways. The **DO...LOOP WHILE** within the **WaitForButton** routine will repeat while the condition is true. This occurs while the value of the pushbutton input is 1 or not pressed. Once the pushbutton is pressed, the condition will be false and the loop will end.

In the **GetFreq** routine, the **DO...LOOP UNTIL** will repeat until a value within range has been entered. **DEBUGIN** accepts data from the Debug Terminal and stores it as a decimal in the **FreqVal**.

In **GetCount**, a **DO...LOOP WHILE** is used to request the number of times to play the tone and will repeat while the value is outside the appropriate range.

In **SoundTone**, a **FOR...NEXT** loop is used. This is a special conditional loop used for repeating a sequence a set number of times:

```
FOR variable = Start_Value TO End_Value
```

The loop begins with the defined value set to the **Start_Value**. The code within the loop is performed. When **NEXT** is encountered, the variable is incremented and checked against the **End_value**. If the variable is not greater than the **End_Value**, the loop repeats. **X** is started at 1 and the loop continues until **X** exceeds the value entered by the user for **CountVal**.

Compare the flowcharts to the code for each routine. The use of either **WHILE** or **UNTIL** is at the programmer's discretion as long as it performs the task intended.

Challenge 1-5

√ Save ConditionalLooping.bs2 under a new name, then add variables and code required to allow the user to enter the duration the tone should be played (entered in milliseconds). Limit the maximum allowable duration to 1000 milliseconds.

CONCLUSION

Process control refers to the control of one or more system parameters. Typically, some form of input is used to adjust this process. A simple process, such as controlling temperature, may be performed in multiple ways. The control and complexity of the system is based on need. For process control the BASIC Stamp is ideally suited for many systems.

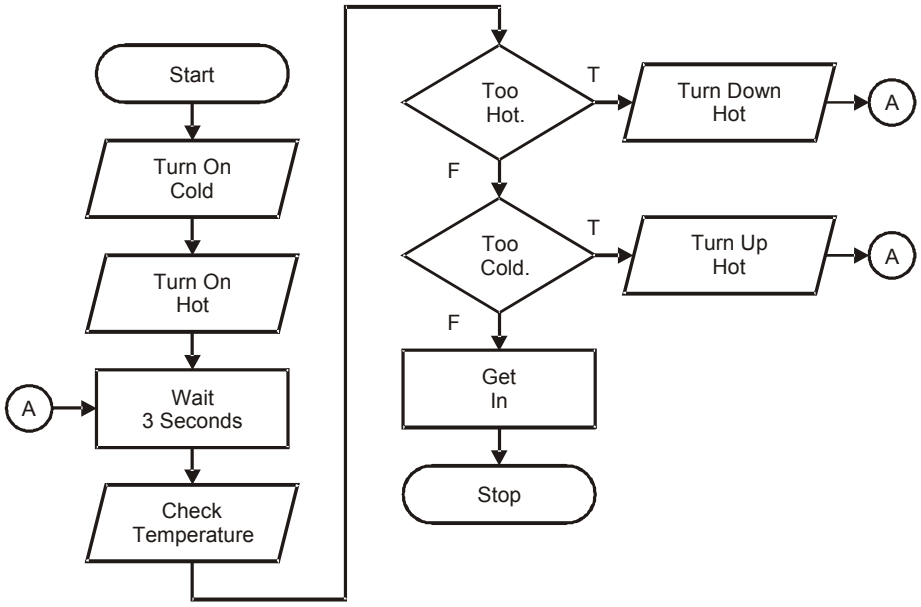
Flowcharts are a visual representation of a program or a process. The flowchart represents the necessary steps to perform the desired actions. Through the use of symbols the actions of the program or process are graphically depicted. With knowledge of PBASIC, the programming language of the BASIC Stamp, the process represented by the flowchart may be programmed into the BASIC Stamp.

Means to control output devices include using the **HIGH** and **LOW** commands; **FREQOUT** is used to sound tones; data may be sent to the computer using the **DEBUG** instruction. Conditions may be checked and simple true/false decisions may be made using the **IF...THEN** instructions. Looping is performed using the **DO...LOOP**, and adding **WHILE** or **UNTIL** looping may be performed conditionally. Programs may be broken down into smaller processes that are called with the **GOSUB** command and exited with the **RETURN** command.

SOLUTIONS TO CHAPTER 1 CHALLENGES

Challenge 1-1 Solution

Figure 1-12 Shower Temperature Flowchart with True/False



Note that the yes-no questions became true-false statements.

Challenge 1-2 Solution

Your program might look like this:

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - SimpleFlowchartChallenge.bs2
' Code from flowchart
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]--------------------------------------------------------
Photo     PIN  0                ' Alias for photoresistor circuit on P0
LED       PIN  5                ' Alias for LED on P5
Buzzer    PIN  10               ' Alias for buzzer on P10
```

```
' -----[ Main Routine ]------------------------------------------------------
DO
  HIGH LED                      ' Turn ON LED
  PAUSE 500                     ' 1/2 second delay
  FREQOUT Buzzer, 1000, 2000    ' Sound buzzer at 2000Hz for 1 second
  LOW LED                       ' Turn OFF LED
  PAUSE 500                     ' 1/2 second delay
LOOP                            ' Loop back to DO to repeat continuously
```

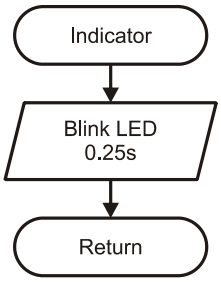
Challenge 1-3 Solution

```
' -----[ Title ]------------------------------------------------------------
' Process Control - ConditionalLEDBlinkChallenge.bs2
' Modify ConditionalLEDBlink for If-Else
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
Photo     PIN  0      ' Alias for photo resistor circuit on P0
LED       PIN  5      ' Alias for LED on P5
Buzzer    PIN  10     ' Alias for buzzer on P10
PB        PIN  13     ' Alias for pushbutton on P13
PBVal     VAR  Bit    ' Bit variable to hold pushbutton value
PhotoVal  VAR  Word   ' Word variable to hold RC Time value
BuzzerDur CON  250    ' Constant for duration of tone for buzzer

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

' -----[ Main Routine ]-------------------------------------------------------
DO
  ' ******** Read Pushbutton
  PBVal = PB          ' Read Pushbutton Value and assign to PBVal

  ' ******** Display Pushbutton value
  DEBUG CLS,"Pushbutton value = ", DEC PBVal,CR

  ' ******** Button Pressed Conditional and Code
  IF (PBVal = 0) THEN          ' If pushbutton pressed is true then,
    FREQOUT Buzzer, 1000,2000  ' True - Sound buzzer
  ELSE
        HIGH LED               ' False - Blink the LED
        PAUSE 500
        LOW LED
  ENDIF
  ' ******** 1/4 second pause
  PAUSE 250
LOOP                           ' Loop back to DO to repeat continuously
```

Challenge 1-4 Solution

1. To the main flowchart's loop, add another predefined process:

2. Make a flowchart for the predefined process (names should match):



3. To the programs Main Routine **DO...LOOP** add a subroutine call for your predefined process:

```
GOSUB Indicator
```

Under the Subroutines section, add the subroutine for the predefined process:

```
Indicator:
  HIGH LED
  PAUSE 250
  LOW LED
RETURN
```

Challenge 1-5 Solution

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - ConditionalLoopingChallenge.bs2
' Sounds tone using conditional loops
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]--------------------------------------------------
Photo     PIN  0          ' Alias for photo resistor circuit on P0
LED       PIN  5          ' Alias for LED on P5
Buzzer    PIN  10         ' Alias for buzzer on P10
PB        PIN  13         ' Alias for pushbutton on P13
PBVal     VAR  Bit        ' Bit variable to hold pushbutton value
PhotoVal  VAR  Word       ' Variabe to hold RC Time value
FreqVal   VAR  Word       ' Frequency to sound
CountVal  VAR  Byte       ' Number of tones to sound
DurVal    VAR  Word       ' Duration to sound tone
X         VAR  Byte       ' General Counting variable
```

```
' -----[ Main Routine ]------------------------------------------------------
DO
  GOSUB WaitForButton
  GOSUB GetFreq
  GOSUB GetCount
  GOSUB GetDuration
  GOSUB SoundTone
LOOP

' -----[ Subroutines ]--------------------------------------------------------
WaitForButton:
  DEBUG CLS, "Press the pushbutton to begin",CR
  DO
  LOOP WHILE (PBVal=1)
RETURN

GetFreq:
  DO
    DEBUG CR,"Enter the frequency to play (1 to 4000)",CR
    DEBUGIN DEC FreqVal
  LOOP UNTIL (FreqVal <= 4000)   ' loop until within range
RETURN

GetCount:
  DO
    DEBUG CR,"Enter the number of times to play (1 to 10)",CR
    DEBUGIN DEC CountVal
  LOOP WHILE (CountVal > 10)     ' loop while out of range
RETURN

GetDuration:
  DO
    DEBUG CR,"Enter the duration to play tone in milliseconds (0 to 1000)",CR
    DEBUGIN DEC DurVal
  LOOP WHILE (DurVal > 1000)     ' loop while out of range
RETURN

SoundTone:
  FOR X = 1 TO CountVal          ' Start X at 1 for counting up to CountVal
    FREQOUT Buzzer,DurVal,FreqVal
    DEBUG "Buzzing ", DEC X,CR
  NEXT                           ' Add 1 to X and loop if X <= CountVal
RETURN
```

Chapter 2: PC Based Monitoring and Control

2

In the program LightAlarmWithSubroutines.bs2 from Chapter 1 (page 18), it would be very difficult to determine rate of change, to look for trends, or to spot abnormalities as numbers change on the screen. It would also be difficult at this point to change the settings when the program is running since the alarm level setpoints are written in the downloaded code. However, additional hardware and programming could be added to your BASIC Stamp project to allow a means of adjusting the setpoints.

Process control systems that include computer monitoring and control are often referred to as Supervisory Control And Data Acquisition, or SCADA systems. LabView® from National Instruments is a very popular program in industry for data acquisition and control, though it requires a fairly expensive license and sometimes additional interface cards for your PC. StampPlot Pro from SelmaWare Solutions (and the authors of this text) is an alternative that is flexible and very affordable, in fact free for use by home and educational users. It was developed specifically for the monitoring and control of the BASIC Stamp.

ACTIVITY #1: USING STAMPPLOT FOR MONITORING AND CONTROL

Parts Required

Same as Chapter 1, Activity #2 (page 6).

StampPlot Pro Version 3, Release 6, referred to as StampPlot from here on, will be used as a PC based monitoring and control interface for many of the projects in this text. StampPlot Pro and related files may be downloaded and installed from the Process Control product page at www.parallax.com. From the Education menu, choose Stamps in Class Tutorials. Scroll down to *Process Control*, then click on the link to the products page. The download links will be at the bottom of the page.

- √ Download and install StampPlot Pro Version 3 Release 6 (or latest version).
- √ Run StampPlot from your desktop: Start → All programs → StampPlot Pro.

Figure 2-1 shows the default Plot Selection screen for StampPlot. Normally, one of the plot images would be clicked for a StampPlot configuration. Instead, configuration files called macros have been developed for this text. For more information on using StampPlot, please see the StampPlot help files.

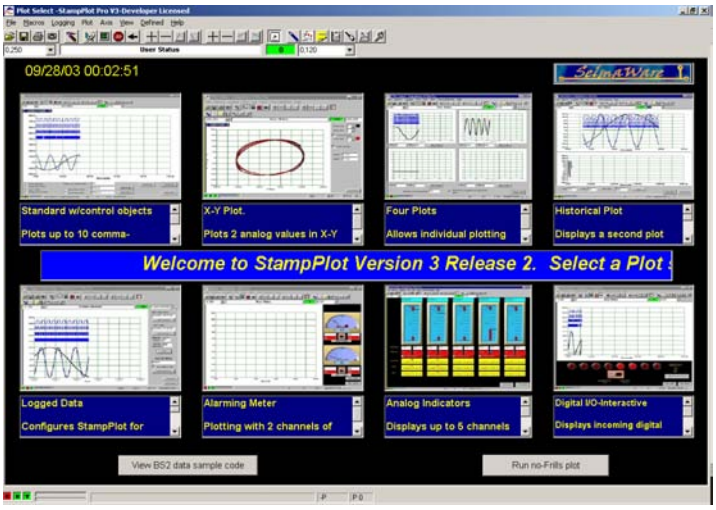


Figure 2-1
StampPlot Pro

Plot Selection
Default Screen

- √ From the Register Menu, select 'Free Home/Educ Standard License' to register your version of StampPlot. A warning message that you will not be able to use certain features to create macros will be displayed. These features are not necessary for this text.
- √ Click OK.
- √ Close StampPlot.
- √ Download and install the Process Control Macros from the same location as StampPlot.
- √ From your Start menu, go to Programs → Parallax Inc → StampPlot → SIC Process Control → Ch 2 → sic_pc_light_level.spm. This will load StampPlot with the selected macro, a text file containing configuration and data manipulation instructions for StampPlot.

StampPlot should load and look similar to Figure 2-2.

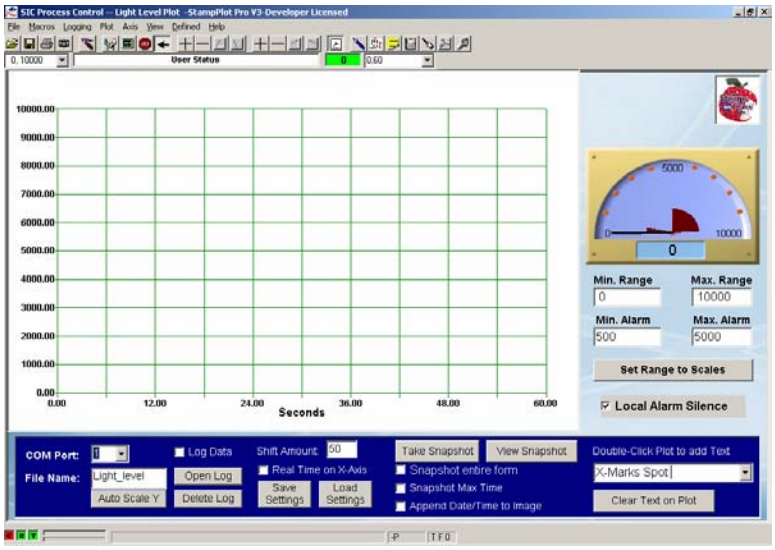


Figure 2-2
StampPlot with Light Level Macro Loaded

Let's test the software.

- √ Run LightAlarmsWithSubroutines.bs2 from page 18.
- √ Note the Com Port in use at the top-left of the BASIC Stamp Debug Terminal.
- √ Close the Debug Terminal.
- √ On StampPlot, select the Com Port noted in the Debug Terminal.

Only one application can use a COM Port at any one time. The user must disconnect on StampPlot (F6) to program the BASIC Stamp. Conversely, the user must close the BASIC Stamp Editor's Debug Terminal before connecting on StampPlot.

- √ Press F6 on the keyboard to connect. The C in the lower left corner should turn green, and the R should begin flashing red.
- √ Press F7 to enable plotting.
- √ Press F12 to reset the plot.
- √ Change the amount of light falling on your sensor. Three lines should begin plotting, the current value (black) and the high (green) and low (blue) level set points. Adjust the light of your sensor.

Figure 2-3 is a sample plot for this activity.

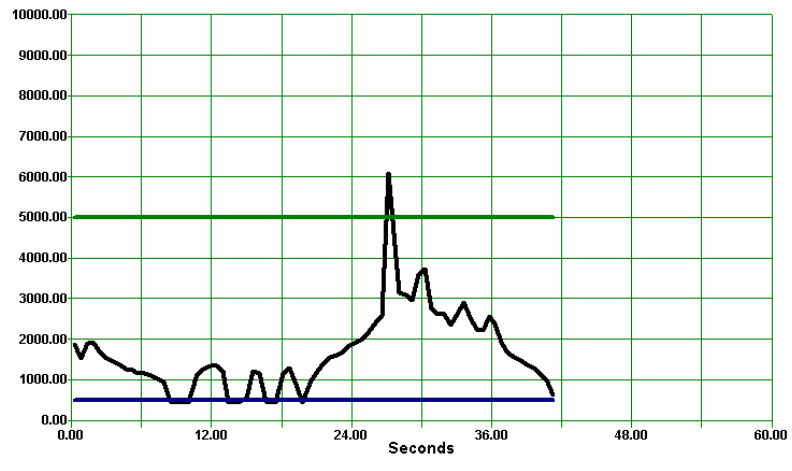


Figure 2-3
Sample Light Level Plot

If the high and low-level setpoints are exceeded, a new window opens titled "Messages" as shown in Figure 2-4. This lists the high and low level alarm messages sent by the BASIC Stamp along with the date, time and seconds into plotting they were received.

Figure 2-4 Messages Window with Alarm Warnings

Note that the virtual meter is displaying the current value. Enter new values in the text boxes under the meter and use your TAB key to move to another control to adjust the meter range. You must TAB off the control, or click another control, before the data is used. The meter alarm levels may also be set through the use of the next two text boxes,

but notice that the plotted alarm levels DO NOT change since that information is coming from the BASIC Stamp.

By clicking the "Set Range to Scales" button, the range of the meter will adjust to match the current scale on the Y-Axis of your plot.

By un-checking "Local Alarm Silence", the PC will sound an alarm when the level exceeds the meter's alarm levels (audio card & speakers required on the PC).

Code Discussion

StampPlot receives the serial data that was going to the Debug Terminal, analyzes it, and based on the format, utilizes the data in different ways. All strings must end with a carriage return as a general rule.

String Formatters	
String	**Action**
One or more values separated by commas	Strings are processed as analog values and plotted.
String begins with %	Strings are processed as digital values and plotted accordingly.
String begins with !	Strings are processed as control instructions to configure/control StampPlot.
String is not a value and has no special start character	Strings are processed as messages.

The analog data for the light levels is sent with the code of:

```
DEBUG DEC PhotoVal, ",", DEC PhotoMin, "," ,DEC PhotoMax,CR
```

An example of the data string sent for this code is: 500,825,5000.

The three comma-separated values will be plotted as three analog values. All strings sent to StampPlot must end in a carriage return. The macro will also use these values in updating the meter and for other uses.

Challenge 2-1

Perform the following:

- √ Disconnect on StampPlot (F6).
- √ Save LightAlarmsWithSubroutines.bs2 under a new name.
- √ At the end of the Initialization section of your BASIC Stamp program, add:

```
DEBUG CR, "!SPAN 0,2000", CR  ' Sets Y-axis range
```

- √ In the Main Routine **DO...LOOP**, add the code after **DO**:

```
GOSUB DisplayPB
```

- √ In the subroutine sections add the following routine:

```
DisplayPB:
    DEBUG IBIN PB, CR                ' Plot pushbutton as digital
  RETURN
```

IBIN is a modifier to send the value as indicated in binary. In the Debug Terminal you will see %1 or %0 depending on the state of the pushbutton. The symbol % indicates the value is binary.

Plotting Multiple Digital Values: Multiple digital values may be plotted by sending a string of binary data preceded by %, such as %1011, which could represent four digital I/O states or conditions. Use **IBIN4** when sending 4 values to ensure leading zeros are sent.

- √ Program the BASIC Stamp and close the Debug Terminal.
- √ Reconnect on StampPlot (F6).

1. What effect does the code **!SPAN 0,2000** have on the Y-Axis?

2. Press and release the pushbutton several times slowly. How is the digital value of the pushbutton displayed?

ACTIVITY #2: STAMPPLOT INTERACTIVE CONTROL

StampPlot also supports interactive control with the BASIC Stamp. The BASIC Stamp can request data from StampPlot and use the data for updating parameters.

Parts Required

Same as Chapter 1, Activity #2 (page 6).

- √ Open StampPlot to the macro sic_pc_light_level.spm.
- √ In the BASIC Stamp Editor, open LightAlarmWithSubroutines.bs2 from page 18 and save it with a new name.

Now let's add the following elements to your code:

- √ In the Main Routine **DO...LOOP**, add the code after **DO**:

```
GOSUB ReadStampPlot
```

- √ In the Subroutines section add the following routine:

```
ReadStampPlot:                    ' Update values from StampPlot
  DEBUG "!READ (txtMinA)", CR     ' Read value of Min Alarm text
  DEBUGIN DEC PhotoMin            ' Accept data and store
  PAUSE 50                        ' Allow echo to clear from BS2
  DEBUG "!READ (txtMaxA)", CR     ' Read value of Max Alarm text
  DEBUGIN DEC PhotoMax            ' Accept data and store
  PAUSE 50                        ' Allow echo to clear from BS2
RETURN
```

- √ Run the modified program.
- √ Close the Debug Terminal and connect on StampPlot. Both the lower-left R and T indicators should be blinking as data is received and transmitted.
- √ Change the Min. Alarm and Max. Alarm text box values (be sure to Tab-off to set).

What occurs to the alarm level setpoints on the plot? They should change to match the new settings. The BASIC Stamp is reading the values from the interface.

Code Discussion

When the BASIC Stamp executes:

```
DEBUG "!READ (txtMinA)",CR
```

...StampPlot recognizes the **!READ** instruction and sends to the BASIC Stamp the value of **txtMinA**, which is the name of the textbox containing the minimum alarm level. This command:

```
DEBUGIN DEC PhotoMin
```

...accepts data arriving on the data port and stores it as a decimal value. Execution will cease until serial data arrives. Another way to accept arriving data for above would be:

```
DEBUG "!READ (txtMinA)", CR
SERIN 16, 84, 200, TimeOut1, [DEC PhotoMin]
TimeOut1:
```

...where the parameters are:

SERIN ***pin, baud_value, timeout_value, timeout_label,*** **[** ***DEC variable*** **]**

While this looks more complex, **SERIN** does support timeouts. If StampPlot is not connected, or if there were other communication problems, your program would not sit idle endlessly waiting for data. This is usually an undesirable situation in a process control scenario! Depending on your needs, either may be used.

The **PAUSE 50** after the **SERIN** allows data echoed back from the BASIC Stamp to clear prior to sending new data.

When using DEBUGIN, it is important to begin the program with PAUSE 1000. Placing **PAUSE 1000** in the Initialization section ensures that the COM port connection is stable before requesting data from StampPlot. If it isn't, the BASIC Stamp will 'hang' awaiting data that never arrives. If the BASIC Stamp does appear to hang – no sign of data arriving – press the reset button on the BASIC Stamp board.

ACTIVITY #3: STAMPPLOT TOOLBAR CONTROLS

Parts Required

Same as Chapter 1, Activity #2 (page 6).

- √ Open StampPlot to the macro sic_pc_light_level.spm.
- √ In the BASIC Stamp Editor, open LightAlarmsWithSubroutines.bs2 from page 18 and save it with a new name.

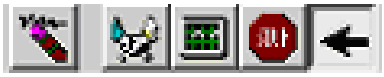
StampPlot is very versatile, and choosing what to use can be a little overwhelming. This activity will explore some of the basic controls of StampPlot including the specialized control section for the Process Control activities.

Figure 2-5 is the toolbar of StampPlot. Many of these controls will be important as you perform data acquisition.

Figure 2-5 StampPlot Toolbar

The toolbar control functions from left to right are listed. Important controls for this text are indicated by an asterisk (*). For more information, please see additional StampPlot documentation.

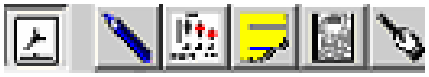
- Open: Opens a saved plot, log file, macro, or snapshot image file. Save: Saves the current plot data to a *.plt file in the StampPlot Data directory.
- Print: Prints the plot.
- *Snapshot: Creates a JPG image of the plot and saves it to the StampPlot Data directory.



- *Reset: Resets the plot to time 0 and erases the plot.
- *Connect: Connects StampPlot to the selected Com Port. Note: StampPlot is configured to reset the Stamp when the connection is made.
- *Plot: Enables plotting of incoming data.
- *Stop Plot: Stops plotting when the maximum time is reached (Stop normally activates under other conditions, but has been configured for this for our purposes).
- *Shift Plot: Enables the plot to shift left when the maximum time is reached. If Shift is not enabled, the plot will reset when the maximum time is reached unless Stop Plot is enabled. (Turning shift off does not reset the plot by default, but has been configured as such for our purposes.)

- *Double Y-Axis span or range.
- *Half Y-Axis span or range.
- *Shift Y-Axis span or range up.
- *Shift Y-Axis span or range down.

- *Double X-Axis span or range.
- *Half X-Axis span or range.
- *Shift X-Axis span or range left.
- *Shift X-Axis span or range right.



- Time Stamp: Add date and time to data for messages and data file.
- Configure: Configures StampPlot for various settings and configurations.
- Values: Shows plot values under mouse pointer and minimum and maximum values plotted.
- Messages: Opens Message Window.
- Debug/Immediate: Opens the StampPlot Debug Terminal.
- Playback: Allows playing of the current plot at various speeds.

The left and right dropdown boxes below the toolbar can be used to select or enter a Y-Axis analog span and X-Axis time span in seconds. Tab-off to set.

At the bottom are connection, data points and queue indicators.

- C = Green when connected, red when disconnected.
- R = Red when data is arriving.
- T = Red when data is transmitted.
- Top bar: Data points – This depicts how full the total number of data points for storing data is. This data is used to redraw the plot when required. Once filled, the oldest 25% data will be flushed from memory.
- Bottom bar: Queue – This depicts the amount of data that has arrived and is awaiting processing. If data arrives too quickly, this will begin to fill and plotting will be delayed. If this occurs, place a `PAUSE` in the program to slow down data sent to StampPlot.

Challenge 2-3: Configuring StampPlot

1. Use StampPlot controls for a configuration to plot 30 seconds of data for a light level range of 500 to 1000. The plot should reset and repeat every 30 seconds.

2. Use StampPlot controls for a configuration to plot 60 seconds of data for a light level range of -2000 to 2000. The plot should stop at the end of that time.

3. Use StampPlot controls for a configuration to show 15 seconds of data for a light level range of 0 to 5000. The plot should shift continually when the maximum time is reached.

4. Open the Values Window while collecting data (Menu View→Values). Move your mouse pointer over the plot and note how the values correspond to the X- and Y-axis values.

ACTIVITY #4: SPECIALIZED INTERFACE CONTROLS

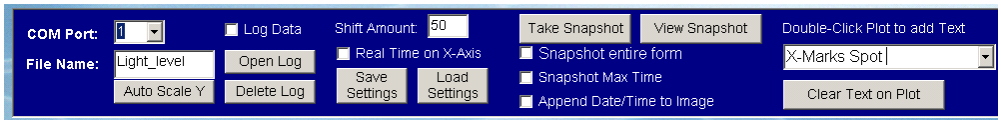
Parts Required

Same as Chapter 1, Activity #2 (page 6).

- √ Open StampPlot to the macro sic_pc_light_level.spm.
- √ In the BASIC Stamp Editor, open LightAlarmsWithSubroutines.bs2 from page 18 and save it with a new name.

To support common tasks and data acquisition for this text, all the Process Control StampPlot macro interfaces will have the control interface section bar shown below in Figure 2-6.

Figure 2-6 Specialized Interface Controls for Process Control StampPlot Macros



- **COM Port**: Used to select the serial communications port that the BASIC Stamp is connected to.
- **File Name**: An assortment of files and settings will use this specified name. The user may change the name for collecting data under different file names as desired.
- **Auto Scale Y**: Auto-scales the Y-Axis based on the minimum and maximum values for the currently plotted data.
- **Log Data**: When checked, arriving data will be logged in a data file for review, spreadsheets, or other uses. The arriving data will be time-stamped unless de-selected on the toolbar. The filename will be (file_name)_dat.txt where (file_name) is the text entered into the File Name text box.
- **Open Log**: Opens the current data log in use. If the file does not exist, nothing will occur.
- **Delete Log**: Deletes the current data log.
- **Shift Amount**: Specifies the percent the plot will shift to the left when the maximum time is reached. Lowering this value allows smoother scrolling of the plot but may slow plotting.
- **Real Time on X-Axis**: Sets the X-axis to show time as date/time of day instead of seconds.
- **Save Settings**: Save the current configuration of the plot and screen controls to the Windows registry. The data is saved using the text entered into the File Name textbox. By changing the file name, unique configurations can be saved.
- **Load Settings**: Loads a named configuration of the plot and screen control from the Windows registry. The name of the configuration is the text in the File Name textbox. This allows recall of specialized settings based on the user's need.
- **Take Snapshot**: Takes a JPG image of the plot. Images are saved to the StampPlot Data directory (C:\Program Files\StampPlotPro_V3\Data). Snapshots will be named using the text provided in the File Name text box.
- **View Snapshot**: Opens the last snapshot taken with the default image software installed on your computer.
- **Snapshot entire form**: The snapshot is the plot only by default. By selecting this choice, the snapshot image will be of the entire interface form. The interface must remain visible on the screen when using this feature.
- **Snapshot Max Time**: When the maximum plot time is reached, a JPG image of the plot is created and saved to the Data directory. The name of the file will be

the text in the File Name textbox. The user may also take a snapshot at anytime by using the Snapshot button the toolbar.

- **Append Date/Time to image**: Appends the current date and time to the snapshot file name, (File_Name)Date_Time.jpg, to create unique image files every time the plot reaches maximum time and "SnapShot at Max Time" is enabled. If this is not enabled, the snapshot image will be over-written each time unless the name is manually changed.
- **Double-Click Plot to add Text**: When the plot area is double-clicked, this will add the text in the box to allow annotating of your plot. New text may be entered in the drop-down box and will be added to the drop-down for ready-recall.
- **Clear Text on Plot**: Clears your added text from the plot.

Challenge 2-4: Using Interface Controls for Logging

1. Enable logging of data, delete the current logs, plot one minute's worth of samples with levels above and below the setpoints, and view the logs. Note the format of a line. Compare the data to the plotted value. What data is represented for each comma-separated value?

2. Configure StampPlot to reset at the end of each plot (toolbar button) and to take a snapshot of the plot when at the maximum time (interface checkbox). Allow at least one complete plot to be collected. View the saved plot image.

3. Configure StampPlot to shift at the end of each plot with a shift percentage of 75%. Enable snapshots with the date and time appended, and to automatically take snapshots at maximum plot time. Allow data to be collected for at least three complete plots.
    - Use 'Open Plot' on the toolbar.
    - Change File type in the file open window to 'Snapshots (*.jpg)'.
    - Note the name used for each snapshot. Select and open a snapshot.

4. Note the current settings of the interface controls, including the meter textboxes (change these to something you'll remember). Save the current configuration using the 'Save Settings' button on the interface. Close StampPlot and re-open the macro. Use the 'Load' interface button and observe the changes to the interface settings.

5. Plot the light level and select to stop the plot at maximum. About halfway through, fully darken the light falling on the sensor. Once stopped, annotate the plot with text about what occurred at that point.

CONCLUSION

Computer based software is useful for data acquisition and control of a system. It allows the operator to note current conditions, perform trend analysis, collect data to files, and update system settings. StampPlot is specialized software designed for the BASIC Stamp for this purpose.

StampPlot allows the real-time acquisition and plotting of analog and digital data. Furthermore, it provides both specialized interfaces as in virtual instruments to view data, and interactive control with the BASIC Stamp. Data strings from the BASIC Stamp can be used to send analog data, send digital data, control StampPlot, or send messages to the user. StampPlot also provides means to log data, messages and images to file for later analysis and review.

SOLUTIONS TO CHAPTER 2 CHALLENGES

Challenge 2-1 Solution

1. The Y-Axis is set to values of 0 and 2000.
2. The pushbutton value is plotted as a blue trace at the top of the plot. It changes between 2 states, high and low for binary values.

Challenge 2-3 Solutions

1. Use the right drop-down box below the toolbar at top to set a range. Manually enter 0,30 and tab-off to set. The right + and – buttons may also be used to adjust. Use the left drop-down box below the toolbar at top to set a range. Manually enter 500,1000 and tab-off to set. The left + and – buttons may also be used to adjust. To reset after 30 seconds, turn off the shift button on the toolbar at top.
2. As in number 1, enter values for range and time in the top drop-down boxes and tab-off. To stop the plot, the 'Stop' button on the toolbar should be on (depressed). After being stopped, the plot will need to be reset prior to connecting again or it will stop once again.

3. As in number 1, enter values for range and time in the top drop-down boxes and tab-off.
4. To shift and plot the 'Stop' button should be off, and the 'shift button' should be on in the toolbar.

Challenge 2-4 Solutions

1. Use the 'Log Data', 'Delete Logs' and 'Open Log' controls in the lower interface section. A typical log entry is:

   05/02/04 12:11:17.63,6.370,1,720,500,5000
   Where the data represented is:
   Date and time, seconds into plot, digital value of pushbutton (may be blank), current value, minimum setpoint, maximum setpoint.

   In general, the format will be:
   Date and time, seconds into plot, digital string, each analog value

2. On the toolbar at top, the 'Shift' button should be up.
   - On the interface controls, the 'Snapshot Max Time' control should be checked.
   - After shifting once, click the 'View Snapshot' button to view.
3. On the toolbar at top, the 'Shift' button should be up.
   - In the interface section, set 'Shift Amount' to 75%
   - Check the 'Append Date/Time' checkbox.
   - Check the 'Snapshot at Max Time' checkbox.
   - When you use the 'Open Plot' on the toolbar and select *.jpg, a list of image names with data and time should be present.
4. When the configuration is loaded, the saved setting should return.
5. The toolbar 'Stop' should be on.
   - Reset the plot and collect data.
   - Once done collecting data, enter 'Totally Dark!' in the 'Double-Click to add text' drop-down box.
   - Double-click the plot above the point it was darkened.

Chapter 3: Digital Input Conditioning

Digital Input seems pretty cut and dried. An input voltage at Vdd is recognized as a digital HIGH (binary 1). An input voltage at Vss (ground) is recognized as a digital LOW (binary 0). However, what if the BASIC Stamp input is not connected to either? What state will it assume, if any? What if the input is 2.5 V?

Devices which supply an input to the BASIC Stamp may not always provide +5 V for a HIGH and 0 V for a LOW. It is important to understand the digital input characteristics of the BASIC Stamp. Also important are the proper techniques of conditioning signals from mechanical input devices such as pushbuttons. Conditioning from electronic input devices is also frequently necessary as well.

In this chapter we will explore the input characteristics of the BASIC Stamp, the basic operation of a BJT (Bipolar Junction Transistor) and mechanical and electronic switch interfacing.

ACTIVITY #1: MEASURING THE THRESHOLD VOLTAGE

A HIGH level, or logic 1, is typically the positive voltage of the system. A LOW level, or logic 0, is typically the negative supply, or ground reference, of the system. For the BASIC Stamp these are labeled Vdd (+5 V) and Vss (0 V, which is "ground").

What if the voltage at the input were 3.5 V? Is this HIGH or LOW? What about 2.5 V? 2.0 V? 1.0 V? Since a digital system only can be one of two states, at what input voltage do the HIGH and LOW states transition? This activity will measure the threshold voltage below which the input is LOW and above which the input is HIGH. An Analog to Digital Converter (ADC) will be used to measure and plot the voltage levels at the input. A full discussion of the ADC is covered in Chapter 6.

Parts Required

(1) ADC0831 Analog to Digital Converter
(1) 10 kΩ Single-Turn Potentiometer
(2) 220 Ω Resistor
(1) LED – Red

√ Construct the circuit in Figure 3-1.

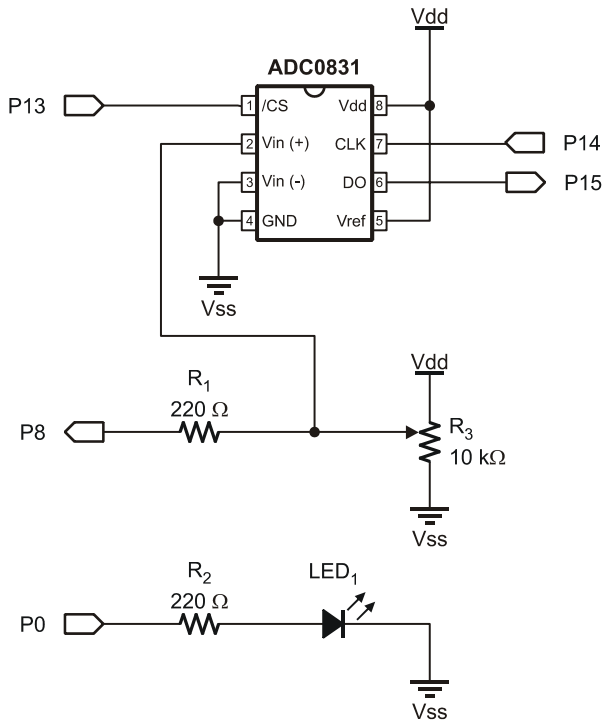


Figure 3-1
Analog and Digital Data Monitoring Circuit

Example Program: DataMonitoring.bs2

√ Enter and run the BASIC Stamp program DataMonitoring.bs2.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - DataMonitoring.bs2
' Monitors and Plots Analog and Digital Data
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
DigDataIn   VAR Bit      ' Digital input data
ADC_DataIn  VAR Byte     ' Analog to Digital Converter data
```

```
LED         PIN 0        ' LED output pin
DigIn       PIN 8        ' Digital input pin monitored
ADC_CS      PIN 13       ' ADC Chip Select pin
ADC_Clk     PIN 14       ' ADC Clock pin
ADC_Dout    PIN 15       ' ADC Data output

' -----[ Initialize ] ------------------------------------------------------
OUTPUT LED               ' Set LED as output
PAUSE 1000               ' Allow connection to stabilize

' -----[ Main Routine ]-----------------------------------------------------
DO
  GOSUB ReadData
  LED = DigIn
  GOSUB PlotData
  PAUSE 500
LOOP

' -----[ Subroutines ]------------------------------------------------------
ReadData:                ' Read ADC 0831
  LOW ADC_CS             ' Enable chip
  DigDataIn = DigIn      ' Read digital input value to coincide with ADC read
  SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_DataIn\9] ' Clock in data from ADC
  HIGH ADC_CS            ' Disable ADC
RETURN

PlotData:                ' Send data to StampPlot
  DEBUG IBIN DigDataIn,CR ' Plot indicated binary value
  DEBUG "[",             ' Bracket for StampPlot math operation
        DEC ADC_DataIn,  ' Analog data
        ",*,.0196]",CR   ' Convert ADC value to voltage by StampPlot
RETURN
```

- √ Close the Debug Terminal.
- √ Load StampPlot with the macro sic_pc_data_monitoring.spm
- √ Connect and plot.
- √ Adjust the potentiometer. The analog voltage and plotted value should change accordingly. LED_1, on the BASIC Stamp output pin P0, indicates the HIGH/LOW status of P8.
- √ Note the voltage at which the input P8 changes between HIGH and LOW logic levels. This is the threshold voltage.

Figure 3-2 shows an example test in which the threshold voltage was found to be approximately 1.45 V. The digital trace at the top goes high and low as the analog voltage goes above or below the threshold voltage.

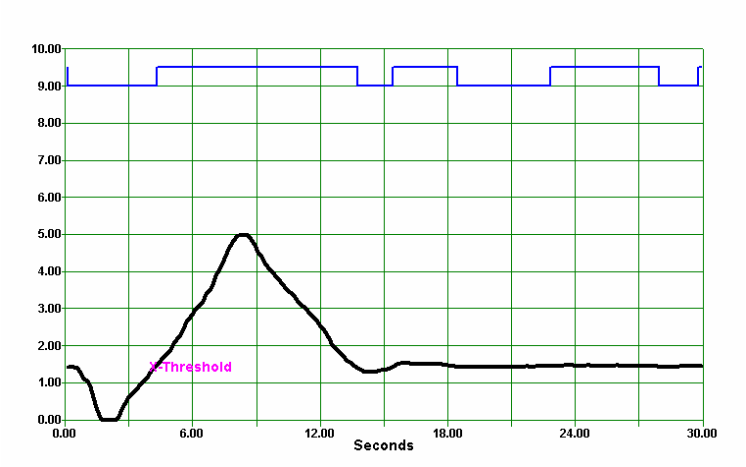


Figure 3-2
Logic Threshold Voltage Plot

Manufacturer data sheets provide guaranteed threshold voltage for a device. Legal HIGH and LOW values are considered above and below these levels.

- V_{IH} – Voltage In-High: Voltage above which assured to be HIGH on the input.
- V_{IL} – Voltage In-Low: Voltage below which assured to be LOW on the input.

For the BASIC Stamp:

- $V_{IH} = 2.0$ V
- $V_{IL} = 0.8$ V

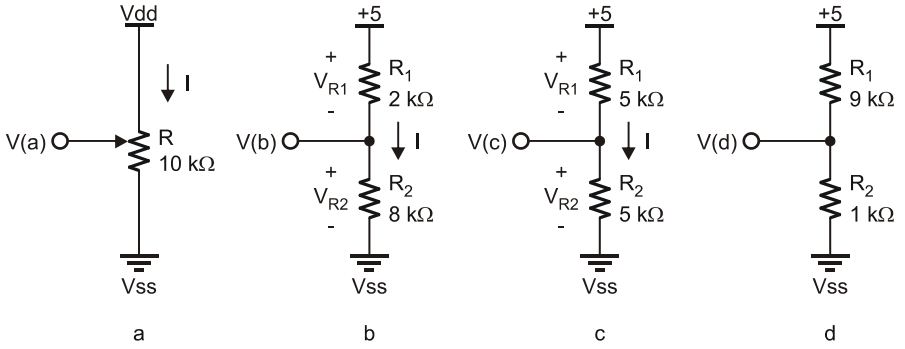
The BASIC Stamp has a very clear and repeatable threshold. Halfway between V_{IH} and V_{IL} is the 1.4 V TTL logic threshold at which the input will change sensed states. Not all digital devices have the same thresholds. Additionally, some inputs may employ Schmitt Triggers, or hysteresis, where threshold levels differ, depending on whether the voltage is increasing or decreasing to change states. The concept of hysteresis will be explored further in later chapters.

PROGRAM DISCUSSION

LED_1 in Figure 3-1 is used to indicate the state of P8: LED On = HIGH.

The potentiometer may simply be thought of as a variable voltage divider, such as in Figure 3-3. Keep in mind that each of these potentiometer voltages, V(a), V(b), V(c), and V(d), occurs as the knob on the potentiometer is turned to certain positions. Also keep in mind that each of the voltages can be applied to P8 in Figure 3-1.

Figure 3-3 10 kΩ Voltage Divider *(Do Not Build)*



The voltage output is a function of the voltage drop to ground across the lower portion of the potentiometer illustrated by R2 in Figure 3-3. Kirchoff's Voltage Law states that the "algebraic sum of the voltages in a series circuit must equal the supply voltage." The voltage dropped across R1 and R2 must equal the supply voltage, Vdd, of 5 V. Using Ohm's Law, the circuit may be analyzed:

Current = Voltage / Resistance or I = V/R

The total resistance must equal the value of the potentiometer, 10 kΩ. As such, the current flow through the resistor is:

I = V/R = 5 V/10 kΩ = 0.5 mA

The wiper only changes where it is tapping and splitting the total resistance. The voltage across R_2 for Figure 3-3b can be found by:

$V_{R2} = I(R_{R2}) = (0.5\ mA)(8\ k\Omega) = 4\ V$

How much voltage is dropped across R_1 in Figure 3-3b?

$V_{R1} = (IR_{R1} = (0.5\ mA)(2\ k\Omega)\ ) = 1\ V$

The total voltage of the series circuit in Figure 3-3b is:

$V_{R1} + V_{R2} = 1\ V + 4\ V = 5\ V$

Kirchoff's Voltage Law is upheld in that the sum of the voltages equals the supply voltage.

The voltage divider formula may also be used to find V_{R2}:

$V_{R2}=Vdd(R_{R2}/R_1 + R_2)) = 5\ V(8\ k\Omega/(8\ k\Omega + 10\ k\Omega) = 5\ V(8\ k\Omega/10\ k\Omega) = 4\ V$

For Figure 3-3c, what is V_{R2}? Without using any math, we can see the resistances are equal; therefore, the voltage drops must be equal to one-half of the supply voltage.

√ Calculate V_{R2} for Figure 3-3d.

This voltage is measured by the analog to digital converter and used as input to P8. When the voltage divider produces a voltage at or above 1.45 V, P8 senses a HIGH. When below 1.45 V, P8 senses a LOW.

The analog to digital converter is measuring the voltage, 0 to 5 V, and the BASIC Stamp is reading the ADC using the **`SHIFTIN`** instruction. The voltage is represented by 8-bits for a digital value from 0 to 255. **`DEBUG`** sends this value to StampPlot in the form of:

```
DEBUG "[ADC_DataIn,*,.0196]"
```

255 **x** .0196 = 4.998 V

The brackets instruct StampPlot to perform math on the data string prior to plotting it. For a digital value of 128, the string would be **`[128,*,.0196]`**.

StampPlot will perform the math, plot and display 128 * 0.0196 or 2.51, representing the voltage. Deeper discussions on ADCs and scaling data are in later chapters.

Saving digital input is performed within the routine to read the ADC, in an attempt to read both the ADC and digital input at the same time. The ADC value and the digital value may not track perfectly depending on how quickly the analog level changes.

Challenge 3-1: Reversing the Potentiometer Supply Voltage

- √ Note the direction of rotation needed to achieve an increasing voltage.
- √ Reverse the Vdd and Vss connections to the potentiometer.

What has changed when rotating the potentiometer? Explain why this change has occurred. (Hint: Refer to Figure 3-3a, and consider what occurs when Vss and Vdd are swapped).

ACTIVITY #2: NIGHT-LIGHT PROCESS

The input of the BASIC Stamp changing between HIGH and LOW at a fixed voltage can provide a simple means of control using analog signals. The input simply needs to go above and below the threshold level for the controller.

In this activity, a photoresistor will be added to the bottom of the variable resistor in the ADC circuit from Activity 1. This modified circuit creates a light-controlled voltage divider input to the BS2 for control of a light (the LED) at a certain level of darkness.

Parts Required

Circuit from Activity #1 (Figure 3-1 on page 46)
(1) Photoresistor

- √ Modify the circuit as shown in Figure 3-4 by adding the photoresistor between the potentiometer and Vss. Do not modify the ADC or LED portions of the circuit.
- √ Re-run the program DataMonitoring.bs2, if needed.
- √ Close the Debug Terminal.
- √ Run StampPlot macro sic_pc_data_monitoring.spm.
- √ Connect and plot.
- √ Allow the photoresistor to be exposed to ‘daylight levels’ of light.
- √ Adjust the potentiometer for a daylight voltage of 1.0 V.

√ Cast a shadow over the the photoresistor. What happens to voltage? When does your light energize?

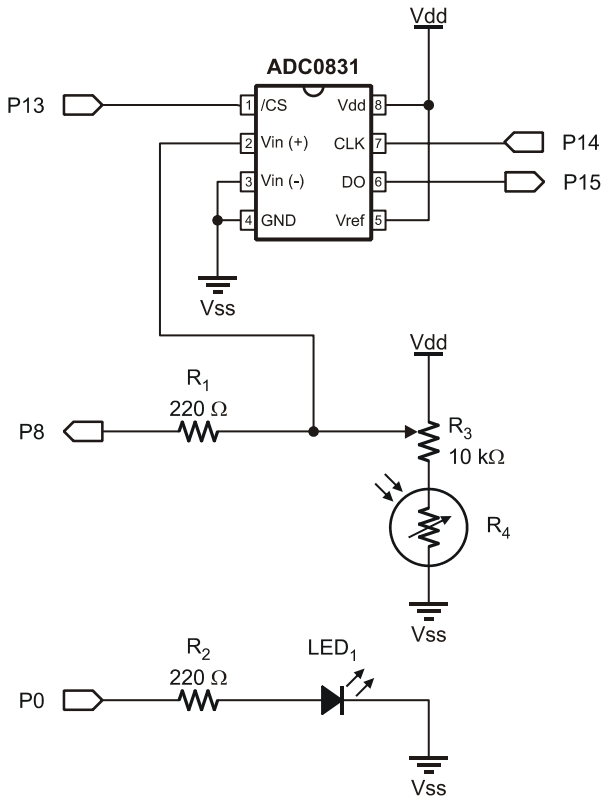


Figure 3-4
Photoresistor Added to Potentiometer Portion of the Circuit

In the CdS (Cadmium Sulfide) photoresistor, light photons excite electrons and allow them to flow more freely. This in turn changes the resistance. As light level increases, resistance decreases. In our case, as we shade the photoresistor, its resistance increases.

A greater voltage is dropped on the bottom half of the voltage divider (from the wiper to Vss). When the voltage drop to ground increases above the threshold voltage, the BASIC Stamp senses P8 as a logical HIGH.

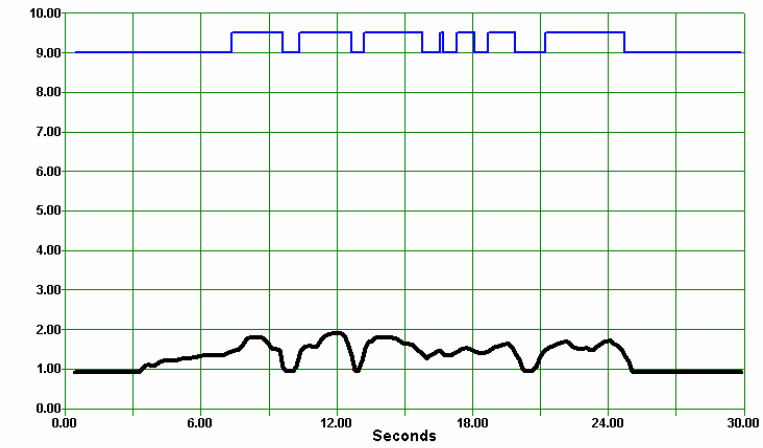


Figure 3-5
Night-Light Plot

Challenge 3-2: Photographic Darkroom Alarm

√ Modify the circuit to exceed the threshold when excess light falls on the sensor. Draw your circuit modifications. Discuss settings and results.

ACTIVITY #3: UNCOMMITED INPUTS AND CONDITIONING SWITCHES

An uncommitted input is one not dedicated to a voltage potential, is said to be "floating" and may not be sensed by the BASIC Stamp as a definite logic HIGH or LOW. Simply connecting a mechanical switch between the input and Vdd leaves the input floating when the switch is open. Current flow to Vdd or Vss is necessary to "commit" the input (when in the open condition). This exercise will help make this concept clear.

Parts Required

(1) ADC0831
(2) Resistors – 220 Ω
(1) Resistor – 1 kΩ
(1) Resistor – 10 kΩ
(1) Pushbutton – Normally Open
(1) LED – Red

√ Replace the photoresistor circuit Figure 3-4 with the pushbutton circuit as shown in Figure 3-6.
√ Re-run the program DataMonitoring.bs2
√ Close the Debug Terminal.
√ Run the StampPlot macro sic_pc_data_monitoring.spm.
√ Connect and plot.

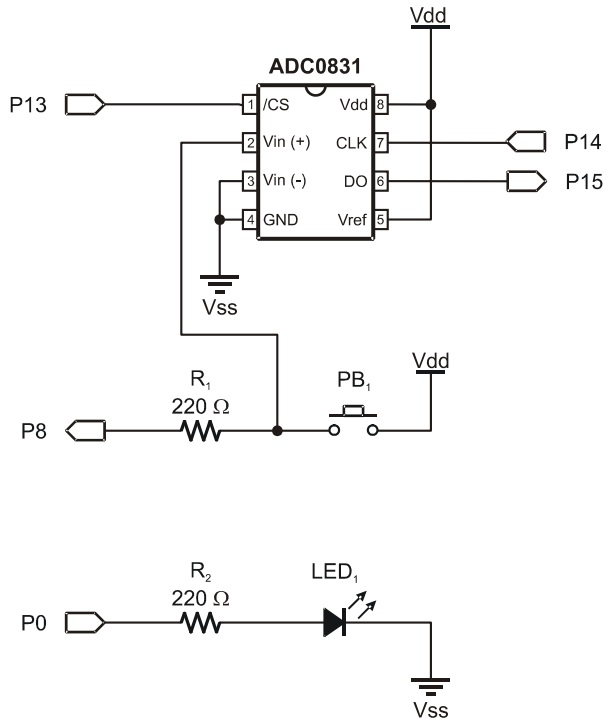


Figure 3-6
Uncommitted Pushbutton

√ Monitor the digital and analog values.
√ Momentarily touch the lead of R_1 on the pushbutton side. What occurs? (Results may vary depending on conditions.) Try rubbing your hair first to build up a static charge on your hand.
√ Press the pushbutton while momentarily touching the lead. What occurs?

Figure 3-7 is a sample plot of the results.

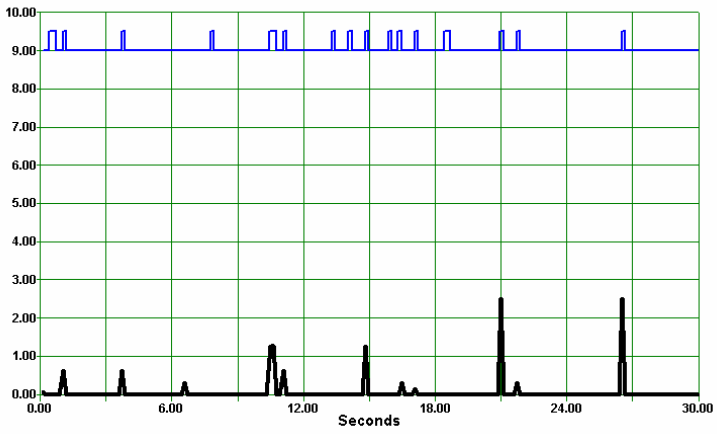


Figure 3-7
Plot of Uncommitted Input

When the pushbutton is pressed, current passes through the switch to register a solid HIGH value. The input is now committed and no longer floating.

When the normally-open (N.O.) pushbutton is not pressed, the input to P8 is floating and not committed to any voltage level. By touching the lead, the static electricity on your body is creating voltage spikes on the input above the threshold voltage. This noise causes the digital input to switch states. During process control, the uncommitted input may cause erratic and undesirable conditions. Again, the monitored analog voltage and digital input are not measured simultaneously and may not match HIGH and LOW values exactly.

- √ Place a 10 kΩ resistor from the P8 side of the pushbutton to Vss as shown in Figure 3-8.
- √ Test the circuit again. Are the results more stable?

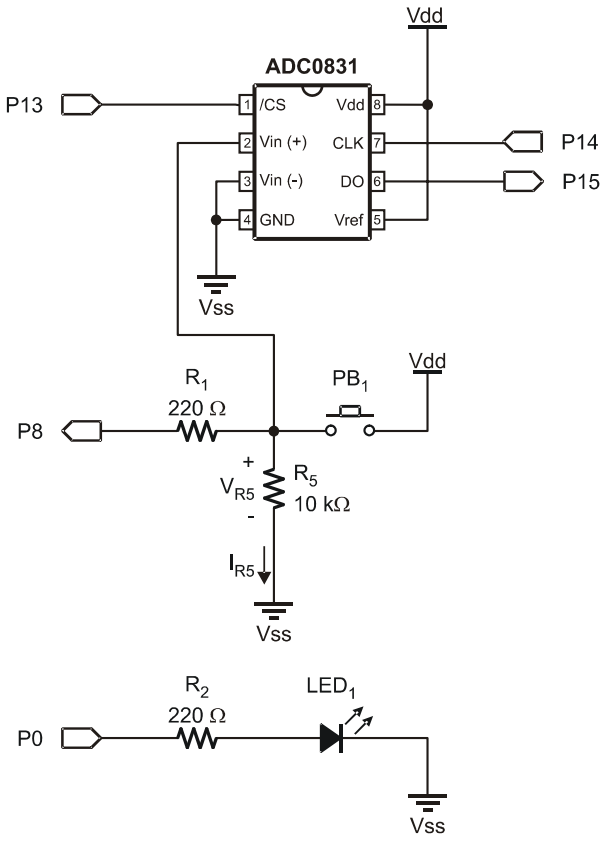


Figure 3-8
Active-High Pushbutton with Pull-Down Resistor

The pull-down resistor R_5 is used to force the input to a low voltage when the pushbutton is not pressed, keeping it LOW. When the button is pressed, current flows through R_5, allowing a HIGH to be sensed on P8. The pull-down resistor should be sized to prevent excessive current flow when the button is pressed.

$$I_{R5}=V_{R5}/R_5 = 5\text{ V}/10\text{ k}\Omega = 0.5\text{ mA}$$

Typical values for resistors used for this purpose are 1 kΩ, 10 kΩ or even 100 kΩ. A minimum amount of current is required, typically several microamps.

The pushbutton circuit is termed Active-High because when the button is active (pressed), the input state will be HIGH, a pull-down resistor is used to force the input LOW (when the pushbutton is not pressed).

Always check data sheets for the digital device in use. Different devices have different input voltage and current specifications. They may be damaged by static electricity when inputs are left uncommitted. Some devices may have internal pull-up resistors, alleviating the need of adding them to the input. TTL devices will typically assume a HIGH state if not connected, but this is not assured, especially during internal clocking or switching.

Challenge 3-3: Active-Low Pushbutton Circuit

Consider the circuit in Figure 3-9.

1. This circuit is termed Active-Low. Why?
2. Does this circuit use a Pull-Up or Pull-Down resistor? Why?
3. Reconfigure your circuit to match, and test. Discuss your results.

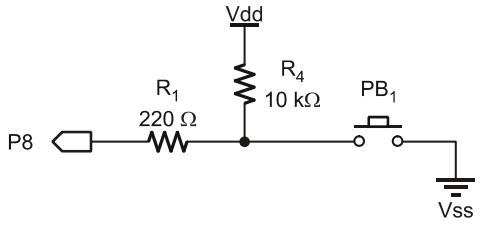


Figure 3-9
Active-Low
Pushbutton Circuit

ACTIVITY #4: THE TRANSISTOR AS A SWITCH

The transistor revolutionized electronics and is the basic building block in both analog and digital systems today. The Bipolar Junction Transistor (BJT) can be used as an amplifier to take a small analog signal, such as sound waves hitting a microphone, and amplifying that signal many times to be blasted out by speakers at rock concerts.

The transistor may also be used as a digital switch – ON or OFF. The BASIC Stamp and the microprocessor that runs your computer system are two examples of thousands, or

millions, of transistors working together as an Integrated Circuit (IC) to perform sophisticated operations.

As a switch, the transistor may be driven to a condition where it drops virtually no voltage and allows full current flow in a load. In this condition, it acts similar to a closed switch. No drive control applied to the transistor causes it to behave like an open switch. The transistor is semi-conductor configured to control current flow, allowing it to be fully on, fully off, or anywhere in between.

Figure 3-10 represents a typical discrete transistor. A BJT has 3 leads: Base (B), Emitter (E) and Collector (C). This transistor symbol tells us that it is an NPN.

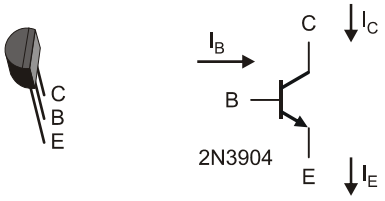


Figure 3-10
2N3904 NPN Transistor

The BJT is a current-controlled device. A small base-current (I_B) is used to control a much larger collector-current (I_C). I_B controls I_C.

The transistor has 3 operating regions and can be thought of as a water valve. When the valve (the base) is off, there will be no water flow in the pipe (collector-emitter). As you begin opening the valve, the amount of water flowing in the pipe is proportional to how far the valve is opened. At a certain point, opening the valve any further may not produce any appreciable change in water flow. Restrictions in the pipe and supply water pressure limit the flow rate.

There are three operating regions of the transistor as shown in Figure 3-11.

- Cutoff Region: Insufficient voltage on the base to produce appreciable current flow in the base and, therefore, no collector current. As an electronic switch, the collector to emitter is "open" and collector-current (I_C) is essentially zero.

- Active Region: The amount of current flow in the collector is directly proportional to the current flow in the base. The BJT is somewhere between fully-off and fully-on, controlling current flow. I_C is equal to the base-current

(I_B) multiplied by a gain factor called Beta (β) or, when dealing with strictly DC values, h_{FE}.

$I_C = I_B \times h_{FE}$.

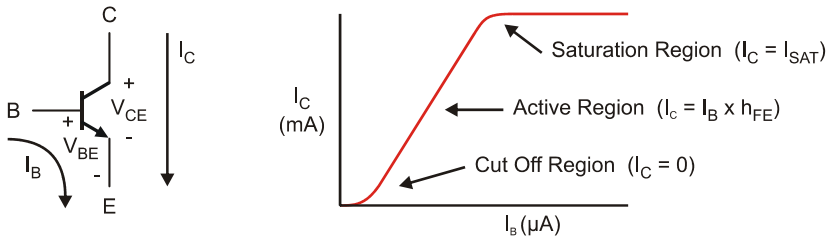
- Saturation Region: An increase in base current does not change the collector current. The electronic switch is closed. Resistance in the collector-emitter circuit limits I_C. The transistor is in saturation, and the collector current is termed saturation current (I_{SAT}).

In keeping with Kirchoff's Currrent Law (KCL) which states that the algebraic sum of the currents entering any node is zero:

$I_B + I_C - I_E = 0$

$I_E = I_B + I_C$

Figure 3-11 Transistor Current Flow and Characteristic Curve



All devices have limits and specifications for proper operation. Table 3-1 shows the pertinent specifications for the 2N3904. Some specifications are characteristics of the device, such as the voltage drop at the base-emitter junction (denoted by "Device" in the table). For example, the Collector to Emitter voltage drop is typically 0.3 V when current is flowing.

Other specifications are limitations imposed on the user to prevent damage to the device (denoted by "Use" in the table). For example, the maximum supply voltage to the device.

Table 3-1: 2N3904 Transistor Specifications		
Parameter	**Value**	**Meaning**
h_{FE} or ß	100 – 300	Current Gain I_C/I_B (Device)
I_C	200 mA	Maximum Collector Current Continuous (Use)
V_{CE}	0.3 V	Voltage dropped Collector-Emitter when in saturation (Device)
V_{BE}	0.65 V – 0.95 V	Voltage across Base-Emitter junction (Device)
V_{ECO}	40 V	Maximum Voltage Collector-Emitter (Use)
P_D	625 mW	Maximum Power Dissipation (Use)

Let's perform testing and calculations for a simple transistor circuit.

Parts Required

(1) ADC0831
(2) Resistors – 220 Ω
(1) Resistor – 1 kΩ
(1) Resistor – 47 kΩ
(1) Potentiometer – 10 kΩ
(1) Transistor – 2N3904
(1) LED – Red

- √ Construct the circuit shown in Figure 3-12.
- √ Run the program DataMonitoring.bs2
- √ Close the Debug Terminal.
- √ Run StampPlot macro sic_pc_data_monitoring.spm.
- √ Connect and plot.

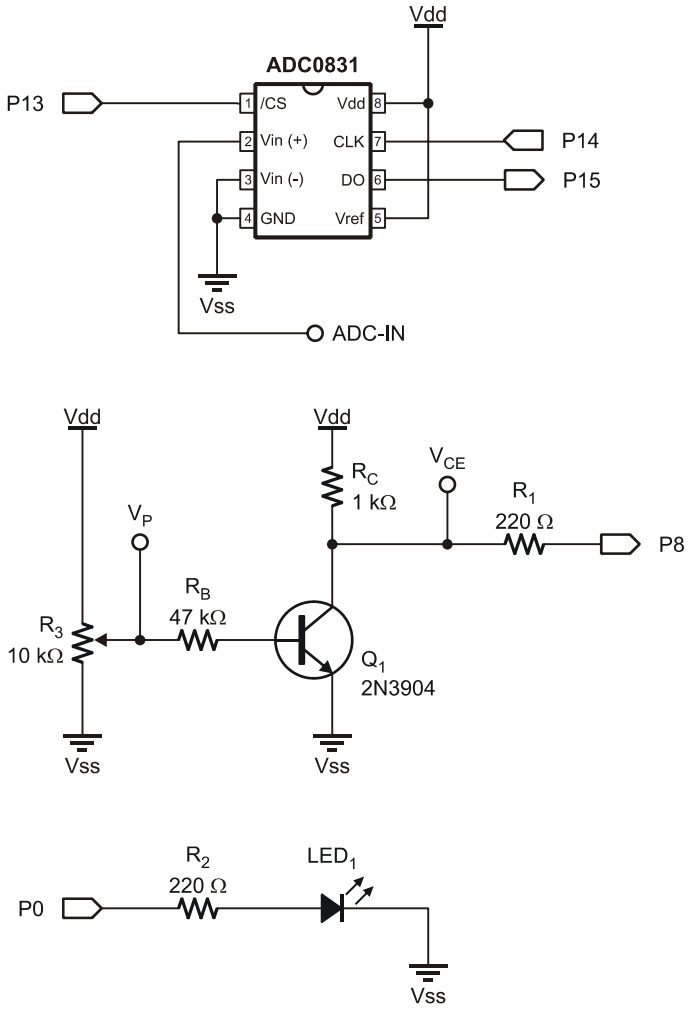


Figure 3-12 Transistor Monitoring Circuit

√ Connect ADC-IN to V_P. Adjust the potentiometer and note the direction that causes an increase in voltage (clockwise or counterclockwise).

√ Disconnect ADC-IN from V_P, and connect it to V_{CE}. What occurs as the potentiometer is adjusted? Does the increase in voltage follow V_P, or is it just

the opposite of it? Does the voltage at V_{CE} change over the full movement of the potentiometer or only over a portion of movement?

Analyzing what is occurring with reference to Figure 3-13:

- As V_P increases in voltage, the current through the base (I_B) increases.
- I_B increasing leads to increased current in the collector (I_C).
- As I_C is increased, more voltage is dropped across the collector resistor R_C.

  $V_{RC} = I_C R_C$

- The voltage drop across the transistor Collector to Emitter (V_{CE}) must decrease.

  $V_{CE} = 5\ V - V_{RC}$

- So, as V_p increases, V_{CE} decreases over a certain range.

As the potentiometer is adjusted from minimum to maximum, the transistor goes from cutoff through the active region to saturation.

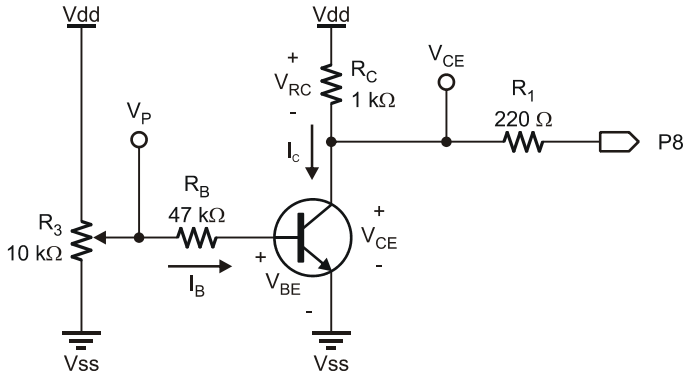


Figure 3-13

Voltage and Current Characteristics

Cutoff Region:

With insufficient base current, the transistor will essentially be off (an open switch) and acts as a very high resistance from collector to emitter. I_C will be zero, and the voltage at the output (V_{CE}) will be supply voltage, or Vdd in this case. This is also known as V_{CUTOFF}.

Linear Region:

In the linear region, the collector current, (I_C), is a function of the base-current multiplied by the DC gain of the transistor, h_{FE}. The 2N3904 can have a gain of 100 to 300. V_{CE} is a function of the current flow, creating a voltage drop across R_C. For example, given: $h_{FE} = 200$, I_B=10 µA, and a 5 V supply; calculate I_C, V_{RC} and V_{CE}.

$I_C = I_B\, h_{FE} = (10\ \mu A)(200) = 2000\ \mu A = 2\ mA.$
$V_{RC} = I_C\, R_C = (2\ mA)(1\ k\Omega) = 2\ V$
$V_{CE} = Vdd - V_{RC} = 5\ V - 2\ V = 3\ V$

Vcc vs. Vdd Officially, the collector voltage is called Vcc when working with BJTs which have a collector. Vdd is more properly used with Field Effect Transistors (FET) which have a drain instead of a collector. The BASIC Stamp is constructed with FETs, thus the Vdd designation is used.

What if I_B were 100 µA? What do the calculations show for I_C, V_{RC} and V_{CE}?

$I_C = I_B\, h_{FE} = (100\ \mu A)(200) = 20000\ \mu A = 20\ mA.$
$V_{RC} = I_C\, R_C = (20\ mA)(1\ k\Omega) = 20\ V$
$V_{CE} = Vdd - V_{RC} = 5\ V - 20\ V = -15\ V$

Does this make sense? How did we get 20 V across R_C with a supply of 5 V? The transistor had long gone into saturation when V_{CE} went down to 0 V and all of Vdd was applied across R_C.

Saturation Region:

As shown, there must be some limit to how much current can be developed in the collector. The current limit is based on the supply voltage and the value of R_C and is called saturation current (I_{SAT}).

$$I_{SAT} = Vdd/R_C = 5\ V/1\ k\Omega = 5\ mA.$$

At saturation, when the transistor is in full conduction, V_{CE} will be at the minimum, and the transistor will be conducting as much collector current as possible based on the restriction of R_C (fully-on or acting as a closed switch). At saturation, the collector to emitter junction of the transistor will always drop a small amount of voltage, typically 0.3 V. Therefore, I_{SAT} will be slightly less.

$$I_{SAT} = (Vdd - 0.3\ V)/R_C = 4.7\ V/1\ k\Omega = 4.77\ mA$$

Over a current range of 0 mA to 4.77 mA, the transistor will be in the active region. With an h_{FE} of 200, control is defined over a range of 0 mA to 23.5 μA for I_B.

$$I_B = I_C/h_{FE} = 4.77\ mA/200 = 23.5\ \mu A$$

Any base current above this value will cause the transistor to be in saturation and at maximum current.

Consider an increase in the value of R_C to 10 kΩ. Based on a 5 V supply, with an h_{FE} of 200, calculate values for I_{SAT} and I_B at saturation.

$$I_{SAT} = (Vdd - 0.3\ V)/R_C = 4.7\ V/10\ k\Omega = 0.477\ mA$$
$$I_B = I_C/h_{FE} = 0.477\ mA/200 = 2.35\ \mu A$$

Would the potentiometer require more or less voltage to drive the transistor into saturation? Since 10 times less current is required, 10 times less voltage is required at the base to drive the transistor into saturation.

Transistor Power Dissipation

Power is the work performed by a device or system per unit of time. In electronics power is measured in watts. Light bulbs are devices we commonly purchase based on the power output, such as 60 watt, 100 watt or even 200 watt bulbs. A light bulb's power is in the

amount of light that is produced (lumens) as well as the heat produced and dissipated to the air around it. Any device that has current flowing through it and a voltage drop across it coverts electricity into power. This power may be useful work (light, motion) or heat to be dissipated – which could also be useful at times, such as your clothes dryer.

DC Power Dissipation: The power dissipated by an element in a DC circuit is given by the voltage across the element multiplied by the current flowing through it:

Power = Voltage x Current

P = VI

When the power takes the form of heat, it must be dissipated from the device either by convection to the air or through other means. The CPU in your computer system consumes a large amount of power, and if the heat is not removed, damage to the CPU will quickly occur. Heat sinks provide heat conduction from the device, a greater area of cooling and often fans are added for forced convection to remove heat more efficiently.

Transistors, such as the ones in your CPU, have current flowing through them and have a voltage drop across them. Since a transistor operates in 3 distinct areas, when is the most power consumed?

- When in cutoff, the voltage drop is at maximum (V_{CUTOFF}), but current flow is minimum (theoretically 0).
- When in saturation, the current flow is at maximum (I_{SAT}) but the voltage drop is minimum (0.3 V).

Maximum power is used by the transistor when it is mid-point biased where V_{CE} is ½ V_{CUTOFF} and current is ½ I_{SAT} (these occur at the same time).

$$P_{Q1\ Max} = ½\ (Vdd) \times ½\ (I_{SAT})$$

For our circuit, the highest power can be calculated:

Vdd = 5 V, I_{SAT} = 5 mA

$P_{Q1\ Max}$ = (0.5)(5 V) x (0.5)(5 mA) = 6.25 mW

The maximum power that the 2N3904 can dissipate (P_D) is 200 mW without adding heat sinks and fans. We are well within the device's specifications.

Keep in mind that when using the transistor as a switch (in saturation or cutoff), the maximum power occurs only during the transition so that the power dissipated will be very low. The more frequently the transition occurs (frequency), the more the transistor is passing through the active region; therefore, the higher the average power that is being dissipated.

Challenge 3-4: Calculating Current and Power

Given a 2N3904 transistor with a collector supply voltage of 40 V and R_C of 500 Ω, calculate I_{SAT} and $P_{Q1\ MAX}$. Is there a concern based on power consumption and heat generation?

ACTIVITY #5: EFFECTS OF RESISTOR SIZING

R_C size plays a big role in the circuit by defining I_{SAT} and power. It also plays a role in the response of the circuit, how quickly it can respond to input changes.

Parts Required

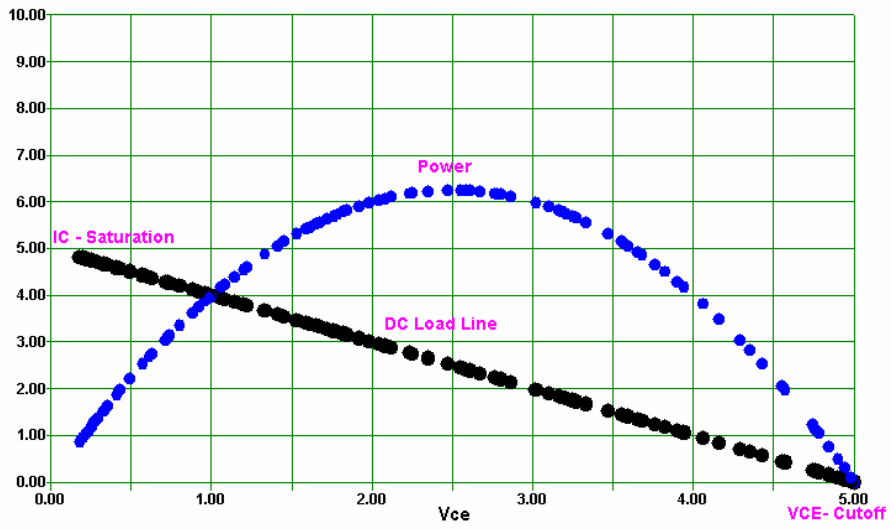
Circuit from Activity #4
(1) Resistor – 10 kΩ

- √ Begin with the circuit from Activity 3, Figure 3-12 with ADC-IN connected to V_{CE}.
- √ Run the program DataMonitoring.bs2.
- √ Run the StampPlot macro sic_pc_load_line.spm.
- √ Connect and plot.
- √ Adjust the potentiometer slowly between its minimum and maximum values.

StampPlot calculates and plots the DC Load Line for this value of R_C (1 kΩ). You can see a sample image in Figure 3-14.

The Load Line is a graphical representation of V_{CE} to I_C over the linear region. Note that when in cutoff, V_{CE} is at the supply voltage of 5 V. When in saturation, I_C is at maximum, based on the size of R_C and Vdd. Furthermore, based on I_C and V_{CE}, the power of the transistor (P_{Q1}) is plotted. Note the shape of the curves and when power is the maximum.

Figure 3-14 DC Load Line and Transistor Power



- √ Adjust the potentiometer slowly. Note the range over which the load line goes from cutoff to saturation.
- √ Replace R_C with a 10 kΩ resistor.
- √ In StampPlot, change the value of R_C to 10.
- √ Adjust the potentiometer slowly over its full range.

What happens to the load line? The value of I_{SAT} is smaller by a factor of 10. Power is reduced by a factor of 10 also.

Relative to using a 1 kΩ value for R_C (see Figure 3-12 and Figure 3-13), how much movement does it now take to control the transistor over its full active range? Since I_{SAT} is so much smaller, a much lower value of I_B required for saturation is defined, and thus a much lower value of V_P to reach saturation, which means the potentiometer needs to be rotated a smaller amount.

- √ Return R_C to a value of 1 kΩ.
- √ Change StampPlot RC value to 1.

- √ Reset the plot and re-acquaint yourself with the results of movement for the potentiometer.
- √ Replace R_B with the 10 kΩ resistor.
- √ Reset the plot and adjust the potentiometer very slowly to obtain a new plot.

Has the load line for the 1 kΩ resistor changed? How much movement of the potentiometer is required over the full active region as compared to the 47 kΩ? When R_B is reduced, the base current is higher for the same voltages. Because I_B is higher, I_C is higher over a smaller movement range of the potentiometer.

- √ Return R_B to a value of 47 kΩ.

Challenge 3-5: Considerations for a Transistor Switch

1. If the transistor were being used as a digital switch, would a higher or lower value of R_C be desirable based on the DC Load Line and input voltage response? Why?

2. What would the DC Load Line look like if a 100 kΩ resistor were used for R_C? Draw a plot of the Load Line and Power. Scale the plot accordingly for readability.

Other Considerations in Sizing Resistors

While it may appear that a high value of R_C is desirable for switching action of a transistor, this is not necessarily the case. The switching speed of the circuit is faster as the value of R_C decreases. The data sheets for the 2N3904 can lend credence to this. In later chapters we will see this in action. Figure 3-15 is a characteristic curve of rise time (t_r) vs. I_C.

As I_C increases (lower R_C), the rise time of the transistor decreases. This is approximately the time required to go from cutoff to saturation. The lower the rise time value, the faster the transistor can go from cutoff to saturation and vice-versa. Increasing R_C leads to increased sensitivity but slower switching.

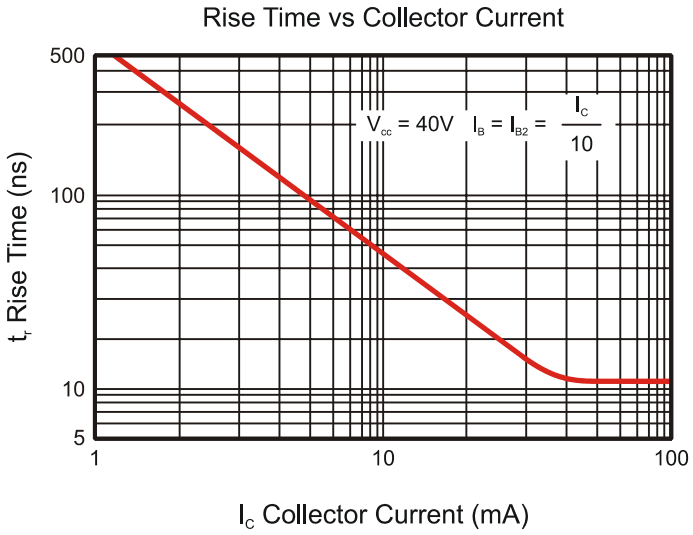


Figure 3-15
2N3904 Transistor
Characteristic Curve

Another effect is loading on the output. Take, for example, a voltmeter with an input impedance of 1 MΩ. If R_C is 1 MΩ and the transistor is in cutoff, what voltage will be read? It should be 5 V since the transistor is in cutoff, but R_C of the output and the meter input form a voltage divider as shown in Figure 3-16. The actual measured voltage would be 2.5 V. This is termed "loading".

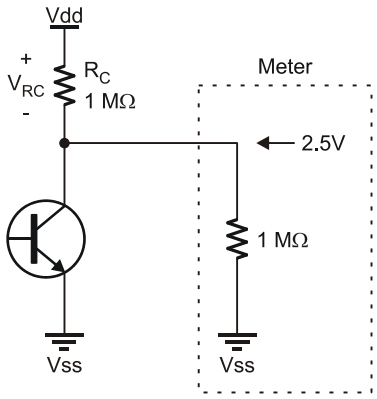


Figure 3-16
Loading Effects

Impedance is similar to resistance, but also takes into account AC characteristics, which will not be explored in this text.

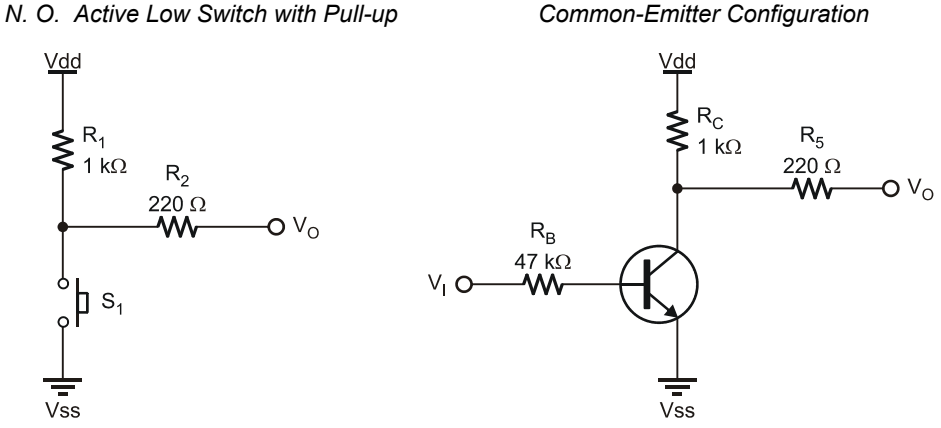
Ideally, the output impedance of a device should be zero, and the input impedance should be infinite to transfer the maximum voltage. Since we don't live in an ideal world, the input impedance should be at least 10 times the value of the output impedance between devices so that loading effects are not an issue. For example, if the input impedance of a device is 1 MΩ, the output impedance of the device supplying it should be no more than 100 kΩ.

Using the transistor as an input from another device, a higher base resistance is desirable to prevent loading and excessive current from the supplying device. Note that the control of the transistor is dependent on base current. As long as R_B is sized properly, this voltage may be smaller or larger than Vdd. This allows a means to interface devices operating at different supply voltages.

ACTIVITY #6: SWITCHING CONFIGURATION COMPARISONS

Figure 3-17 is a comparison of an Active-Low pushbutton switch and the common-emitter transistor circuit that we have been using.

Figure 3-17 Switch Equivalent Common-Emitter Configuration

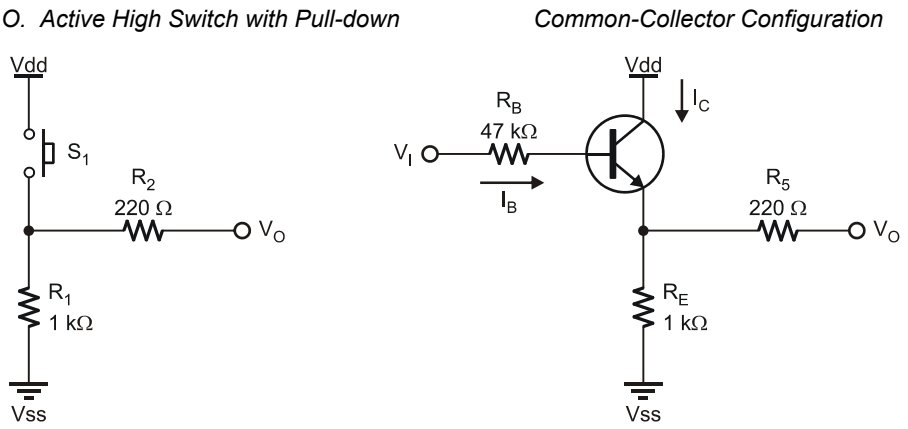


Note the similarities in operation:

- The pushbutton is activated by pressing it. The transistor is active (in saturation) when sufficient voltage is applied to V_I providing the base current required.
- Both circuits have a HIGH output when the switch is not active through the use of pull-up resistors.
- Both outputs go LOW when the device is activated.

Another transistor configuration is the common-collector. What is common (emitter, collector, or base) can be recognized by which terminal is NOT used by either the input or the output. Consider the comparison in Figure 3-18 to a pushbutton switch.

Figure 3-18 Switch Equivalent Common-Collector Configuration



The transistor will be in cutoff and act as an open-switch when insufficient V_I is applied to the transistor base. Just as with the pushbutton the output will be LOW through the pull-down resistor.

When V_I is a sufficient voltage, the base current will drive the transistor into saturation, acting as a closed-switch and Vdd will be felt on the output (minus 0.3 V dropped across the collector-emitter junction).

The common-collector is slightly more difficult to analyze, and we will only briefly discuss it. In the common-emitter the base current is calculated through:

$$I_B = (V_I - 0.7)/R_B$$

The collector current is determined from this value using h_{FE}.

$$I_C = I_B\, h_{FE}$$

In the common-collector, what determines the base current? Both R_B and R_E are in series from Vss to V_I, so a quick assumption would be $R_B + R_E$. But this is not correct because R_E lies on the emitter side, so the effects of h_{FE} must be considered. From the base's perspective, R_E is not 1 kΩ for this circuit but $h_{FE} \times R_E$.

$$I_B = (V_I - 0.7\text{ V})/(R_E\, h_{FE} + R_B)$$

Reducing the value of R_B to zero with a Vin of 5 V there may be insufficient base current to drive the transistor into saturation current. I_{SAT} is defined by Vdd/R_E in this configuration. Assuming $h_{FE} = 100$:

$$I_B = (V_I - 0.7)/(h_{FE}\, R_E + R_B)$$
$$I_B = (5\text{ V} - 0.7)/((100)(1\text{ k}\Omega) + 0) = .043\text{ mA}$$

$$I_C = I_B\, h_{FE} = 4.3\text{ mA}$$

$$I_{SAT} = (Vdd - 0.3)/R_E$$
$$I_{SAT} = (5\text{ V} - 0.3\text{ V})/1\text{ k}\Omega = 4.7\text{ mA}$$

With a V_I of 5 V and no R_B, I_C is almost at I_{SAT}, but not quite.

Challenge 3-6: Decreasing R_E in the Common-Collector Configuration

If R_E is decreases to 100 Ω, will a 5 V Vin and R_B of 47 kΩ be able to drive the transistor into saturation?

ACTIVITY #7: TYPICAL INDUSTRIAL SWITCHES

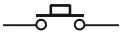
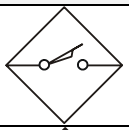
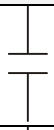
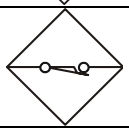
The pushbutton is just one of many switches available. Figure 3-19, Figure 3-20, and Table 3-2 show a variety of different switches commonly used in industrial applications. These switches may be either mechanical or electronic in their operation.

A mechanical switch, such as the pushbutton, opens or closes contacts to allow current to flow. When being used as an input to the BASIC Stamp, a pull-up or pull-down resistor is needed. The need for output conditioning is true of many electronic switches as well.

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 will eventually fatigue, or its contacts will arc, oxidize, or wear to the point of being unreliable.

Figure 3-19 Different Switches used in Industry

Table 3-2: Schematic Symbols for Various Industrial Switches

	Pushbutton	Mechanical Limit	Proximity Switch	Relay Contacts
Normally Open				
Normally Closed				

The proximity switches shown in Figure 3-20 are commonly used in industry to detect the presence of an object and operate on one of three principles:

- Inductive proximity switches sense a change in an oscillator's performance when metal objects are brought near. 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. 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 retro-reflective optical switches, an object moving into the switch's range may reflect the light beam back to the sensor. Through-beam optical switches are set up such that the object blocks the light beam going between the light source and the receiver.

Figure 3-20
Inductive, Capacitive and Optical Proximity Switches

A common final output stage of an electronic switch is shown in Figure 3-21.

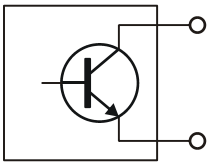


Figure 3-21
Typical Electronic Switch Output

This configuration allows maximum flexibility for engineers in integrating it into their systems. The output may be configured as common-emitter or common-collector, and the resistors sized appropriately.

Other devices may simply output a digital HIGH and LOW. If the device does not use the same supply, ensure that output voltages are compatible with the BASIC Stamp. If the switch is powered from another supply, the grounds will need to be connected together. Figure 3-22 and Figure 3-23 illustrate various methods of interfacing digital devices.

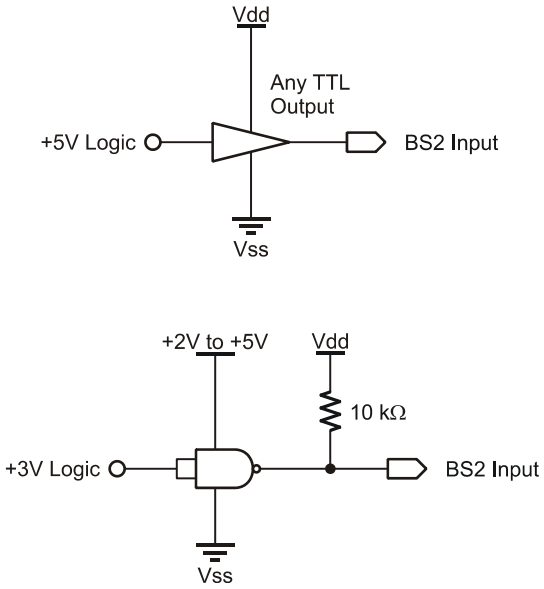


Figure 3-22
Digital Interface Circuits

Top: Standard TTL to BASIC Stamp

Bottom: Low Voltage Logic to BASIC Stamp

Figure 3-22 (top): TTL and CMOS logic inputs powered from a +5 volt supply can be applied directly to the BASIC Stamp 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).

Figure 3-22 (bottom): Low-voltage (+3 V) devices can be interfaced using a 74HCT03, or similar open-drain devices, with a pull-up resistor to the BASIC Stamp module's +5 volt supply. Supply the chip with the low-voltage supply and make the grounds common.

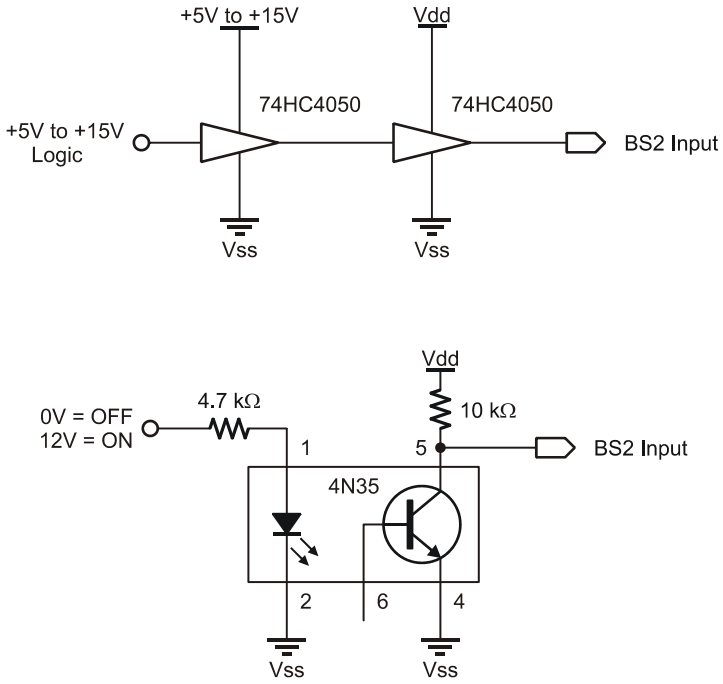


Figure 3-23
Digital Interface Circuits

Top: High Voltage Logic to BASIC Stamp

Bottom: Optocoupler to BASIC Stamp

Figure 3-23 (top): 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.

Figure 3-23 (bottom): 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 because they do not need to share a common-ground. This isolation provides effective protection of each circuit in case of an electrical failure of the other.

The optical reflective switch will be explored further in Chapter 4.

Challenge 3-7: Wiring a Relay

The relay can use high voltage, AC or DC, to energize a solenoid that closes or opens electrically isolated mechanical contacts that can be used as input to the BASIC Stamp. Signal conditioning these contacts is similar to the use of pull-up or pull-down resistors.

√ Wire the contacts (by drawing) in Figure 3-20 so the BASIC Stamp senses a HIGH when S1 is closed, which energizes the solenoid and magnetically opens the normally closed relay contacts labeled K1.

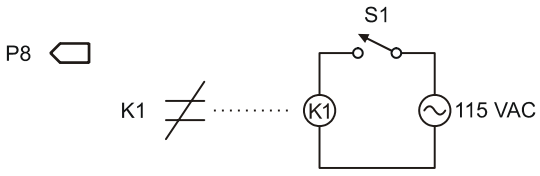


Figure 3-24
Sensing 115 V with a Relay

(Drawing to be completed by student – do not build)

CONCLUSION

When acting as inputs, the BASIC Stamp typically senses a digital HIGH value for any voltage above approximately 1.4 V and a digital LOW below 1.4 V. An input to the BASIC Stamp does not necessarily need to be 5 V or 0 V, but must at least cross this threshold voltage. Input voltages above 6 V will damage the BASIC Stamp.

BASIC Stamp inputs are uncommitted. That is, they are neither HIGH nor LOW without an input committing them to the positive voltage or down to ground respectively. Most mechanical switches require a pull-up or pull-down resistor for an Active-Low or Active-High configuration respectively. Mechanical and electronic switches often require proper conditioning.

Bipolar Junction Transistors (BJT) are current controlled devices that can operate as current amplifiers or electronic switches. The base current controls the current in the collector. With no base current, the transistor will be cutoff and act as an open switch. With sufficient base current, the transistor's collector current will be at saturation, and the transistor will act as a closed switch. The saturation current is a function of the supply

voltage and the resistance of the collector for the configuration studied. The DC Load Line is a graphical representation of output voltage and collector current over the linear range.

The value of R_C in a common-emitter configuration determines the saturation current, and thus, the base current required to drive the BJT into saturation. The higher the value of R_C the more sensitive the transistor will be to base current. But high values of R_C also limit response and switching speeds of the transistor. The maximum power dissipated by the transistor occurs when it is conducting ½ of saturation current, which leads to heating.

SOLUTIONS TO CHAPTER 3 CHALLENGES

Challenge 3-1 Solution

When the potentiometer supply leads are reversed, rotation will result in the opposite direction of voltage change as compared to previously. For example, when rotated clock-wise, the wiper of the potentiometer will be closer to Vdd as opposed to Vss previously.

Challenge 3-2 Solution

Placing the photoresistor on the Vdd side of the potentiometer modifies the circuit for a darkroom sensor. When more light falls on it, the resistance decreases dropping more voltage across the bottom half causing voltage to increase. The potentiometer must again be adjusted for the desired light/dark threshold in the darkroom.

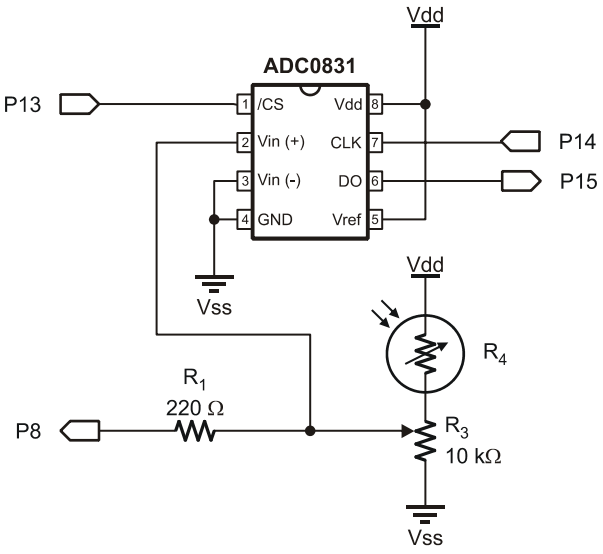


Figure 3-25
Photographic Darkroom Alarm Circuit Solution

Challenge 3-3 Solution

1. The circuit in the figure is termed Active-Low because when the button is pressed (active) a connection is made to Vdd.
2. A pull-up resistor used to keep the input high (up = high) when the switch is not active.
3. Your circuit should look like Figure 3-26. Pressing the button after reconfiguring should have caused the digital plot to go low.

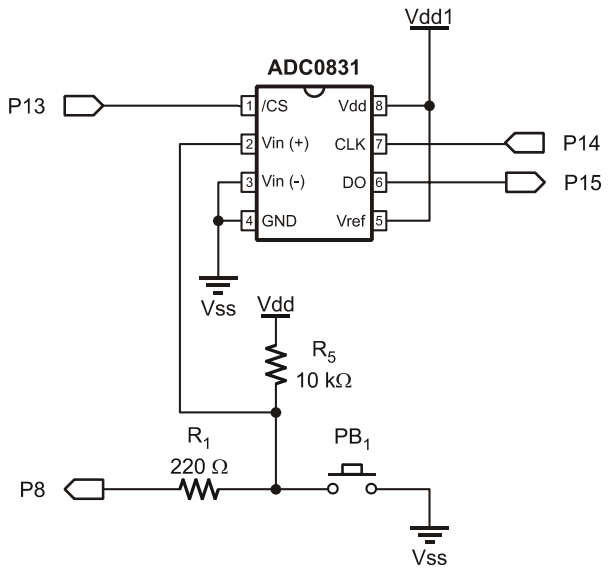


Figure 3-26
Active-Low
Pushbutton Solution

Challenge 3-4 Solution

I_{SAT} = (40 V− 0.3 V)/500Ω = 79.4 mA

Maximum continuous is 200 mA, so OK.

$P_{Q1\ MAX}$ = ½ I_{SAT} x ½ V_{dd} = (0.5)(79.4 mA) x (0.5)(40 V) = 794 mW

Maximum is 625 mW so heat sinking is required.

Challenge 3-5 Solution

1. A higher value of R_C is desirable based on the response to the input (sensitivity). Lower I_B is required, therefore less in change of V_{IN} and shorter time spent in linear region in going from saturation to cutoff.
2. I_{SAT} = (Vdd − 0.3)/R_C = 4.7 V/100 kΩ = 0.047 mA
   $P_{Q1\ MAX}$ = ½ I_{SAT} x ½ V_{dd} = (0.5)(0.047 mA) x (0.5)(5 V) = .1175 mW

   Load line would go to 0.047 on Y-axis to 5 V (cutoff) on X-axis.

Challenge 3-6 Solution

$I_B = (V_I - 0.7)/(h_{FE} R_E + R_B)$
$I_B = (5\ V - 0.7)/((100)(100\ \Omega) + 47\ k\Omega) = 0.075\ mA$

$I_C = I_B \times h_{FE} = (0.075\ mA)(100) = 7.5\ mA$

$I_{SAT} = (Vdd - 0.3)/R_E$
$I_{SAT} = (5\ V - 0.3\ V)/100\ \Omega = 4.7\ mA$

$I_C > I_{SAT}$. Saturation current can be reached.

Challenge 3-7 Solution

- When S1 is open, the relay is de-energized and contacts K1 are closed. P8 senses LOW.
- When S1 is closed, the relay is energized and contacts K1 open. P8 senses HIGH.

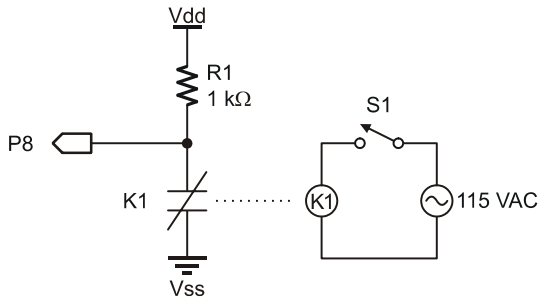


Figure 3-27
Completed Drawing

Do Not Build

Chapter 4: Sequential Processes and Optical Switches

Sequential processes are those that follow a defined sequence of actions based on events. Events involved in a sequential process may be time based, where different actions occur with defined intervals of time. One example is a clothes dryer, where the sequence of drying occurs at set intervals. The dryer applies heat and rotates the drum for 50 minutes. After that time the heat is turned off while the drum continues to turn, providing a cool-down cycle for 5 minutes. After 5 minutes of cooling, the drum stops rotating and the buzzer sounds.

The events may also be input based, such as a button being pressed or a sensor detecting a condition which sets a sequence of actions in motion or allows the sequence to continue. Consider a clothes washer. What types of events are involved?

- The start button is pressed.
- Water valves open and the tub fills to a defined level based on load size with a detector sensing water level.
- The agitator begins to cycle.
- Agitation stops and the spin and drain cycle begins.
- And so on....

Which of the above events are most likely input based, and which are time based? Filling the machine is an input event since a sensor is used to detect the water level. The remaining are most likely time based.

There exists a wide variety of input devices and sensors used to detect the absence or presence of an object or material. Many of these are non-contact in that there is no physical contact between the sensor and the condition being sensed. Examples include capacitance, inductance and magnetic. A very popular class of non-contact sensors are optical; they use light for detection or transmission of data. In many cases, the wavelength of light is in the infrared (IR) range not visible to the human eye.

Some examples of optical sensors include:

- Using a light beam to detect presence of an object or person at a moderate distance (several feet).
- Using an emitter and detector pair for very close detection, such as motion of your mouse, the passing of paper in a printer, or the opening of a printer cover.
- Using reflected light to measure the rotation of a shaft.
- Using light to transmit data at a reasonably close range (TV remote) or perhaps a very long range (fiber-optic telecommunications).

This chapter uses a close-range IR optical sensor to demonstrate operation and signal conditioning. The sensor is used as part of sequential processes for detection and event input.

ACTIVITY #1: CONNECTING AND TESTING THE OPTO-REFLECTIVE SWITCH

The QRB1114 shown in Figure 4-1 is an opto-reflective switch. The actual switching circuit is a phototransistor. Light hitting the transistor causes a current flow in the base-emitter, which in turn causes an amplified current flow in the collector-emitter. The package also contains an infrared LED to use as a light source (emitter). The LED emits infrared light (IR) not visible to the human eye, and the phototransistor is most sensitive to this wavelength of light.

In this figure, (E) is the emitter of the phototransistor, and (C) is the collector. (A) stands for the anode of the IR LED, and (K) marks the cathode.

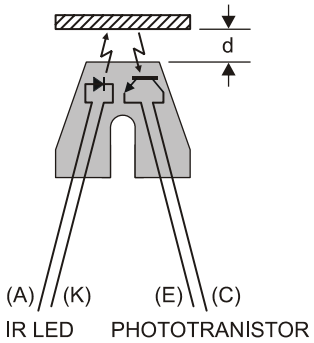


Figure 4-1
Opto-Reflective Switch

The IR LED emitter and phototransistor are combined in a single package. The pair are angled for maximum reflection from a surface at a distance of 0.15 inches (3.8 mm) or about 1/6 of an inch. This non-contact switch responds to an object passing in front of its window within the range of detection.

From the study of transistors in Chapter 3, a correct configuration is required for this switch to be sensed as a digital input.

- The uncommitted collector will require a pull-up.
- The collector resistor must be sized such that the output has transitions between light and dark conditions above and below the BASIC Stamp threshold voltage.
- The emitter is connected to ground.
- Sufficient base current exists. In this case it would be due to IR LED emissions reflecting off an object.

The IR LED is rated at a maximum current of 40 mA. Given that the IR LED drops around 1.5 V, this leaves 3.5 V for calculating the size of the resistor needed to limit current to the IR LED.

$$R = V/I = (Vdd - V_{LED})/I_{LED} = (5.0\ V - 1.5\ V)/40\ mA = 87.5\ \Omega.$$

A 100 Ω current-limiting resistor will be used. Since the base current (dependent on IR radiation) and gain are not quantifiable, determining collector current requires some experimentation.

Parts Required

(1) Resistor – 100 Ω
(2) Resistors – 220 Ω
(1) Resistor – 10 kΩ
(1) Resistor – 100 kΩ
(1) Resistor – 1 MΩ
(1) Opto-Reflective Switch – QRB1114
(1) ADC0831
(1) LED – Red

√ Construct the circuit shown by the schematic in Figure 4-2 and the wiring diagram in Figure 4-3. Note that Q_1 and the IR LED are both inside the QRB1114. Set aside the 100 kΩ and 1 MΩ resistors for now.

√ Mount the QRB1114 as shown. Bend the device so that it is parallel to the table surface.

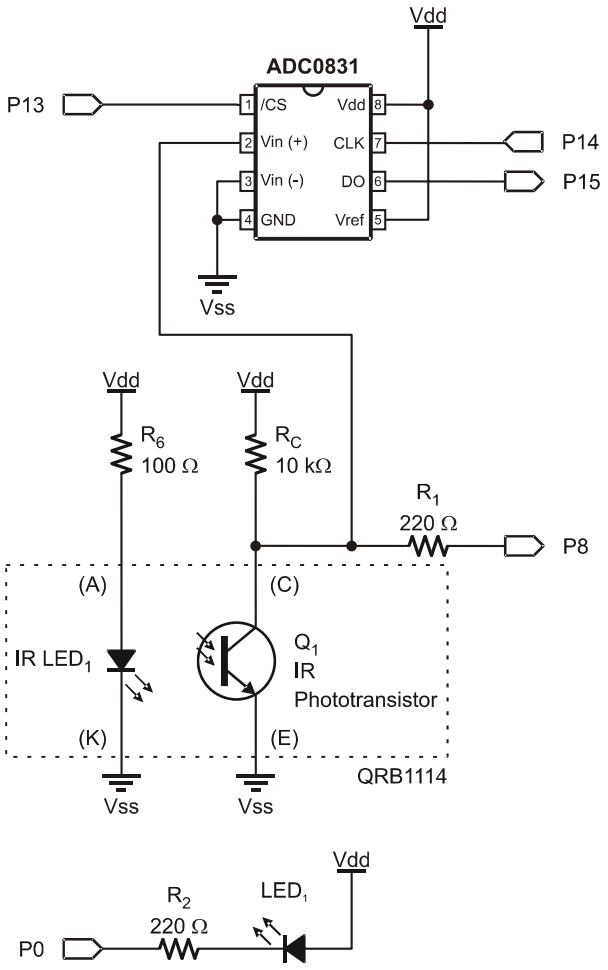


Figure 4-2
Opto-Reflective Switch Monitoring

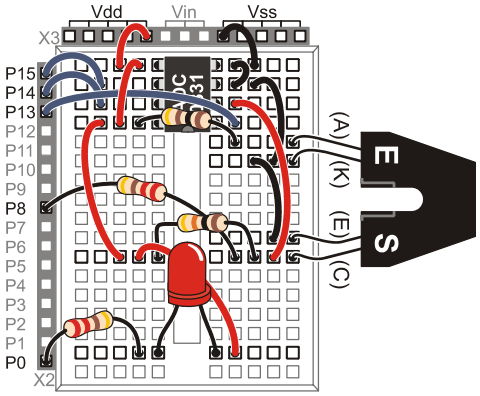


Figure 4-3
Opto-Reflective Switch Wiring Diagram

NOTE: the ACD0831 is installed "upside-down" with Pin 1 at the lower right in this picture.

√ Run the program DataMonitoring.bs2 from page 46.
√ Close the Debug Terminal.
√ Run StampPlot macro sic_pc_opto_plot.spm.
√ Connect and plot. Nothing will show or update initially on the main plotting area.
√ Move your hand just a few inches toward and away from the QRB1114 sensor.

The box labeled VCE should show different readings as you move your hand. If the VCE readings do not change, something is amiss, and it is time to troubleshoot:

√ Check StampPlot to make sure you are connected, and plotting.
√ Try to reset or refresh the plot. (From the menus, select Plot → Reset Plot.)
√ Check your wiring to ensure you have not missed any connections.
√ Make sure the 100 Ω and 10 kΩ resistors are in their correct places.
√ Once the VCE readings update, move on to the next step.

When all is working, move on:

√ Verify that 10 is selected from the RC(K) drop-down menu in StampPlot.
√ Using the ruler in Figure 4-4, position the front of the opto-reflective switch at the 0 CM position as shown in Figure 4-4. For best results, use a book or another object to raise the level of the ruler to be even with the device.

√ Use a folded piece of white paper or a white 3 x 5 card for the reflective object. For maximum reflection, fold it so that the surface facing the detector is as vertical as possible.
√ Start at 50 mm (5 cm) and move the paper towards the switch. Watch the VCE reading to see how the voltage drops as the paper moves closer to the switch.

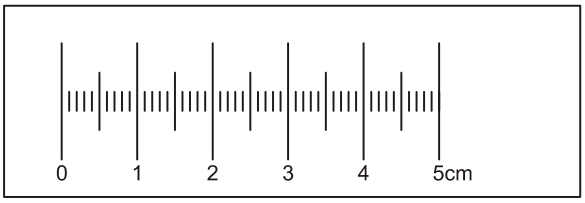
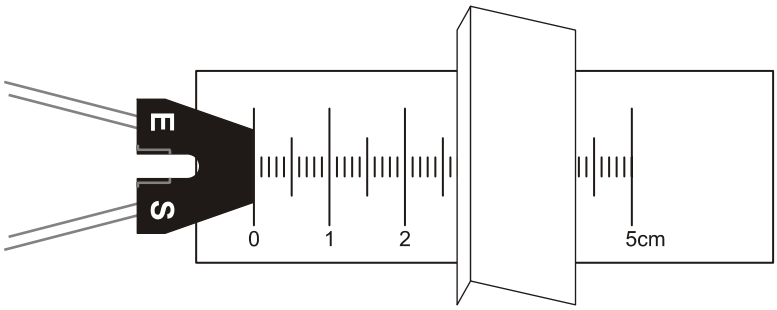


Figure 4-4
Reflection Ruler

Duplicate for cutting out can be found in Appendix A.

Figure 4-5 Positioning of Switch and White Paper



This StampPlot macro gives the ability to make a plot of the voltage readings. See Figure 4-6 for an example plot. The individual points on the graph are made by typing a distance in the mm box, then clicking the Plot button. The current VCE reading will be plotted vs the distance entered. As points are input in this manner, the macro connects each point with a straight line.

Making a Plot

√ Start at 50 mm (5 cm) and move the paper towards the switch. Watch the VCE box for slight drop, maybe 0.1 V, in voltage, or just move 5 mm closer.
√ Enter the distance in the 'mm' box.
√ Click Plot to plot your reading.

- √ Repeat for at least 10 readings, moving a little closer to the switch each time. Each time you click Plot, a new point will appear on the graph.
- √ Be sure to include the lowest voltage achieved and at least one closer where voltage begins to rise again. The change in voltage may not be very significant (lowest voltage around 4.5 V).

Label your plot:

- √ In the bottom-right corner of the StampPlot screen, it says "Double-Click Plot to add Text," with a drop-down text box underneath. (You may have to expand your window size to see this area.)
- √ Click in the text box at the bottom right of the StampPlot screen.
- √ Highlight and delete anything that is in there, and type in a new phrase: "10 k Ohm response".
- √ Now double-click on the plot where you want to place the label, and then your text will show up on the plot.
- √ Save or print your plot.

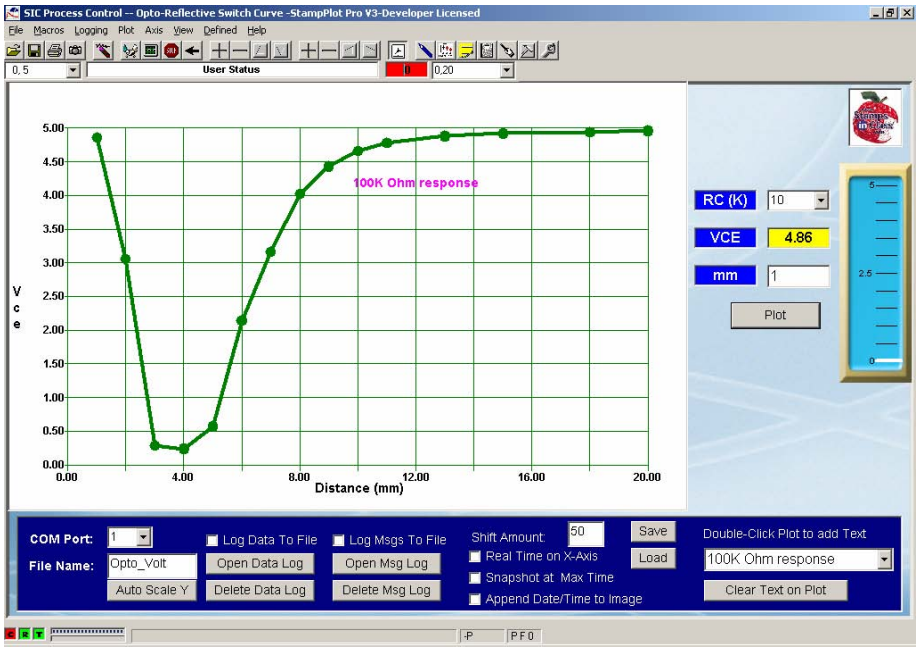
Did the voltage drop far enough to register as a LOW, that is, does it drop below the BASIC Stamp TTL threshold of 1.4 V? With some resistor values, it may drop below 1.4 V, but with others it may never drop below 4 V, indicating that there is not sufficient current to reach saturation. Some choices may be to use a more reflective material, to increase the power output of the LED (not an option – near 40 mA already), or to increase the size of R_C. Let's try some other resistors

- √ Replace R_C with a 100 kΩ resistor.
- √ Select 100 from the RC(K) drop-down menu.
- √ Repeat the activity, and label the resulting line "100 K Ohm response"
- √ Replace R_C with a 1 MΩ resistor.
- √ Select 1000 from the RC(K) drop-down menu.
- √ Repeat the activity, and label the resulting line "1 M Ohm response"

Using the 1 MΩ resistor, the cutoff voltage may not read 5 V. The value of RC is approaching the input impedance value for the ADC and the BASIC Stamp causing loading effects. As long as the voltage is above 2.0 V (V_{IH} for the BASIC Stamp) it will not be an issue.

Based on the value that provides the best response, which resistor would you use for a digital switch? Figure 4-6 shows the results of our test with the 100 kΩ value. Note as the distance is reduced (X-axis), the voltage at R_C decreases (Y-Axis). But after a certain point, there is insufficient angle for reflection and voltage rises once again.

Figure 4-6 Opto-Reflector Response with 100 kΩ RC



Challenge 4-1: Determining Distances with Different Materials

√ Using the 1 MΩ R_C value, test at least 5 surfaces, including different colors and surface finishes. At what distance from the opto-reflective switch is a voltage of 1.5 V reached? Complete Table 4-1. Be sure to include the test reflector (white paper) used in this activity as a reference.

Table 4-1: Surface/Distance Combinations for 1.5 Volts	
Surface Material	**Distance (mm)**
White Paper Test Reflector	

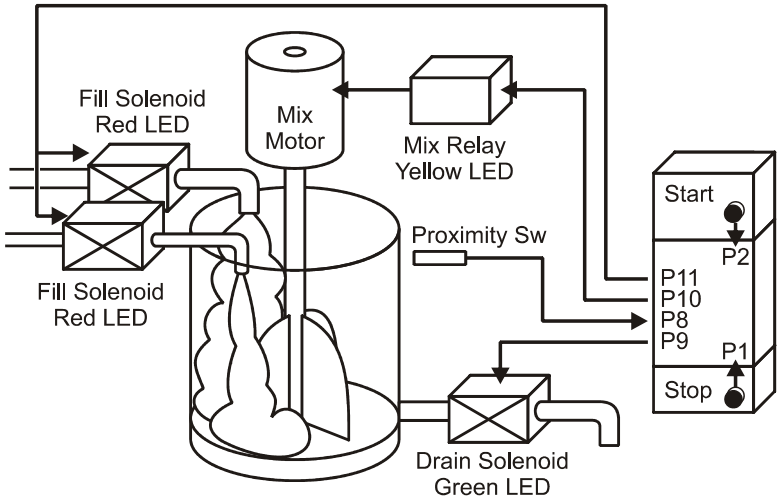
ACTIVITY #2: BATCH OR SEQUENTIAL PROCESS CONTROL

A batch, or sequential process, uses input signals and timing to perform a straightforward set of operations. A good example of a sequential process is a traffic light. A sensor embedded in the road detects a vehicle, and a sequence begins to switch the lights, controlling the flow of traffic.

In this experiment, a mix process will be simulated using the opto-reflector and pushbuttons for inputs. The opto-reflective switch will be used to sense level. A more realistic approach for sensing the level would be an inductive proximity sensor that may be adjusted to 'ignore' the container's mass, and detect level inside an opaque container. LEDs will be used to indicate the status of the process.

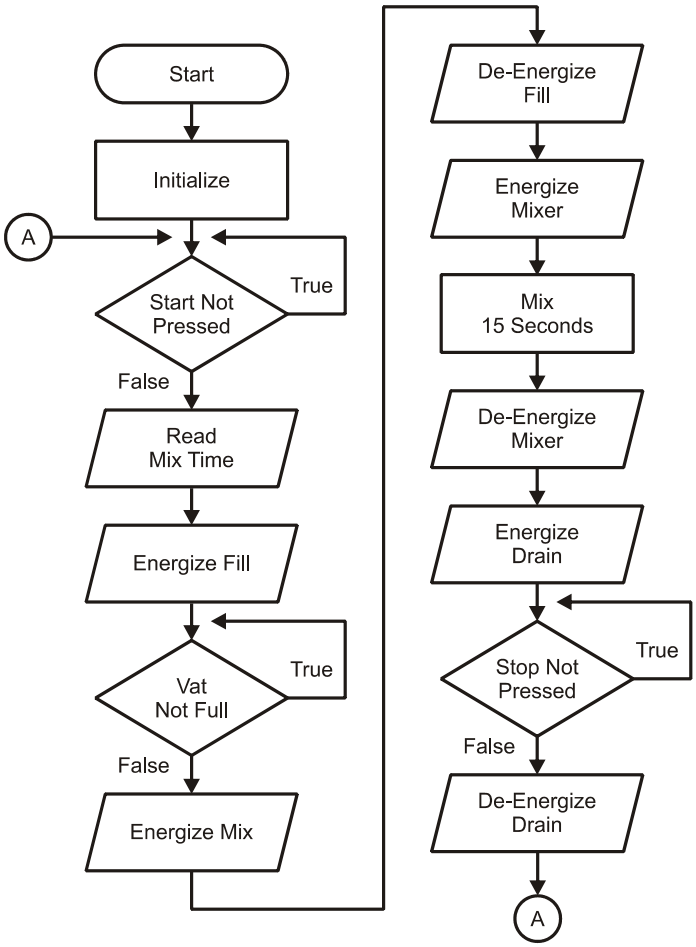
The following is an operational description of the batch process illustrated in Figure 4-7. A flowchart of the process is in Figure 4-8.

Figure 4-7 Batch Mix System



- When the Start button is pressed, the vat will begin to fill from both sources.
- The proximity switch detects when the vat is full.
- Once full, the fill valve will be closed and mixing will occur for a specified time.
- When mixing is complete, the drain valve will open and remain so until the operator presses the Stop button.

Figure 4-8 Batch Sequence Flowchart



Parts Required

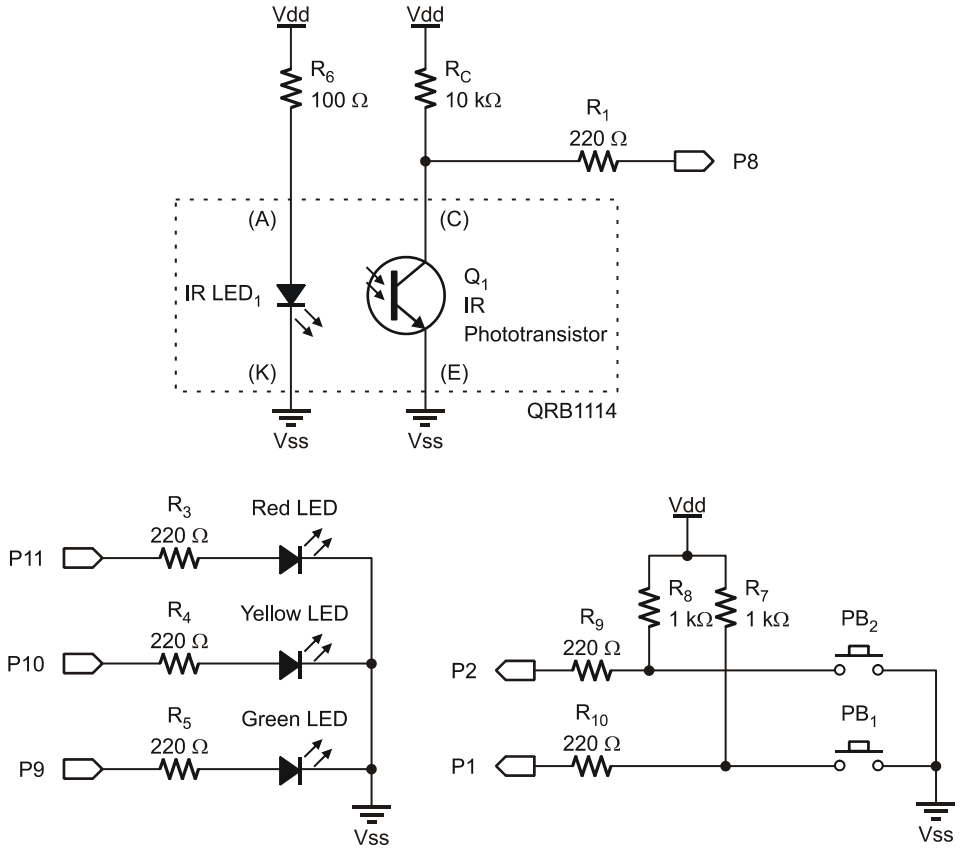
(1) Resistor – 100 Ω
(6) Resistors – 220 Ω
(2) Resistors – 1 kΩ
(1) Resistor – 10 kΩ
(1) Opto-Reflective Switch – QRB1114

(1) LED – Green
(1) LED – Yellow
(1) LED – Red
(2) Pushbuttons

√ Construct the circuit shown in Figure 4-9 and Figure 4-10. Leave the ADC circuit on your board for later use.

Figure 4-9 Batch Sequence Circuit - Schematic

Note: The ADC circuit isn't needed so it isn't shown, but leave it on the board for future use.



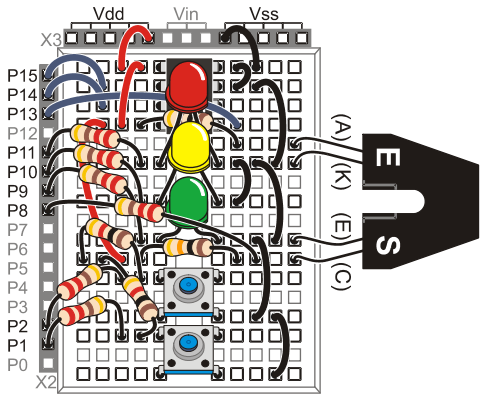


Figure 4-10
Batch Sequence Wiring Diagram

Please note that the ADC circuit is not used in this activity but is left on for future use.

4

Example program: BatchMix.bs2

√ Enter and run program BatchMix.bs2.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - BatchMix.bs2
' Control System for filling, mixing and draining a vat
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
Stop_SW         PIN     1           ' PB1
Start_SW        PIN     2           ' PB2
Opto_SW         PIN     8           ' Opto-Reflector

Drain           PIN     9           ' Green LED
Mix             PIN     10          ' Yellow LED
Fill            PIN     11          ' Red LED

Total_Mixed     VAR     Word        ' Total amount of solution mixed
Mix_Seconds     VAR     Byte        ' Time to mix
X               VAR     Byte        ' General Counting

Vat_Volume      CON     25          ' Size of Vat

' -----[ Initialization ]-----------------------------------------------------
LOW Drain                           ' Set initial states
LOW Mix
LOW Fill
```

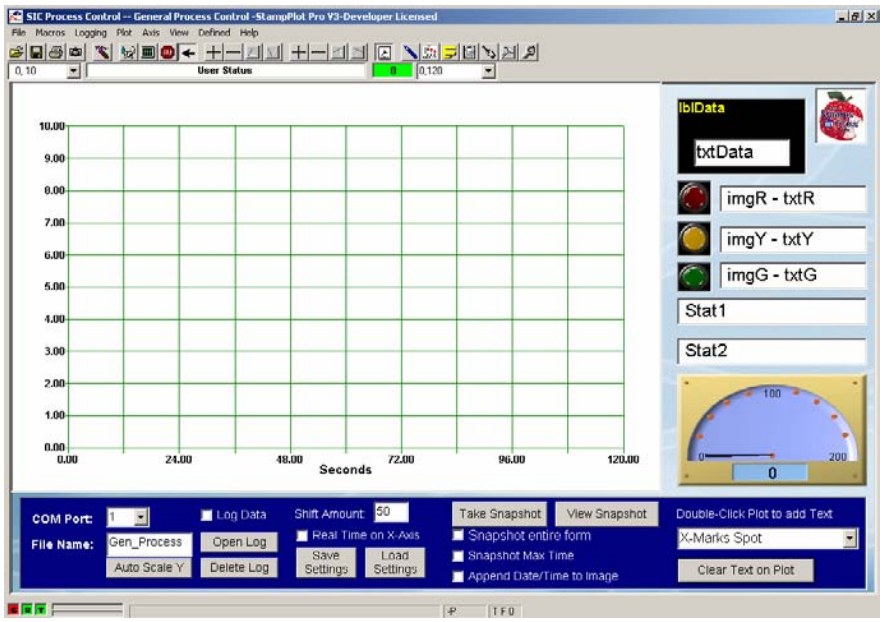
```
PAUSE 500                        ' Connection stabilizing time

DEBUG CR,"!RSET",CR,             ' Reset StampPlot
      "!CLRC",CR,                ' Clear any text on plot
      "!SPAN 0,500",CR           ' Set Y-Axis span

DEBUG "@TEXT 1A,D0,1A,(Blue), Start Sw",CR,   ' Label digital data traces
      "@TEXT 1A,D1,1A,(Blue), Proximity Detector",CR,
      "@TEXT 1A,D2,1A,(Blue), Stop Sw",CR,
      "@TEXT 1A,D3,1A,(Blue), Fill",CR,
      "@TEXT 1A,D4,1A,(Blue), Mix",CR,
      "@TEXT 1A,D5,1A,(Blue), Drain",CR

DEBUG "!O lblData = Mix Time:\n(Sec)",CR,     ' Label input text box area
      "!O txtData = 15",CR,                   ' Set initial value for mix time
      "!O txtR = Filling",CR,                 ' Label other controls
      "!O txtY = Mixing",CR,
      "!O txtG = Draining",CR,
      "!O Stat1 = Idle",CR,
      "!O Stat2 = Total Gallons:",CR,
      "!O txtFileName = Mix_Seq",CR           ' Label file name for saves

DEBUG "!O Meter = 0,0,500",CR     ' Set SP meter - current,min val, max val

DEBUG "!RSET",CR                  ' Reset after configuring

' -----[ Main Routine ]-----------------------------------------------------
DO
  DO UNTIL (Start_Sw = 0)         ' Wait until start button pressed
     GOSUB Display_Data
  LOOP

  DEBUG "!READ (txtData)",CR      ' Read time to mix from plot
  DEBUGIN DEC Mix_Seconds         ' Accept data, store in Mix_Seconds.

  HIGH Fill                       ' Begin fill
  DEBUG "!O Stat1 = Filling",CR

  DO UNTIL (Opto_Sw = 0)          ' Wait until stop button is pressed
    GOSUB Display_Data
  LOOP

  LOW Fill                        ' Stop fill
  HIGH Mix                        ' Start mixing

  FOR X = 1 TO Mix_Seconds        ' Mix while updating plot
    GOSUB Display_Data
      DEBUG "!O Stat1 = Mixing ",
            DEC X,
            " Seconds", CR
      PAUSE 900                   ' 100 pause in Display Data added to this
```

```
  NEXT

  LOW Mix                          ' Stop mixing
  HIGH Drain                       ' Start draining
  DEBUG "!O Stat1 = Draining",CR

  DO UNTIL (Stop_Sw = 0)           ' Wait for stop button
    GOSUB Display_Data
  LOOP

  LOW Drain                        ' Stop draining
  DEBUG "!O Stat1 = Completed",CR
  Total_Mixed = Total_Mixed + Vat_Volume     'Accumulate total
  DEBUG "!O Stat2 = Total Gallons:", DEC Total_Mixed,CR
LOOP

' -----[ Subroutines ] ------------------------------------------------
Display_Data:
  DEBUG "!O imgR =", BIN Fill, CR,           ' Update SP virtual indicators
        "!O imgY =", BIN Mix, CR,
        "!O imgG =", BIN Drain, CR

  DEBUG "!O METER =", DEC Total_Mixed,CR ' Update SP Meter

  DEBUG IBIN Start_Sw, BIN  Opto_Sw, BIN  Stop_Sw,  ' Plot binary data
        BIN  Fill, BIN  Mix, BIN  Drain,CR

  DEBUG DEC Total_Mixed,CR                   ' Plot analog data
  PAUSE 100
RETURN
```

- √ Close the Debug Terminal.
- √ Run StampPlot macro sic_gen_process.spm.

Figure 4-11 is an image of StampPlot configured with this macro. For the next several activities, this interface will be used. Unlike previous macros that accepted data and processed it in specialized ways, this macro is more general in use and will be directly configured and controlled from the BASIC Stamp.

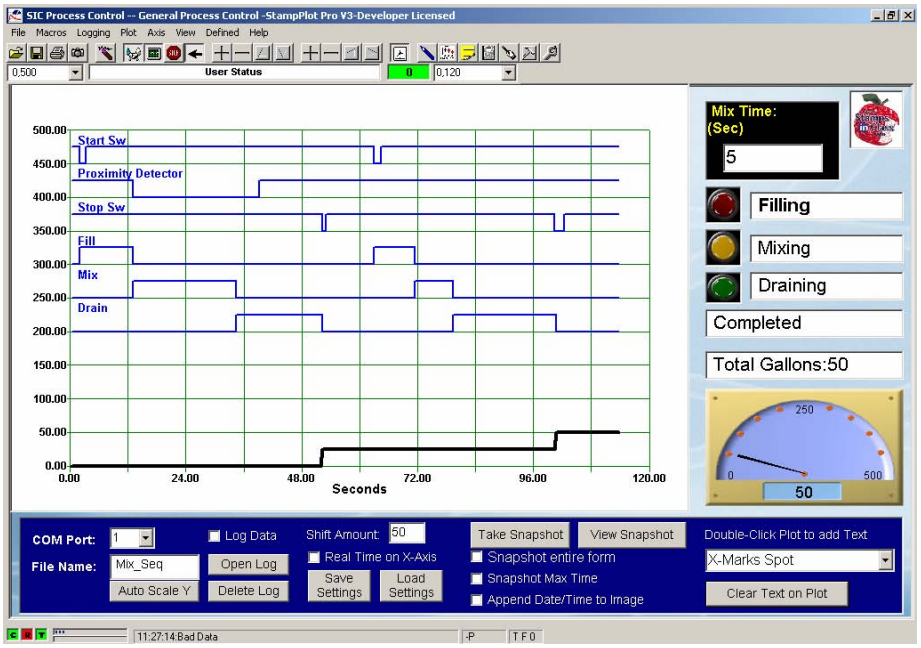
Figure 4-11 General Purpose Interface



- √ Connect on StampPlot. Notice that the controls on the interface are updated accordingly with a default mix time of 15 seconds.
- √ Press the Start Button (PB1) to begin the fill operation.
- √ Observe that the filling operation has begun by indication.
- √ After a short time (about 10 seconds), place an object in front of the opto-reflector to simulate a full vat.
- √ Observe that the mix operation has begun. After 15 seconds, the mixing will cease and draining will commence.
- √ Remove the object to simulate a less-than-full vat.
- √ Allow a short time for vat to empty before pressing the Stop button (PB2).
- √ The drain operation will be complete, and the accumulated total of gallons mixed will be updated as indicated by the analog value increasing.
- √ The system is ready for another batch! Change the mix time to 5 on StampPlot, tab off the box, and repeat the batch sequence.
- √ Note the mix time for the batch.

The three input switches are all Active-LOW, and the three outputs are all Active-HIGH. In Figure 4-12 the sequence of operations can clearly be seen in the traces for both inputs and outputs.

Figure 4-12 Fill, Mix and Drain Batch Sequence Plot



PROGRAM DISCUSSION

Unlike the previous StampPlot interfaces used, this one was designed to be more programmer-friendly for use. Previous macros had code within them to update gauges, meters, text boxes, and for other needs. This macro is not as intelligent in that the BASIC Stamp must send the data to update the various objects on the plot screen, but this also makes the interface more flexible.

Each control has a name. The three virtual indicating lights are named **`imgR`**, **`imgY`** and **`imgG`**. The corresponding text boxes for each are **`txtR`**, **`txtY`** and **`txtG`**. From the BASIC Stamp a **`DEBUG`** instruction can update the controls depending on need. For

example, during initialization, the three indicator text boxes are updated to name their functions, such as:

```
DEBUG "!O txtR = Filling", CR
```

This code will place Filling in the red indicators text box. **!O** means to use a plot object control which is then named and assigned a new value (**!O** is short for **!POBJ-** Plot Object). Examples of this are found throughout the code, updating the two status text boxes such as:

```
DEBUG "!O Stat1 = Draining", CR
```

The label for the mix time area, **lblData**, is again general purpose and may be set to any desirable text as in:

```
DEBUG "!O lblData = Mix Time:\n(Sec)", CR
```

The **\n** is a line feed (new line) symbol to place text on two separate lines.

Data from **txtData** will be read by the BASIC Stamp and used in the control process to define how long the mixing should occur.

```
DEBUG "!READ (txtData)",CR     ' Read time to mix from plot
DEBUGIN DEC Mix_Seconds        ' Accept data, store in Mix_Seconds
```

The three indicators have two images assigned when created by the macro – a 1's value image (lit lamp) and a 0's (dark lamp) value image. Both images are simply JPGs in the StampPlot media\comp directory. In code such as:

```
DEBUG "!O imgR =", BIN Fill, CR
```

The image control is assigned a 1 or 0 value to show the respective image.

Text labels for digital traces are placed on the plot through the use of the StampPlot **TEXT** instruction. The general format for placing text is:

@TEXT *x-coordinate, y-coordinate, size, color, text*

The **@** indicates the text is to be constant or will survive a reset of the plot. **!CLRC** is used to clear constant text and drawings on the plot, like those used in the initialization. Note: there can be no spaces between the X and Y coordinate parameters!

To label our 6 digital traces we used this line of code:

```
DEBUG "@TEXT 1A,D0,1A, (Blue), Start Sw", CR
```

In this example, **1A** is an absolute coordinate meaning as the plot shifts the text will remain static. The same is true for size, **1A** again, so the text won't change size when the plot changes scale. **D0** is a y-coordinate relating to the top (or 0) digital trace. **D1** is the second, and so forth. Note that bit positions and D-positions are backwards. The top trace (**D0**) is actually the most-significant bit (bit 5 for this plot).

Challenge 4-2: Adding an Emergency Stop

If a problem occurs, such as the mechanical joint leaking or a relay smoking, how long will it be before the system can be stopped?

- √ Save BatchMix.bs2 under a new name.
- √ Add a subroutine and **GOSUB** command for shutdown of the entire system at any time; continue to plot data and loop until controller reset. Show the shutdown code and at least one routine call.

ACTIVITY #3: PRODUCTION LOGS

What company could exist without logs and records? Production logs can be used to determine total output produced and to spot trends. Is the filling or draining of the vat slowing due to possible obstructions? Did an error occur during a run with a faulty input device? Did the operator wait until the end-of-shift to hit the Stop button after the mix was complete?

Parts Required

Same as Activity #2

StampPlot has the capabilities to automatically log data to a file for review or to import data into another program, such as Microsoft Excel®.

- √ In the BASIC Stamp Editor, re-run BatchMix.bs2.

- √ Close the Debug Terminal.
- √ Run StampPlot macro sic_gen_process.spm.
- √ Click the "Delete Log" button on the interface and affirm the notice.
- √ Check the "Log Data" check box to begin storing incoming data to StampPlot.
- √ Run a batch.
- √ Click the "Open Log" button to view the data saved as shown in Figure 4-13.

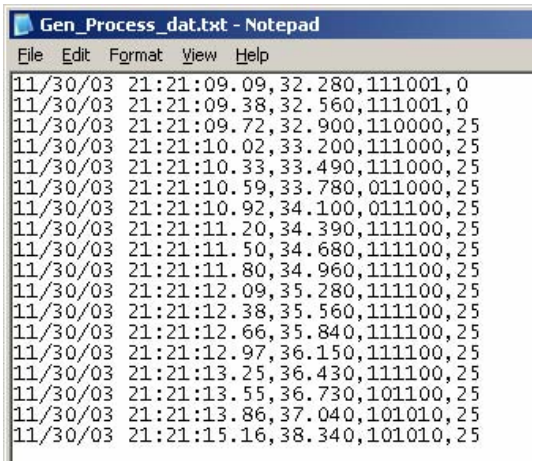


Figure 4-13
Mixing Production
Data Log

The format of the log is:

Date and Time, seconds into the plot, binary data from I/O plotted, analog value(s).

Each bit in the logged binary data represents the following:

Start Sw Proximity Det Stop Sw Filling Mixing Draining

In the line:

11/30/03 21:21:10.92,34.100,011100,25

What is the current status based on the binary data? Keep in mind which I/O are active-high and which are active-low.

Challenge 4-3: Manual Data Logging

StampPlot also allows manual logging of data using the **!LOGD** instruction such as:

```
DEBUG  "!LOGD FILLING",CR
```

Data from the BASIC Stamp can be incorporated:

```
DEBUG "!LOGD Run Complete - Gallons Mixed: ", DEC Total_Mixed, CR
```

Finally, data from StampPlot can be incorporated by enclosing object names and macro values in parentheses:

```
DEBUG "!LOGD (PTIME) STATUS: (Stat1)", CR
```

...where **PTIME** is the time into the plot (in seconds), and **Stat1** is the name of the status text box used for current status.

- √ Save BatchMix.bs2 under a new name.
- √ Modify the program to log only major events (begin filling, etc) and the accumulated gallons mixed at the end of the run.
- √ Do not enable "Log Data" on the interface or this will also log data to the same file.
- √ Click the “Delete Log” button to delete the data file, and confirm your selection.
- √ Perform a fresh run.
- √ Click the “Open Log” button to open the data file.

The Date and Time will be suffixed automatically (time stamped) as long as the button on the toolbar is down or **DEBUG "!TSMP OFF", CR** is not issued. All this text takes up some pretty hefty memory in the BASIC Stamp. If you run out of memory while adding this, delete unnecessary text sent to StampPlot such as labeling the indicators or labeling the digital plot lines. See the StampPlot help files under *Summaries – Math Summaries* for a list of intrinsic StampPlot values that could be used.

ACTIVITY #4: BOX CONVEYOR BELT – COUNTING AND EDGE DETECTION

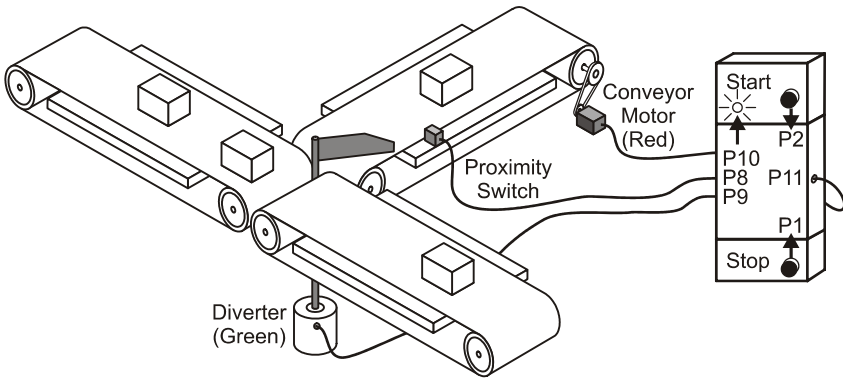
Parts Required

Same circuit as Activity #2

This conveyor belt simulation activity uses the same BASIC Stamp circuit and StampPlot macro as the previous activities, but addresses different issues by using a conveyor system to count and divert boxes to one of two loading bays. The following is the feature set of this control system, depicted in Figure 4-14:

- Detection and counting of boxes to be loaded onto trucks. The yellow LED indicates detection.
- Operation of a gate to divert the boxes to one of two truck-loading bays signified by a green LED. One full truckload is 6 boxes.
- Operation of the conveyor belt with the use of two pushbutton switches – Start and Stop. The energized conveyor is signified with the red LED.

Figure 4-14 Conveyor Belt System

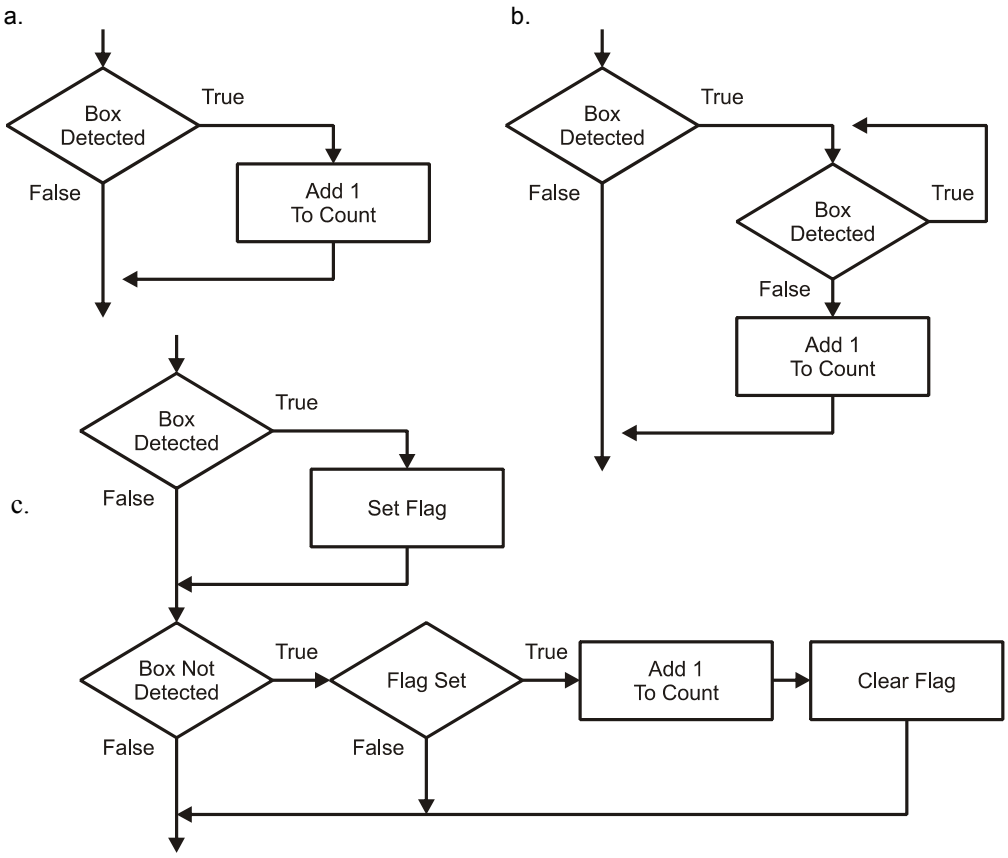


EDGE DETECTION

Before beginning any programming, let us first discuss issues involved in counting with digital inputs. One of the functions of the system is to count the boxes that pass. The digital input is LOW when a box is detected with the opto-reflective switch. Should the

program count every sample if a low level is sensed, as depicted by the partial flowchart in Figure 4-15 (a)?

Figure 4-15 Detection and Counting Algorithms



"Sample Time" is how frequently a program reads a value. If the sample time were 100 ms and the box was present for 2 seconds, how many counts would have occurred? The algorithm would have counted 20 boxes! So the count cannot be based on whether the object is detected or not. The count needs to occur only once: either at the first detection of a box or when the box passes. A count needs to occur on the transition from HIGH to LOW or LOW to HIGH. In digital terms, this is known as "Edge Triggering".

Consider the partial flowchart in Figure 4-15 (b). As long as a box is not detected, the decision will be false and no count will occur. When a sample detects a box, the flow will loop and wait until the box is no longer detected (waiting for the transition) before counting. Do you see any problems in the algorithm? Once the flow enters the waiting loop, no other actions can take place, such as sensing if the STOP button is pressed in the event of an emergency. If someone's sleeve gets caught, they may have lost a limb during the time it takes the box to pass! The batch mix program in Activity 4-2 used subroutine calls in order to continually plot data. The call could also be used to check the status of the stop button. However, if our program has many routines that need to be continuously performed, that may not be the best solution.

Certain processors, such as the BS2p, have polling and/or interrupt capabilities to branch automatically to a location based on conditions, but the BS2 does not.

Through the use of a flag a much more elegant and efficient means is demonstrated in Figure 4-15 (c). A flag is simply used to indicate a condition. One example is raising the flag on a mailbox to indicate a need to pick-up outgoing mail. In programming, a flag is typically a bit variable, set HIGH to indicate the occurrence of a condition.

In Figure 4-15 (c):

- Once a box is detected (TRUE), the flag is set.
- When the box has passed and is not present (FALSE), the flag will be checked.
- If the flag is set, it indicates that a box HAD been present, so the program will add one as passing. - Don't forget to reset the flag for the next box!
- When the box is not present and the flag is not set, program execution will continue without counting a box. This allows the remaining code in the program to continue execution by flagging an event instead of waiting for an event.

In the following code, a nested **`IF...THEN`** could be used to check the condition of the box passing and then the flag to see if a count needs to occur:

```
IF (Opto_Sw = 1) THEN                ' True if no box
  IF (Edge_Flag = 1) THEN            ' True if Flag set
     Box_Count = Box_Count + 1
     Edge_Flag = 0
  ENDIF
ENDIF
```

The count would only occur when both conditions are true. **Opto_Sw** needs to indicate no box present AND the flag needs to be set for a count. That "AND" is called Boolean Logic and the BASIC Stamp supports it directly by use of an **AND** operator:

```
IF (Opto_Sw = 1) AND (Edge_Flag = 1) THEN
   Box_Count = Box_Count + 1
   Edge_Flag = 0
ENDIF
```


When using an **AND** operator, both (or all) conditions must be true for the entire statement to be true. Another operator is **OR**, where either (or any) condition must be true for the entire statement to be considered true. As many operators can be used as needed to qualify a statement. Finally **XOR**, or Exclusive-OR, means one or the other condition can be true, but not both. Table 4-2 is a truth table for the logical operators:

Table 4-2: Boolean Operator Truth Table

Condition A	Condition B	A AND B	A OR B	A XOR B
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE

The final program will have a bit more code in the condition to control the diverter, but the principle is the same. Other than that section, the code is pretty straightforward and the full flowchart is not shown.

√ Enter, save and run BoxConveyor.bs2.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - BoxConveyor.bs2
' Control System for conveyor belt control and box counting
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
Stop_SW       PIN     1         ' PB1
Start_SW      PIN     2         ' PB2
Opto_SW       PIN     8         ' Opto-Reflective switch

Diverter      PIN     9         ' Green LED
Box_Det       PIN     10        ' Yellow LED
```

```
Conveyor      PIN     11       ' Red LED

Box_Count     VAR     Word     ' Total boxes
Box_Truck     VAR     Byte     ' Boxes on truck
Edge_Flag     VAR     Bit      ' Flag that box was detected
Max_Truck     VAR     Byte     ' Maximum boxes per truck


' -----[ Initialization ]----------------------------------------------------
LOW Conveyor                   ' Set initial states
LOW Diverter
LOW Box_Det
PAUSE 500
DEBUG CR,"!RSET",CR,           ' Reset StampPlot
      "!CLRC",CR,              ' Clear constant drawings
      "!SPAN 0,20",CR          ' Set Y axis

DEBUG "@TEXT 1A,D0,1A,(Blue),Conveyor",CR, ' Label digital data traces
      "@TEXT 1A,D1,1A,(Blue),Detect",CR,
      "@TEXT 1A,D2,1A,(Blue),Diverter",CR

DEBUG "!O lblData = Boxes per",CR,         ' Label input text box area
      "!O txtData = 6",CR,                 ' Set initial boxes per truck
      "!O txtR = Conveyor",CR,             ' Label indicator text boxes
      "!O txtY = Detector",CR,
      "!O txtG = Diverter",CR,
      "!O txtFileName = Box_conv",CR       ' Label file name for image saves

DEBUG "!O Meter=,0,200",CR                 ' Configure meter
DEBUG "!RSET",CR

' -----[ Main Routine ]------------------------------------------------------
DO
  GOSUB Run_Control
  GOSUB Count_Box
  GOSUB Display_Data
LOOP

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

Run_Control:
  IF (Start_SW = 0) THEN             ' Start pressed?
        HIGH Conveyor                ' Energize conveyor
        DEBUG "!READ txtData",CR     ' Request boxes per truck from StampPlot
        DEBUGIN DEC Max_Truck        ' Accept returning data
  ENDIF
  IF (Stop_SW = 0) THEN LOW Conveyor  ' Stop pressed?
  RETURN
```

```
Count_Box:
  IF (Opto_SW = 0) THEN               ' Box detected?
    Edge_Flag = 1                     ' True, set flag
    HIGH Box_Det                      ' Energize LED
  ENDIF

  IF (Opto_Sw = 1) AND (Edge_Flag = 1) THEN  ' No box but was detected?
    Box_Count = Box_Count + 1                ' True, add one to count
    Box_Truck = Box_Count // Max_Truck       ' Take modulus to limit max count
    IF (Box_Truck = 0) THEN                  ' Box count at 0?
      TOGGLE Diverter                        ' True, change diverter
    ENDIF
    LOW Box_Det                              ' De-energize detect LED
    Edge_Flag = 0                            ' Reset flag
  ENDIF
  RETURN

Display_Data:
  DEBUG "!O imgR =", BIN Conveyor,CR,  ' Update indicators images
        "!O imgY =",BIN Box_Det,CR,
        "!O imgG =",BIN Diverter,CR

  ' Plot digital status of control
  DEBUG IBIN Conveyor,                 ' Send with leading % in binary
        BIN Box_Det,                   ' other two without leading %
        BIN Diverter,CR


  DEBUG "!O Stat1= Boxes to truck:",   ' Update status boxes
        DEC Box_Count // Max_Truck,CR,
        "!O Stat2= Total boxes:",
        DEC Box_Count,CR

  DEBUG DEC Box_Truck,CR               ' Plot analog value of boxes loaded

  DEBUG "!O Meter=", DEC Box_Count,CR  ' Update meter
  RETURN
```

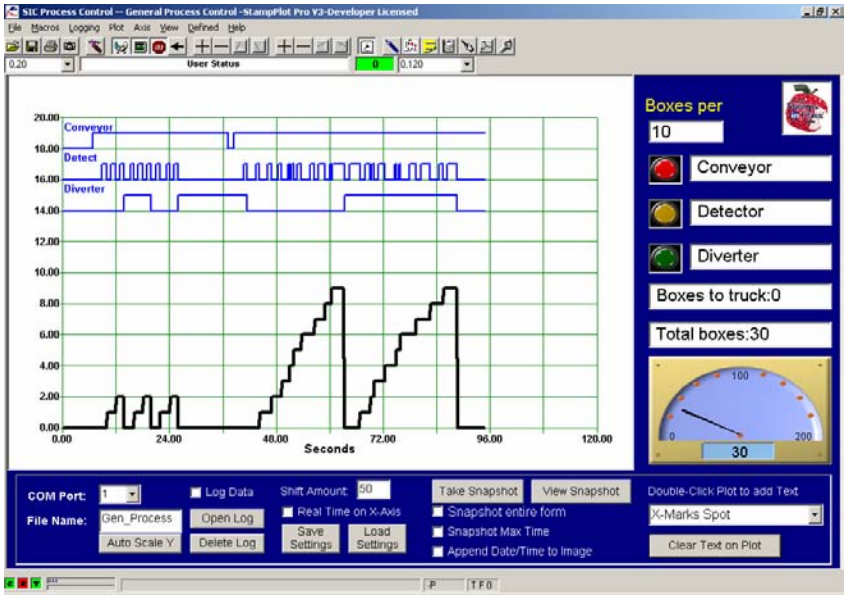
- √ Close the Debug Terminal.
- √ Run StampPlot macro sic_gen_process.spm.
- √ Connect and plot.
- √ Start the conveyor and run some boxes past the detector.

As boxes are counted and the diverter is switched, note the indicators changing. At what point does a new count take place in relation to the detector trace? What count is actually indicated for a full truckload? Why?

- √ Stop the conveyor and change the value of the StampPlot input text box "Boxes Per" to 8.
- √ Start the conveyor and run boxes past. What changes? Why?

Figure 4-16 shows a plot from a sample run that includes one problem you may have noticed in your testing – multiple counts from a single box. This is addressed in the next activity.

Figure 4-16 Sample Box Run For 3 and 10 Boxes per Truck



One other line of code worth mentioning is:

```
Box_Truck = Box_Count // Max_Truck
```

The `//` is the modulus operator, which provides the remainder from division. What would be the remainder of `5//2`? The remainder would be 1. Take the example of filling 6-packs of soda. If you have 40 bottles of soda, how many would be left over after filling as many 6 packs as possible? 40//6 = 4.

Challenge 4-4: Better Start/Stop Control

What happens when both the START and STOP buttons are held down? The LED on the board blinks rapidly. If this were controlling a motor it would be cycling on and off. In an emergency, this is neither good for the motor nor for the person whose hand got caught in the conveyor!

- √ Save BoxConveyor.bs2 under a new name.
- √ Use Boolean operators to start the conveyor only if the Start button is pressed and the stop button is not pressed. Show your code.

ACTIVITY #5: INPUT BOUNCE AND SPURIOUS SIGNALS

One problem is seen when the progress of the box isn't nice and smooth. Imagine the box rocking and rattling on the conveyor belt. Run the simulation and wiggle the 'box' back and forth as it enters the range of detection. What occurs? The sample speed on the detector is fast enough to register several units as the box momentarily enters and exits the range of detection.

Almost any switch that measures a physical quantity can suffer from "bounce". Pushbuttons and switches have mechanical contacts that bounce against each other several times before stabilizing. With optical switches, there is almost never a clean transition as a slow-moving object enters a beam. Sometimes multiple triggering is not important, such as sensing start button being pressed once or a dozen times. At other times, such as counting, it is very important. Another consideration is spurious signals such as a lump of dust falling in front of the detector. Or the earlier vat filling - what would happen as the water splashes near the detector as the vat fills? Would this have signaled a spurious reading of the vat being full?

Electronics are often implemented to smooth out the signal, such as a low-pass filter to reduce noise, bounce and spurious signals. In programming, these issues can be handled in a number of ways, such as re-sampling and averaging. A simple method for handling switch bounce and spurious signals is to pause the length of time it takes an input to stabilize and then re-sample to ensure a good reading. The length of the pause should be long enough for the input to stabilize before sampling again. Pressing a button may require 100 ms to become steady. A slow-moving box on a noise conveyor may require several seconds, but not so long that the signal is missed.

Parts Required

Same circuit as Activity #2

Excessive Signal Durations

When dealing with the physical world and human-input, most things never go as planned, but a well-programmed process control system should be able to deal with these mishaps. As the boxes make their way down the conveyor, it is likely that problems will eventually occur. A box will possibly jam as the diverter shifts. What will our indication be of a problem? Someone on the docks telling us they aren't getting boxes? The controller can be made sophisticated enough to detect when a box is present for too long, shutdown the conveyor, and sound warnings.

The current program is using flags to indicate when a box is detected. It is set repeatedly as the program keeps sampling. An accumulator with the following capabilities can be used to keep track of how many times the box has been detected.

- As the box is detected, a variable is incremented by one with each scan of the input.
- If the accumulator reaches a critical value (10?), the system is shut down and warnings issued.
- If no box is detected the variable is reset to zero.

What's an appropriate critical value? It depends on the sampling rate. For example, if a box requires 4 seconds to pass the beam and the program is sampling every ½ second, a normal accumulated value may be 8. The critical value should be at least twice this.

To add code to detect and respond to a box detection signal of excessive duration, we will need to implement these steps:

- Create a variable to hold an accumulated value.
- Add code to add one to the variable whenever a box is detected.
- Add code to zero the variable when the box is not detected.
- Add code to shut down the conveyor and sound an alarm if the accumulator reaches an excessive value. A good StampPlot alarm can be implemented using:

```
DEBUG "~PWAV nralarm",CR
```

- Use the main status box above the plot for feedback showing the accumulated value when testing:

```
DEBUG "!STAT Accum value=", DEC your_accum_variable, CR
```


So let's try it!

√ Save BoxConveyor.bs2 under a new name: BoxConveyorSignal.bs2.
√ To the Declarations section add:

```
Accumulator   VAR  Byte
```

Assuming a loop time of about 100 ms (processing time), if a normal box passing requires 3 seconds, double would be 6 seconds. How many loops will have been performed? 60.

√ Modify the **`Count_Box`** subroutine by adding the bold lines:

```
Count_Box:
 IF (Opto_SW = 0) THEN                 ' Box detected?
   Edge_Flag = 1                       ' True, set flag
   HIGH Box_Det                        ' Energize LED
   Accumulator = Accumulator +1        ' Add one to accum
   DEBUG "!STAT Accum value = ", DEC Accumulator,CR  'Show count
   IF Accumulator = 30 THEN            ' To many detects
       LOW Conveyor                    ' Turn off conveyor
       DO
          DEBUG "~PWAV nralarm",CR     ' sound alarm until reset
          PAUSE 5000
       LOOP
   ENDIF
 ENDIF

 IF (Opto_Sw = 1) AND (Edge_Flag = 1) THEN  ' No box but was detected?
   Box_Count = Box_Count + 1           ' True, add one to count
   Box_Truck = Box_Count // Max_Truck  ' Take modulus to limit max count
   IF (Box_Truck = 0) THEN             ' Box count at 0?
     TOGGLE Diverter                   ' True, change diverter
   ENDIF
   LOW Box_Det                         ' De-energize detect LED
   Edge_Flag = 0                       ' Reset flag
   Accumulator = 0                     ' Reset accumulator if box detected
 ENDIF
RETURN
```

Challenge 4-5: Debounce with Programming

- √ Save BoxConveyor.bs2 under a new name.
- √ To the `Count_Box` subroutine add code that would prevent input bounce from affecting the count for normal conditions. Pseudocode is provided for one method. Show your code and indicate your results.

  Detect box.
  Pause appropriate time.
  Check box again.
  If still detected, set the flag.

ACTIVITY #6: TACHOMETER – HIGH-SPEED COUNTING

Many processes, such as measuring the revolutions of a shaft, must employ high speed counting. A tachometer measures the number of shaft rotations per unit time. Other examples include measuring the revolutions per minute (RPM) of engines, motors, generators, cutting drills, and CD ROMS.

A tachometer uses input pulses to calculate the RPM. Pulses may be obtained through various methods, including detecting light reflection, using magnetic sensors or having a coupled DC generator provide an analog output.

In this activity, the RPM of a fan will be measured by counting the number of encoder wheel reflections per unit time. As the fan spins, the light from the LED emitter will be reflected to the detector (white segment) or absorbed (black segment). This will provide a pulse stream into the BASIC Stamp. By counting the transitions, the fan's RPM can be calculated.

The PBASIC `COUNT` instruction is used to count the number of pulses for a given time and then store the value in a variable:

COUNT ***Pin, Duration, Variable***

...where ***Duration*** is the time during wthch to count, in 1 ms units for the BASIC Stamp 2. The count results will be stored in the ***Variable*** specified.

Testing the Encoder Wheels

The encoder wheels work only when the black segments absorb the infrared light emitted by the QRB1114. A black surface produced by laser toner and inkjet copiers usually works well, but a home photo printer that accepts glossy paper may use metallic ink that reflects infrared. Commercial printing processes vary. So, we must test our encoder wheels before proceeding.

Parts Required

Same circuit as Activity #2
Scissors and double-sided tape or rubber cement (not included)

- √ Photocopy or tear out the encoder wheels page from Appendix A. You may also print out this page from the Process Control pdf available from www.parallax.com.
- √ Repeat Activity 2, but using both the white and the black areas of the 1 Cycle/revolution encoder. Make a note of the distance where both white reflects and black absorbs the infrared light.
- √ If the printed encoder you are using does not work, try photocopying the page with a different method, or coloring over the black areas with a felt-tip pen.
- √ When you have confirmed that you have a set of encoders that are visible to the QRB1114, carefully cut them out.
- √ Using double-sided tape, rubber cement, or some other non-permanent method, attach the 1 cycle/revolution retro-reflective encoder wheel from Figure 4-17 to the fan. Do not place tape on the surface as it may reflect, affecting your readings.

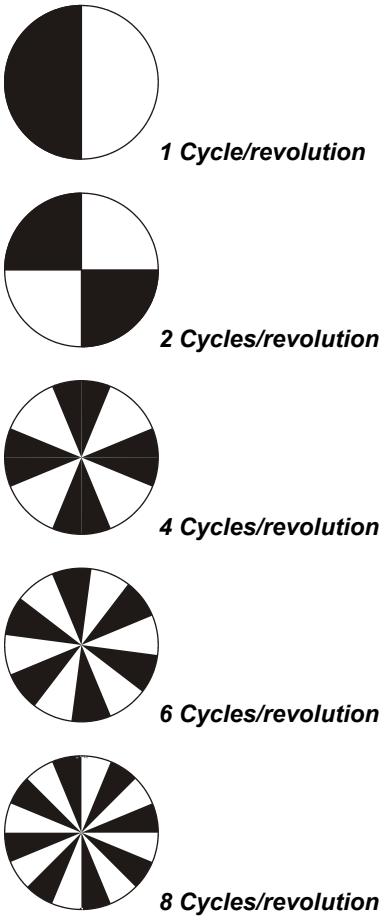


Figure 4-17
Retro-Reflective
Encoder Wheels

Full-size duplicates for photocopying or cutting out may be found in Appendix A.

Now that you have confirmed usable encoders, let's modify our circuit to use the fan. It is just a matter of removing the two pushbutton circuits and the LED circuits connected to P10 and P11, and then adding the fan. The parts list, schematic and wiring diagram are provided below for clarity.

Parts Required

(1) ADC0831
(1) Resistor – 100 Ω
(2) Resistors – 220 Ω
(1) Resistor – 10 kΩ
(1) LED – Green
(1) Opto-Reflective Switch – QRB1114
(1) Brushless DC Fan

√ Build (or modify) your circuit as shown in and Figure 4-18 and Figure 4-19.

Figure 4-18: Tachometer Schematic (top) and Wiring Diagram (bottom)

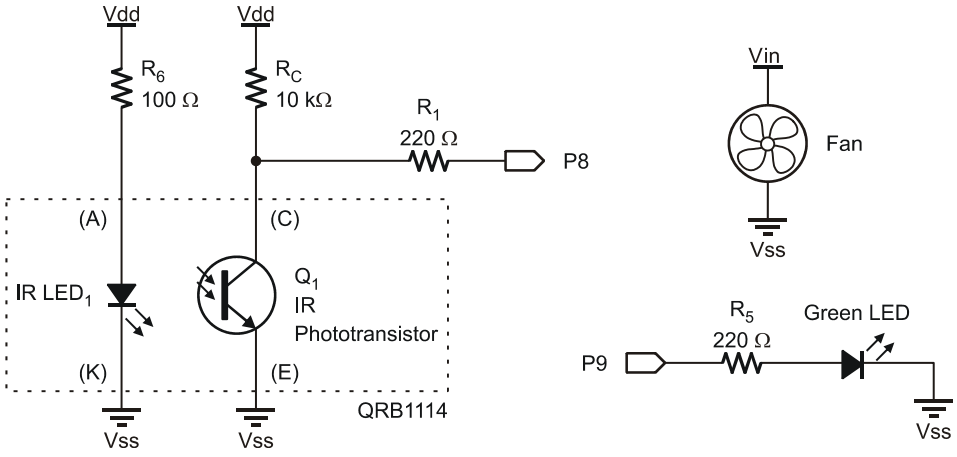
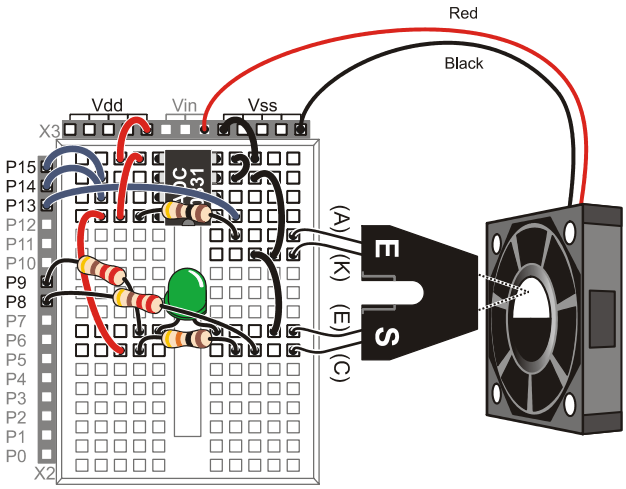


Figure 4-19: Tachometer Wiring Diagram



√ Enter, save and run Tachometer.bs2.
√ Close the Debug Terminal.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - Tachometer.bs2
' Measures RPM of Fan
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
Opto_SW       PIN     8          ' Opto-Reflector
Sampled       PIN     9          ' Indicator to indicate sampling

Opto_Count    VAR     Word       ' Count from opto-reflective switch
RPM           VAR     Word       ' Calculated RPM
SP_Data       VAR     Word       ' Data returned from StampPlot

CyclesPerRev  CON     1

' -----[ Initialization ]--------------------------------------------------
PAUSE 500                         ' Connection stabilizing time

DEBUG CR,"!RSET",CR,              ' Reset StampPlot
      "!CLRC",CR,                 ' Clear any text on plot
      "!SPAN 0,10000",CR          ' Set Y-Axis span
```

```
        ' Label text boxes
DEBUG "!O lblData = Sample Time\n  (mSec)", CR,
      "!O txtData = 1000", CR,
      "!O txtR = ", CR,
      "!O txtY = ", CR,
      "!O txtG = Sampled", CR,
      "!O Stat1 = Counts:", CR,
      "!O Stat2 = RPM:", CR,
      "!O txtFileName = Tach1", CR

DEBUG "!O Meter = 0,0,10000,0,10000", CR     ' Set SP meter
DEBUG "!RSET", CR                            ' Reset after configuring

LOW Sampled

' -----[ Main Routine ]------------------------------------------------
DO
  GOSUB ReadSP
  GOSUB ReadTach
  GOSUB DisplayData
LOOP

ReadSP:
  DEBUG "!READ (txtData)",CR      ' Request data from StampPlot
  DEBUGIN DEC SP_Data             ' Accept returning data
RETURN

ReadTach:
  COUNT Opto_SW,SP_Data,Opto_Count       ' Measure counts per unit time
  TOGGLE Sampled                         ' Toggle LED to show sample done
  RPM = Opto_Count * (60000 / SP_Data) / CyclesPerRev  ' Calculate RPM

RETURN

DisplayData:
  DEBUG DEC RPM,CR                                ' Analog data of RPM
  DEBUG IBIN Sampled, CR                          ' Digital trace of samples
  DEBUG "!O Stat1 = Counts: ", DEC Opto_Count, CR  ' Update controls
  DEBUG "!O Stat2 = RPM: ", DEC RPM, CR
  DEBUG "!O METER =", DEC RPM, CR ' Update SP Meter
  DEBUG "!O ImgG = ", BIN Sampled, CR
RETURN
```

- √ Run StampPlot macro sic_gen_process.spm.
- √ Connect and plot.
- √ Position the fan in front of the opto-reflective switch at the optimum distance, as found in encoder test, or at a distance that provides good indication of RPM. The detector should be pointing at one side or the other of the encoder- not at the

center. A typical reading is 8000 RPM, though this will vary with supply voltage.

√ Lightly touch the front of the fan to vary speed, and monitor on StampPlot. Monitor the RPM.

√ **Skip these next two steps if you are using a BASIC Stamp HomeWork Board!!** Move the fan's red (+) lead to Vdd. The fan may need a 'jump start' by manually spinning it. What is the RPM of the fan now? The fan used is a brushless DC motor rated for 12 V with a dropout, or stall, voltage of 3.5 V. The voltage supplied to it will affect its rotational speed.

√ Return the red lead of the fan to Vin or disconnect for now.

Figure 4-20 is screen shot of the fan's RPM when powered from Vin. The fan was manually slowed by lightly touching it. Note the data obtained. The count obtained with the retro-reflective switch is 134. With a sample time of 1000 ms (1 second), an RPM of 8,040 is calculated (134 x 60 seconds). With the 1 cycle/revolution encoder wheel, counting the pulses for 1 second provides a value for revolutions per second. To obtain revolutions per minute, the count is multiplied by 60.

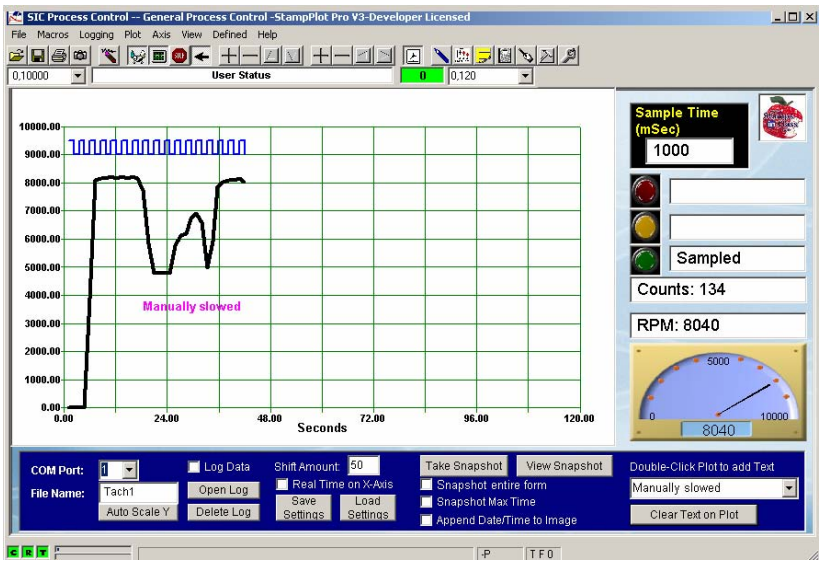
RPM = Count * 60 / #cycles per revolution

In the BASIC Stamp code:

```
RPM = Opto_Count * (60000 / SP_Data) / CyclesPerRev
RPM = 134 * (60000 / 1000) / 1 = 8040 RPM
```

The count is multiplied by 60000 ms (60 seconds) and divided by **`SP_Data`** that was read from StampPlot which is 1000 ms (1 second) by default. Finally, the result is divided by the number of cycles per revolution, or 1 in this case. The digital trace, the green LED, and the StampPlot indicator will toggle, or change states, during each loop iteration (sample).

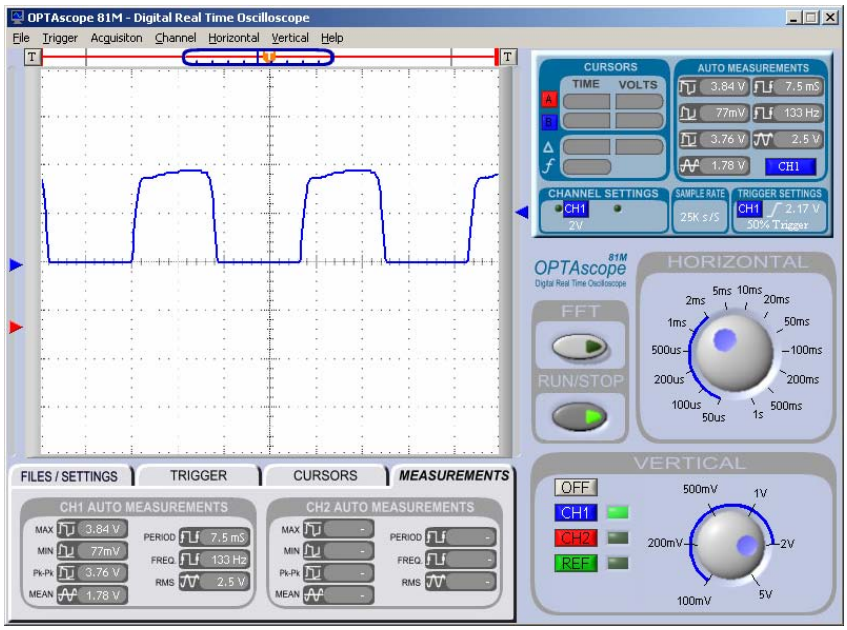
Figure 4-20 Fan RPM with When Powered from Vin



With this configuration and sample time what is the resolution of the tachometer? For each count change, there is a change of 60 RPM. Depending on the application, this may or may not be an acceptable resolution.

Figure 4-21 is an oscilloscope capture of the voltage at the phototransistor's collector. The waveform shows the voltage at P8 at the input to the BASIC Stamp (as the fan rotates).

Figure 4-21 Waveform of Phototransistor Output for 1 Cycle/Revolution Encoder



Note the period of the waveform. The time represented between each horizontal major division, or dashed vertical line, is 2 ms. A complete cycle takes approximately 3.6 divisions.

3.6 div x 2 ms/Div = 7.2 ms for the period of the wave (T).

Frequency is the inverse of period, f = 1/T.

f = 1/7.2 ms = 138 Hz, or the number of cycles per second (Hz).

More accurate readings are obtained by the software and displayed below the waveform – 133 Hz and 7.5 ms. How does the frequency compare to what the BASIC Stamp was reading for a count of 1 second? Also note that the waveform does not go from HIGH to LOW sharply. The rise and fall of the wave takes a finite amount of time (about 0.4 ms). This is due to the time required for the encoder vanes to transition and the response of the phototransistor.

This waveform was captured using the circuit in the next activity. Effects of loading lowered the metered voltage significantly due to the high impedance output of the circuit (RC = 1MΩ) and the impedance input of the oscilloscope used for waveform capture.

Increasing the count sample time leads to greater accuracy and resolution. The best resolution occurs when the count is obtained over a full 60 seconds giving a very accurate value for RPM. What problem does this introduce? If the BASIC Stamp is controlling or monitoring the system, will a 60 second pause in control be a detriment? A process control system may require fast sample times to ensure reliable control.

Test the response and resolution of measuring the fan's RPM by changing the Sample Time in StampPlot (and tabbing off). Complete Table 4-3 by entering the specified Sample Time in StampPlot and completing the table for 2 values of RPM. Calculate the resolution - the change in RPM per count. An example entry is completed for you:

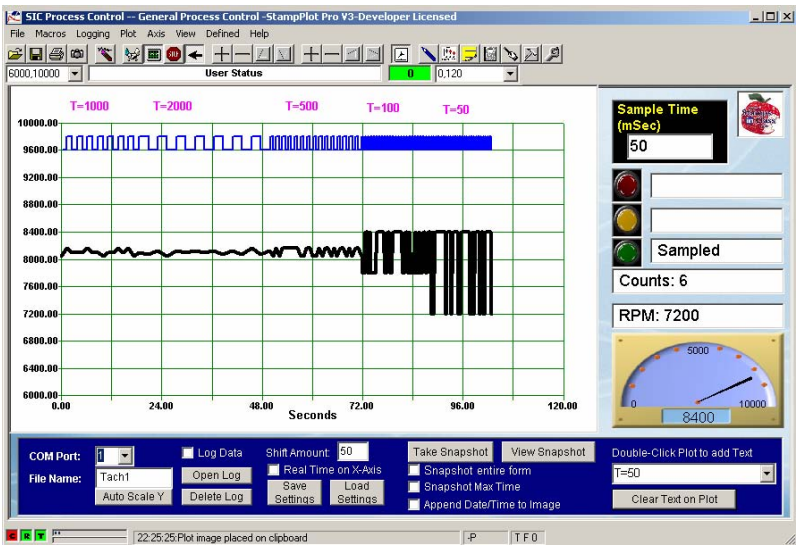
Resolution= (change in RPM)/(change in Count) = (8100−8040)/(135−134) = 60

Table 4-3: Resolution and Sample Time

Sample Time, ms	Count	RPM	Count	RPM	Resolution in RPM
1000	*135*	*8100*	*134*	*8040*	*60*
1000					
2000					
5000					
10000					
500					
200					
100					
50					

Figure 4-22 shows a plot using various sample times. Note the amount of change in RPM resolution for the various sample times. A small change in counts resulted in a very large change for RPM.

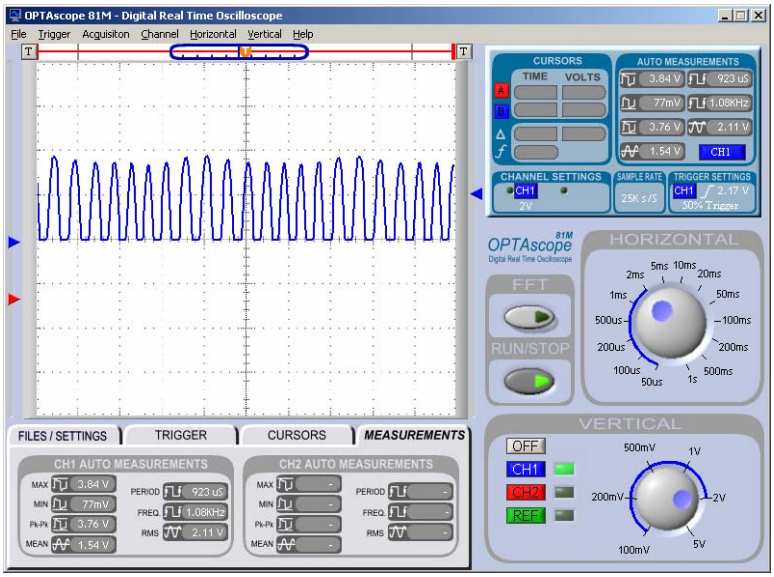
Figure 4-22 Fan RPM Plot with Varying Sample Durations



One way to increase the resolution without having to sample longer is to increase the number of counts per revolution. Using an encoder wheel with two light/dark pairs will double the number of counts and increase the resolution.

Compare the earlier waveform in Figure 4-21 using the 1 cycle/revolution encoder to Figure 4-23 using the 8 cycles/revolution. As the number of encoder patterns increases, more counts occur during the same time period.

Figure 4-23 Waveform of Phototransistor Output
for 8 Cycle/Revolution Encoder



√ Place the encoder wheel for 2 Cycles/Revolution on the fan, doubling the number of counts for the same unit of time.

√ In the code, change the value of the constant **`CyclesPerRev`** to 2.

```
CyclesPerRev    CON   2
```

√ Complete Table 4-4 for the sample times listed.

√ Repeat for all the encoder wheels.

Troubleshooting Tip: At some point the BASIC Stamp may read an RPM of zero or abnormally low. If this occurs, complete Activity #7; then return to finish this table and Activity #6.

Table 4-4: Tachometer Resolution with Different Encoders												
	2 Cycles/Revolution			**4 Cycles/Revolution**			**6 Cycles/Revolution**			**8 Cycles/Revolution**		
Time	**Count**	**RPM**	**Res.**	**Count**	**RPM**	**Res.**	**Count**	**RPM**	**Res.**	**Count**	**RPM**	**Res.**
1000												
5000												
100												
50												

Challenge4-6: Analyzing a Waveform

√ Figure 4-24 is a waveform capture of an encoder with the time base set at 1 ms/Div. From this waveform determine:

- Period (ms): _____
- Cycles/Second (Hz): _____
- Encoder Wheel used (assume 8040 RPM): _____

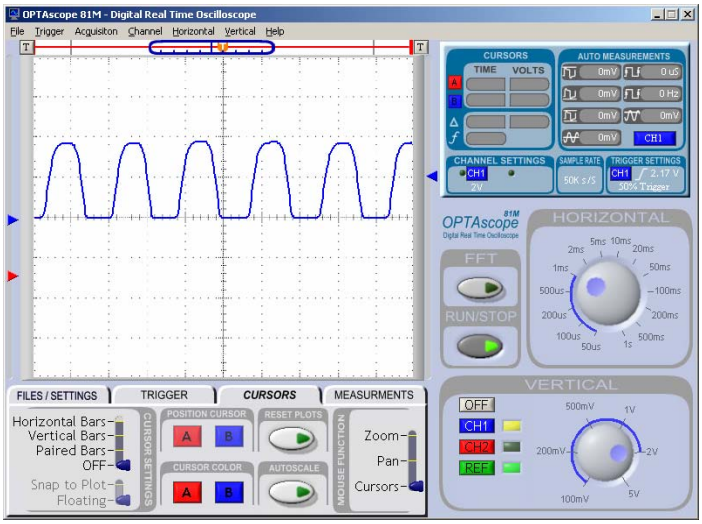


Figure 4-24
Waveform capture at 1 ms/Div

ACTIVITY #7: INCREASING SENSOR RESPONSE

While testing the various encoder wheels in Activity #6, you probably encountered a point where the RPM and count is abnormally low or zero. The response time of the sensor is insufficient to read the rapidly changing light levels in order to produce a sufficient change in signal level for the BASIC Stamp to count.

How can the response time be increased? Recall from Chapter 3 that the response of a transistor is largely dependent on the size of R_C. Can a smaller value of R_C be used and still drop asufficient voltage? If R_C is too small, there may be an insufficient voltage drop to cross the threshold and produce a LOW signal when in conduction.

The circuit needs a low value of R_C, but also needs an output that swings fully between Vss and Vdd. One solution is to use a Darlington Pair arrangement, like the one shown in Figure 4-25. A first stage transistor, Q_1, is used to provide base current for a second stage transistor, Q_2. Q_1 only needs to provide sufficient current to drive the second stage into saturation. This allows a fast response time by lightly loading the first stage, but also a high voltage swing, or gain using the second stage.

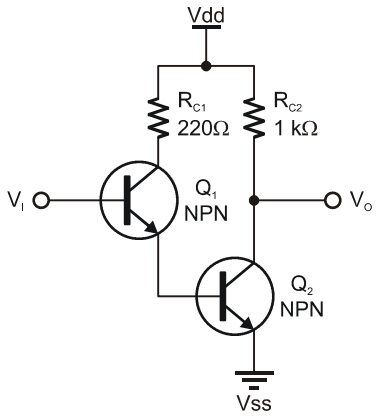


Figure 4-25
Example Darlington Pair Arrangement

DO NOT BUILD

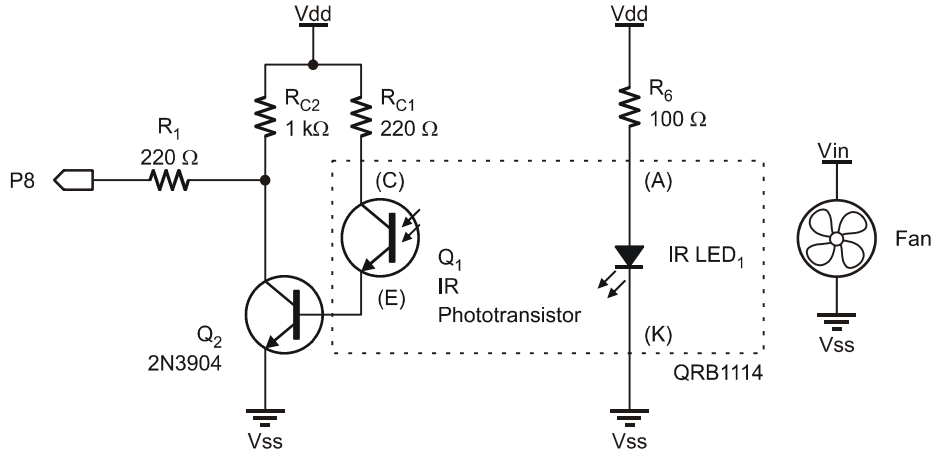
Often R_{C1} is not used in Darlington pairs, and the collector of Q_1 is directly connected to Vdd. It is used in this circuit as a safety measure for the components.

Additional Components Required:

(1) 2N3904 Transistor
(1) 220 Ω resistor
(1) 1 kΩ resistor

- √ Leave the ADC circuit on the board. It is not used in this activity but will be again later. It is not shown in the schematic but remains in the wiring diagram.
- √ Modify the circuit as shown in Figure 4-26 and Figure 4-27.

Figure 4-26 Opto-Reflective Switch Using Darlington Pair - Schematic



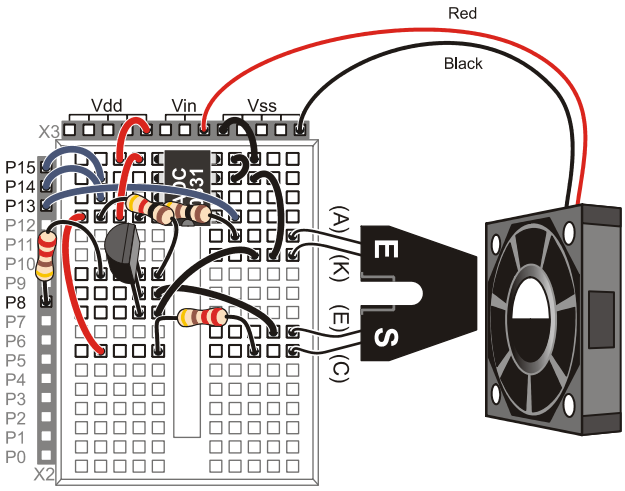


Figure 4-27
Opto-Reflective Switch Using Darlington Pair

Wiring Diagram

√ Re-run Tachometer.bs2, and close the Debug Terminal.
√ Run StampPlot macro pc_gen_process.spm.
√ Complete Table 4-4 in Activity #6 using the higher cycles/count encoder wheels if not yet complete.

You may have to re-adjust the distance between the QRB1114 and the fan when using the Darlington pair circuit.

Challenge 4-7: Adding Speed Indicators

√ Add an active-high red LED to P11, and a green one on P10.
√ Save Tachometer.bs2 under a new name.
√ Add code that will light the Green LED if the RPM is greater than 7500 RPM (or 500 RPM below your normal RPM). Light the red LED if the speed is less than that value.

CONCLUSION

The opto-reflective switch combines an emitter (LED) and detector (phototransistor) in a single package for the detection of near-range objects. The phototransistor acts similar to a BJT transistor, except that the base current is controlled with light instead of voltage. Other optical sensors use photodiodes or PIN diodes that have much better switching action.

Using the opto-reflective switch requires providing sufficient current to the LED to emit light and conditioning the output to produce a voltage when an object reflects the emitted light to the detector. The sizing of the collector resistance is critical in achieving the voltage swing required for input to the BASIC Stamp. Increased switching action can be obtained by using a Darlington Pair arrangement, decreasing the size of the collector resistance needed. A small current from the phototransistor is used to bias a second transistor acting as a switch.

Sequential processes perform a specific sequence of actions based on input or timed events. In this chapter, the opto-reflective switch and pushbuttons were used for events such as performing a batch mix operation or operating a diverter for box loading.

Counting with an opto-reflective switch or other detectors can be either very slow (object counting) or very fast (tachometer). There are issues involved in both types, including the need to use edge detection, increase response, and determine the resolution needed. Resolution of high speed counting can be performed by either increasing how long a count occurs (which decreases the program scan time) or by performing a greater number of counts per revolution, based on the encoder used (which requires greater detector response time).

One last point concerns the use of IR emitters and detectors used with the Boe-Bot robot and TV remote controls. These are more complex detectors in that they will only respond to infrared light that is cycling on and off at roughly 38 kHz.

SOLUTIONS TO CHAPTER 4 CHALLENGES

Challenge 4-1 Solution

Results are dependent on materials tested. Which colors absorbed more infrared light? Did shiny material reflect better?

Challenge 4-2 Solution

√ At end of code, add:

```
Shutdown:
  IF (Stop_SW = 0) THEN          ' If stop pressed
    LOW Fill                              ' Turn off all
    LOW Mix
    LOW Drain
    DO
      GOSUB Display           ' plot data
    LOOP                      ' loop until reset
  ENDIF
RETURN
```

√ Within each loop, add:

```
GOSUB Shutdown
```

Challenge 4-3 Solution

Your partial code may look like this:

```
  DEBUG "!READ (txtData)",CR        ' Read time to mix from plot
  DEBUGIN DEC Mix_Seconds           ' Accept data, put in Mix_Seconds.

  DEBUG "!LOGD Starting Fill",CR
 .
 .
  LOOP

  DEBUG "!LOGD Starting Mix",CR
  LOW Fill                          ' Stop fill
 .
 .
  NEXT
  DEBUG "!LOGD Starting Drain",CR
  LOW Mix                           ' Stop mixing
 .
 .
  DEBUG "!O Stat2 = Total Gallons:", DEC Total_Mixed,CR
  DEBUG "!LOGD Batch Complete - (Stat2)",CR
LOOP
```

Challenge 4-4 Solution

Your code may look like this:

```
Run_Control:
  IF (Start_SW = 0) AND (Stop_SW = 1) THEN  ' Start pressed, Stop NOT pressed
```

Challenge 4-5 Solution

Assuming it requires 3 seconds for a box to pass, add this code block to the **`Count_Box`** subroutine:

```
 IF (Opto_SW = 0) THEN          ' Box detected?
 PAUSE 500                     ' Allow to stabilize
  IF (Opto_SW = 0) THEN        ' Box STILL detected?
    Edge_Flag = 1              ' True, set flag
    HIGH Box_Det               ' Energize LED
  ENDIF
ENDIF
```

Challenge 4-6 Solution

- Period : T = 1.8 Divisions **x** 1 ms/Division = 1.8 ms
- Cycles/Sec (Hz): 1/period = 1/1.8 ms = 555.6 cycles/second
- Encoder: 555.6 cycles/sec **x** 60 sec = 60 seconds/minute = 33,336 cycles/minute (33,336 cycles/minute)(8040 revolutions/minute) = 4 cycles/revolution

Chapter 4-7 Solution

√ To declarations, add:

```
Red_LED     PIN   11
Green_LED   PIN   10
```

√ To the Main Routine's **`DO...LOOP`**, add a call for a new subroutine named **`RPMIndicator`**, right after the call for **`ReadTach`**.

```
DO
  GOSUB ReadSP
  GOSUB ReadTach
  GOSUB RPMIndicator
  GOSUB DisplayData
LOOP
```

√ Add the new subroutine **`RPMIndicator`**:

```
RPMIndicator :
  IF (RPM > 7500) then
    HIGH Green_LED
    LOW Red_LED
  ELSE
    LOW Green_LED
    HIGH Red_LED
  ENDIF
  RETURN
```

Chapter 5: High Current Drive and PWM Control

Earlier chapters discussed means to use transistors, standard or optical, to condition an input signal for the BASIC Stamp. Equally important in process control is the ability to drive large loads such as fans, motors, and heaters as actuators for the process.

The output of the BASIC Stamp is 0 V or 5 V and limited to 20 mA. The output voltage and current are sufficient for driving small devices, such as LEDs, piezoelectric speakers and integrated circuits. The output is not sufficient to drive larger loads directly, such as small DC motors or very large 115 VAC motors, but with output drive interfacing, the BASIC Stamp may very well be used for control of these much higher voltage and current devices.

Even though the output of the BASIC Stamp is digital – on or off – there exist methods to provide variable drive to devices. Examples include controlling the speed of a motor, the energy output of a heater or the brightness of a lamp.

ACTIVITY #1: DC FAN ON-OFF CONTROL

The fan supplied with your kit is a brushless DC motor rated at 12 VDC and draws 0.09 amps (90 mA). The fan will operate with a lower voltage but will run slower and reach a cutoff voltage around 3.5 VDC.

How can the 5 V, 20 mA output of a BASIC Stamp I/O control the fan? One means is by using that wonderful device – the transistor. Recall that the transistor can be turned on (saturated) if sufficient current is applied to the base to act as an electronic switch controlling a much higher current as shown in Figure 5-1.

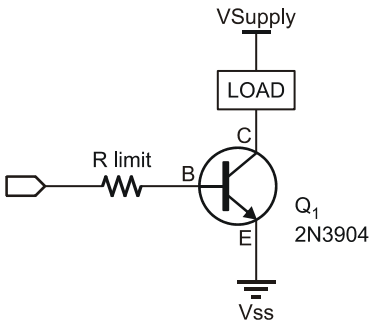


Figure 5-1
Bipolar Junction Transistor (BJT) Current Driver

DO NOT BUILD

But before testing this out, let's ensure the devices can meet the specifications with the fan acting as the load.

- **Q**. Can the transistor collector handle the current requirements of the fan, which is 90 mA?
- **A**. Yes, the 2N3904 is rated for 200 mA continuous.

- **Q**. Can the transistor handle the collect-emitter voltage required by the fan, which is 12 V?
- **A**. Yes, the 2N3904 is rated for 40 V_{CE}.

- **Q**. Can the BASIC Stamp provide sufficient base current to drive the collector at 90 mA?
  A. With a minimum h_{FE} of 100 for the 2N3904:

  $I_B = I_C/h_{FE} = 90\ mA/100 = 0.9\ mA$

The BASIC Stamp can easily handle the base current required. Taking into account good engineering practices of working under worst case conditions and driving the devices to only ½ to ¾ their maximum rating, the specifications look good for drive control.

Parts Required

(1) Resistor – 100 Ω
(4) Resistors – 220 Ω
(1) Resistors – 1 kΩ
(1) Opto-Reflective Switch – QRB1114
(1) LED - Red
(2) NPN Transistors – 2N3904
(1) Brushless DC Fan
(1) BS170 MOSFET Transistor

√ Build the circuit shown in Figure 5-2 and Figure 5-3, leaving the ADC circuit on the board for later use. Set aside the BS170 MOSFET transistor for the moment.

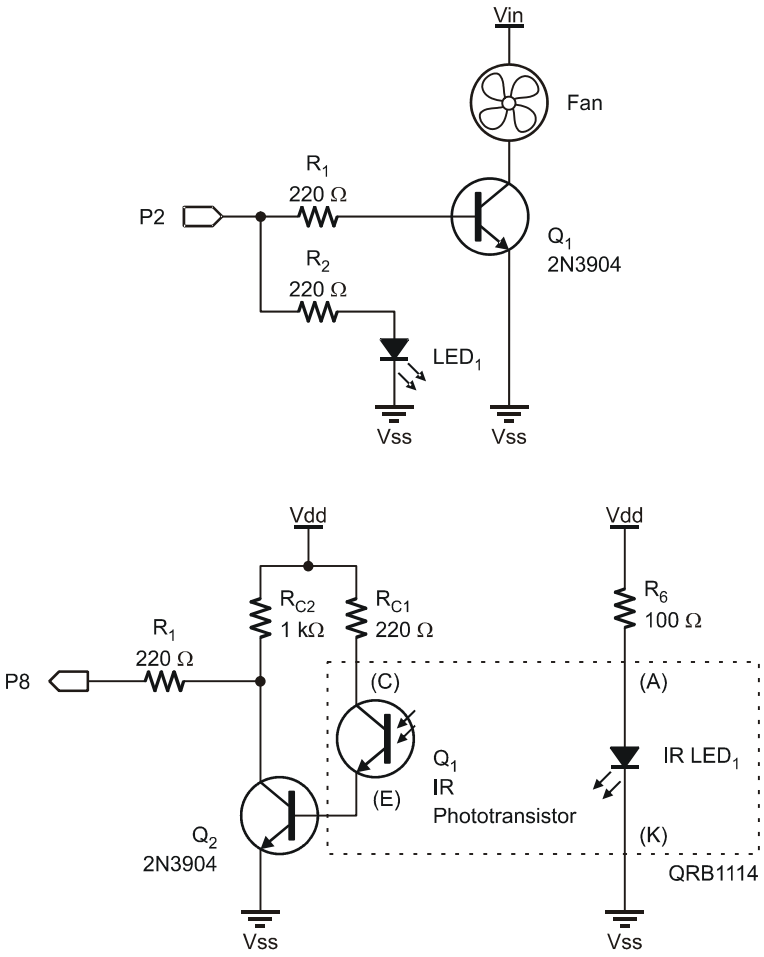


Figure 5-2
Switched Fan Drive Schematic

Note: The ADC circuit is not shown, but leave it on the board for future use.

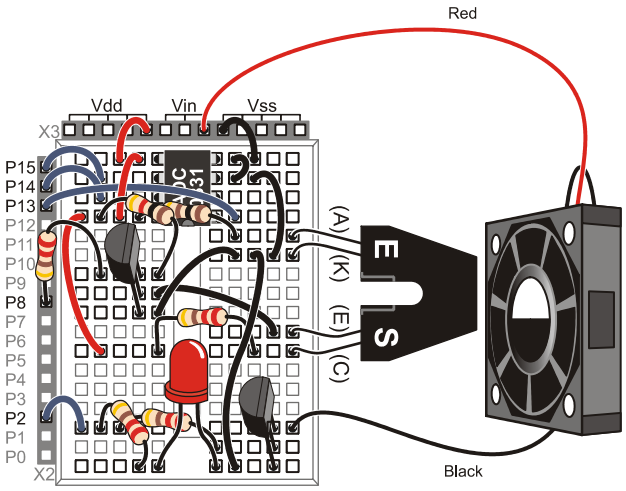


Figure 5-3
Switched Fan Drive Wiring Diagram with 2N3904 Transistor

Note: The ADC circuit is left in place for future use.

√ Enter, save and run FanOnOffControl.bs2.
√ Test the program. The fan should cycle on and off every 5 seconds.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - FanOnOffControl
' Controls 12Vdc fan with transistor driver
' {$STAMP BS2}
' {$PBASIC 2.5}


' -----[ Declarations ]-------------------------------------------------------
Fan     PIN           2             ' Fan Driver I/O
OnTime  CON           5000          ' Time to leave fan on
OffTime CON           5000          ' Time to leave fan off
' -----[ Main Routine ]-------------------------------------------------------
DO
   HIGH FAN           ' Energize fan
   PAUSE OnTime       ' Pause while running
   LOW FAN            ' De-energize fan
   PAUSE OffTime      ' Pause while off
LOOP
```

Next, consider the power Metal-Oxide Semiconductor Field Effect Transistor (MOSFET) drive circuit in Figure 5-5. The MOSFET is driven into saturation by applying gate voltage. A positive 5 volts from the BASIC Stamp output is sufficient to place the MOSFET in an "ON" state. When the device is fully saturated, its ON-state resistance

Rs-on is typically less than 1 ohm. Applying a low (0 V) 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 an excellent 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 heat sinks, the BS170 can handle load currents up to 5 amps, or 500 mA continuous without heat sinks. 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 MOSFETs and BJTs. A diode is often used to provide protection for the transistor when driving inductive loads such as these. However, a diode is not necessary for the small brushless motor used in our experiments.

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.

BJTs and MOSFETs are compared in Table 5-1.

Table 5-1: Comparison of BJT and MOSFET Transistors

Characteristic	BJT	MOSFET
Load Current	Lower	Higher
Load Voltage	Lower	Higher
Control Current	Higher	Practically zero
Switching Speed	Lower	Higher
Linear Response	Good	Poor

While the MOSFET makes an excellent electronic switch, it is not suited for small signal amplification where linear control is required, such as in audio.

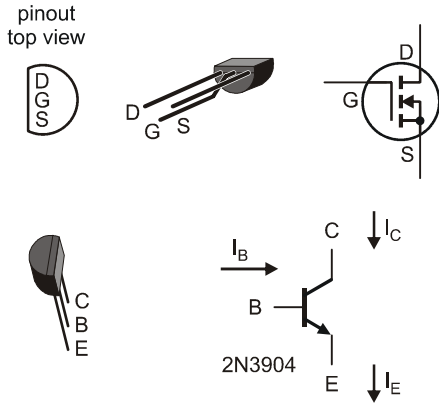


Figure 5-4
Top: BS170 MOSFET
D = Drain
G = Gate
S = Source

Bottom: 2N3904 BJT
C = Collector
B = Base
E = Emitter

√ Replace the 2N3904 transistor that is connected to the fan with the BS170 MOSFET transistor to form the circuit shown in Figure 5-5. Be very careful to observe the proper pin placement.

WARNING! The pinout for the BS170 is the reverse of the pinout for the 2N3904. The flat side of the BS170 will be facing the opposite direction from the way the 2N3904 was facing.

√ Test the circuit.

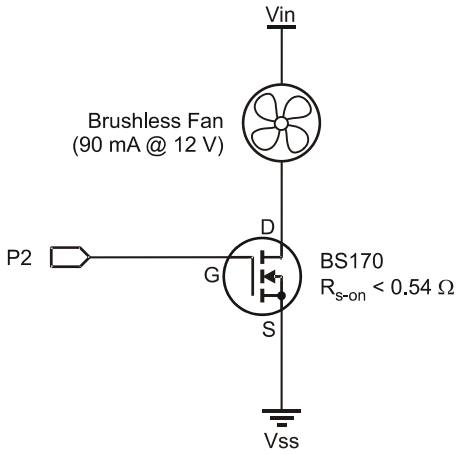


Figure 5-5
FET Driver and
BS170 Circuit

Challenge 5-1: Fast On-Off Cycling

- √ Save the program FanOnOffControl.bs2 under a new name.
- √ Change the `OnTime` and `OffTime` to 5 (5 ms) each and run the program. How fast is the fan running compared to previously?
- √ Test other values where the on and off times add up to 10, such as `OnTime` at 7 and `OffTime` at 3. Note: At low values of `OnTime` (< 4) the fan may not run.

What happens to the fan's speed and to the brightness of the LED? Can you determine why it has these effects? This is the subject of Activity #2, Pulse Width Modulation.

ACTIVITY #2: PULSE WIDTH MODULATION

In the previous activity's Fast On-Off Cycling challenge, the ON time and OFF time were very short. What effect did it have on the fan and the LED? The fan ran at a slower speed and the LED was dimmer. This activity explores Pulse Width Modulation (PWM) in the control of devices.

Parts Required

Same circuit as Activity #1, using the BS170 MOSFET transisitor.

When the fan is running at 12 V, it draws 90 mA of current. This equates to a power draw of 1.08 watts:

$$12\ V_{DC} \times 90\ mA = 1.08\ W.$$

When the fan is off, it draws 0 watts. Over a short period of time, if the fan is on half the time, and off half the time, on average how much power is the fan able to draw? It's able to draw one-half of the maximum power, or 0.54 watt. The fan will run at half speed if it only receives one-half the power it requires.

Another way to look at it is in terms of voltage. If a DC motor is effectively supplied only half the normal voltage, it will run at approximately one-half full voltage speed. The average voltage over time is one-half the maximum when the output is high only 50% of the time.

"Duty cycle" compares the percentage of time the output is high, or on, to the total period, or cycle time. If the period is 10 ms, and the output is high for 5 ms:

Duty Cycle = On-Time/Period = 5 ms/10 ms = 0.5 x 100 = 50%

Figure 5-6a is a waveform with a 50% duty cycle. The average voltage supplied is 50% of the maximum voltage, or 6 V in this case.

$V_{avg} = V_{max}$ x Duty Cycle = 12 V x 50% = 6 V

The fan will be running at 50% of rated speed, or around 4000 RPM.

Next, consider the duty cycle in Figure 5-6b. At what speed would the fan effectively be running for this duty cycle? 25% rated speed or 2000 RPM.

Notice that the time of the wave did not change, only how long the output was high during that time. The lower the duty cycle, the lower the average voltage. The higher the duty cycle, the higher the average voltage.

When running the Fast On-Off Cycling challenge in Activity #1, did the LED appear to dim with lower duty cycles? In fact, the LED did not dim. When the output is high the LED is just as bright as it normally is. Your eye averaged out the light you were receiving between on and off to make it appear dimmer. Try this – darken the room while running the Fast On-Off Cycling challenge and wave the board up and down while watching the LED. You should see the LED at normal brightness as a series of dots as it blinks on and off while being moved.

Infrared LEDs used in TV remotes have a continuous current rating of 100 mA. This is based on how much power the LED can safely handle due to heat dissipation. If a very small duty cycle is used to drive the LED with a much higher current, the average power will be very low but the LED will be much 'brighter' during the on-time. In fact, the LEDs in TV remotes can handle 1 amp for very short bursts! This allows a strong signal to reach the TV receiver without damaging the LED.

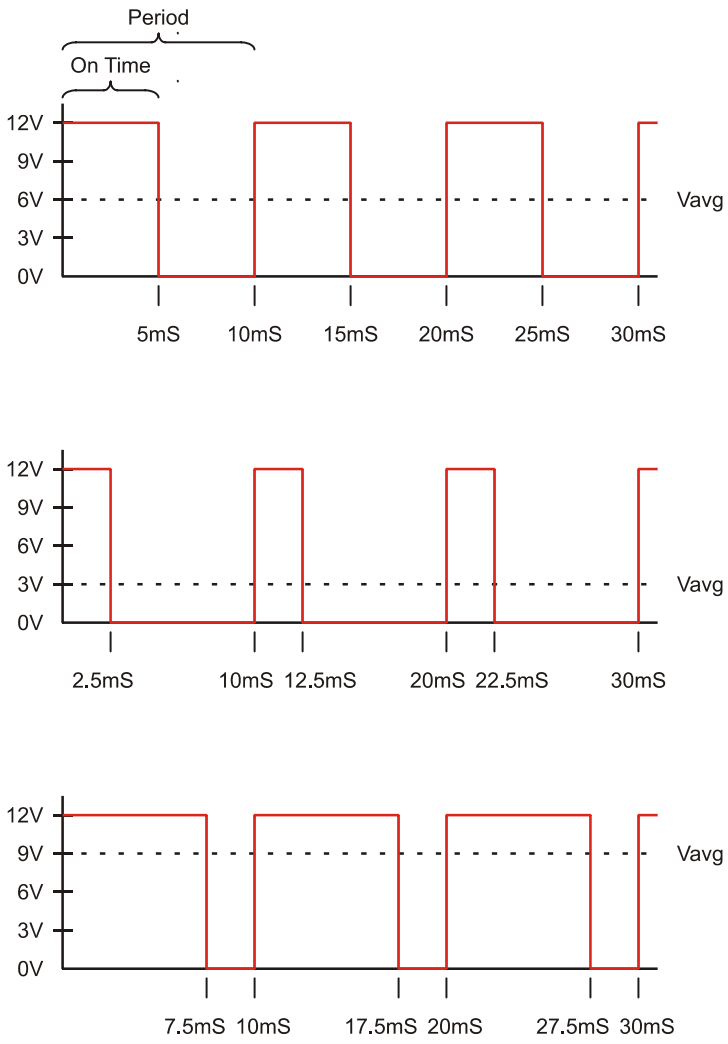


Figure 5-6
Duty Cycles

a.) 50% Duty Cycle

b.) 25% Duty Cycle

c.) 75% Duty Cycle

Ready to run PWM tests on your fan?

Example Program PwmTest.bs2

- √ Attach the 4-cycles/revolution encoder to the fan (cut-outs can be found in Appendix A.
- √ Enter, save and run PwmTest.bs2, then close the Debug Terminal.

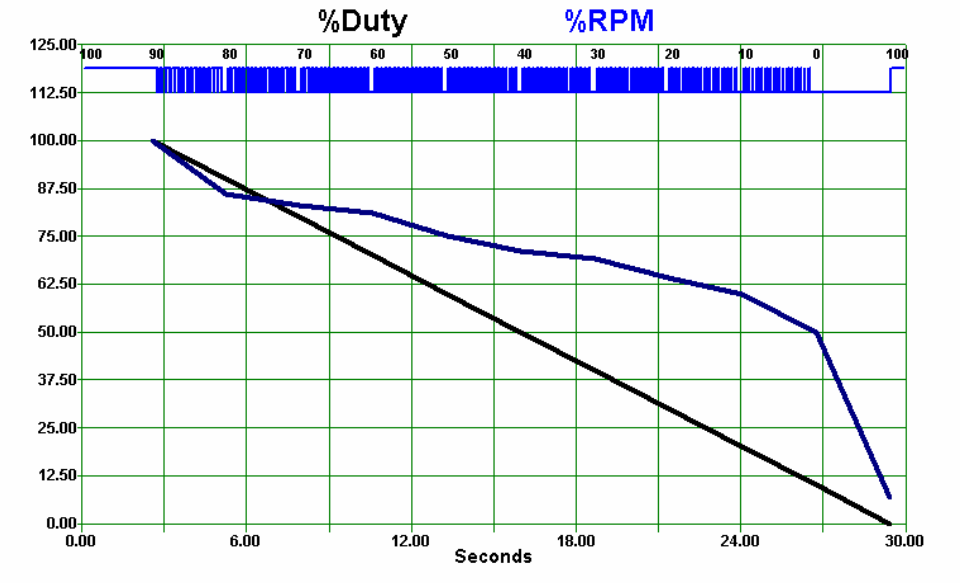
```
' -----[ Title ]-------------------------------------------------------------
' Process Control -  PwmTest.bs2
' Uses Pulse Width Modulation to control fan speed
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
Opto_SW         PIN     8         ' Opto-Reflector
Fan             PIN     2         ' Fan drive pin

Opto_Count      VAR     Word      ' Count from opto-reflective switch
RPM             VAR     Word      ' Calculated RPM
SP_Data         VAR     Word      ' Data returned from StampPlot

CyclesPerRev    CON     4         ' Encoder used

Duty            VAR     Byte      ' Duty cycle value 0-255
Max_Count       VAR     Word      ' Maximum count for calculating Per_RPM
Per_RPM         VAR     Word      ' Percent of RPM 0-100
x               VAR Byte          ' General counting variable

' -----[ Initialization ]----------------------------------------------------
PAUSE 500                         ' Connection stabilizing time

' -----[ Main Routine ]-------------------------------------------------------
GOSUB Calibrate

DO
  FOR Duty = 100 TO 0 STEP 10               ' Drive PWM from 100% to 0%
    DEBUG "!TEXT (PTIME),98A,0.8A,(BLACK),",DEC Duty,CR  ' Label plot
    GOSUB Apply_Drive
    COUNT Opto_SW,100,opto_count            ' Measure counts for 100mSec
    Per_RPM = Opto_Count * 100 / Max_Count  ' Calculate percent RPM
    DEBUG DEC Duty,",",DEC Per_RPM ,CR      ' Plot data
  NEXT
LOOP

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

Apply_Drive:                            ' Apply drive with a 100msc period
    FOR x = 1 TO 15                     ' Apply for 15 repetitions
      IF duty > 0 THEN HIGH Fan         ' Check if any HIGH time
      DEBUG IBIN OUT2,CR                ' Plot digital state of drive
      PAUSE Duty                        ' Apply for amount of high time
```

```
      IF duty < 100 THEN LOW Fan          ' Check if any LOW time
      DEBUG IBIN OUT2,CR                  ' Plot digital state of drive
      PAUSE 100-Duty                      ' Apply for rest of 100mSec
    NEXT
RETURN

Calibrate:                                ' Finds max speed to calculate %RPM
  HIGH Fan                                ' Energize fan
  PAUSE 3000                              ' Allow fan to come up to speed
  DO
    COUNT Opto_SW,100,Opto_Count          ' Count Pulses
    RPM = Opto_Count * (60000 / 100) / CyclesPerRev ' Calculate RPM
    DEBUG CR,"!REQD 0,30,Maximum RPM=", DEC RPM, ' Request speed verification
          "\nEnter 1 if RPM OK.",
          "\nEnter 0 if not OK.",CR
    DEBUGIN DEC SP_Data                   ' Accept user's response
    Timeout:
    PAUSE 100
  LOOP UNTIL (SP_Data = 1)                ' Repeat if user did not enter 1
  Max_Count = Opto_Count                  ' Save maximum count
  DEBUG "!RSET(CR)!RSET",CR               ' Reset plot
RETURN
```

- √ Run StampPlot macro sic_pc_pwm_test.spm.
- √ Position the fan in front of the sensor.
- √ Connect StampPlot.
- √ A message box should appear asking if the fan's RPM is correct based on past tests.
- √ If yes, enter 1 and press Return.
- √ If no, adjust the fan's position to obtain a better reading and enter 0 to test again.
- √ The PwmTest.bs2 program will run through the duty cycles at 10% increments from 100 to 0.

Figure 5-7 on the next page is a sample screen shot.

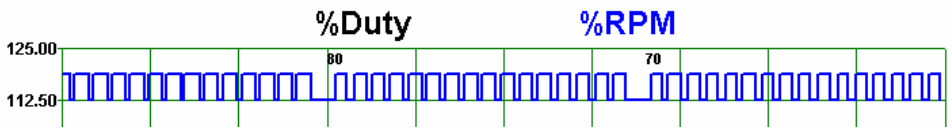
Figure 5-7 PWM Test with Manual Drive



As seen in Figure 5-8, a 100% drive, or duty cycle, is applied, and a sample of RPM is taken. The %Duty and %RPM are plotted (and updated in the gauges). Next, a 95% duty cycle is applied and RPM sampled.

Note that the digital trace shows the switching of the fan. Figure 5-8 is a close-up of the switching. What happens to the HIGH time as the drive is decreased?

Figure 5-8 PWM Drive Cycling



Does the fan produce a linear change in speed in relation to the drive? As we can clearly see, it does not. However, this isn't the circuit's fault. It's a problem with the code in that the fan is not provided a very clean duty cycle. Trying to switch the drive output on and

off quickly in code is not very efficient in that time is wasted looping and performing other code. The 20% drive is not truly 20%.

Program Discussion

In the **Calibrate** subroutine, input from the user verifies that the fan is at the correct speed and used to calibrate for the maximum speed.


```
DEBUG "!REQD 0,30,Maximum RPM=", DEC RPM,
      "\nEnter 1 if Reading OK.",
      "\nEnter 0 if not OK.",CR
```

!REQD is a StampPlot instruction to request data from the user through a message box window and takes the form of:

!REQD ***initial value, time to display in seconds, message***

An example string received from the BASIC Stamp would be:

```
!REQD 30,0,Maximum RPM=7850.\nEnter1 if RPM OK.\nEnter 0 if not OK.
```

\n , the new-line character, is used to place text on the next line.

In this example, the initial value is 0 (false), display for 30 seconds, and the message shows the current RPM. The **DEBUGIN** instruction is then used to accept the returning data and then test the user's response.

A transfer function is used to scale the RPM, or counts, to a percentage in the **DO...LOOP**. A simple transfer function is calculated by multiplying by the maximum output and dividing by the maximum input. Transfer functions are covered in greater detail in Chapter 6.

In this case, the change in output is 0 to 100% and the change in input is 0 to the maximum counts for the fan at maximum speed. In the calibration routine, once the user indicates the fan is at speed, the maximum count value is stored and used later to calculate %RPM.

```
Per_RPM = Opto_Count * 100 / Max_Count
```

If RPM were used, the math would have exceeded our math limits: 8000 x 100 = 80,000.

The BASIC Stamp works in integer math. That is, values with decimal places will be truncated, or have the decimal portion dropped. Also, any intermediary math calculation cannot exceed 65,535.

Using the PWM Instruction

A much more efficient means to produce a clean PWM drive is through the use of the **`PWM`** instruction. Here is the syntax:

PWM *Pin, Duty, Cycles*

The ***Duty*** argument can be 0-255. A ***Duty*** of 0 corresponds to a 0% duty cycle, and a ***Duty*** of 255 corresponds to a duty cycle of 100%. So, for a 50% duty cycle, ***Duty*** would be halfway between 0 and 255. The ***Cycles*** argument can also be 0-255. When using the BASIC Stamp 2, ***Cycles*** specifies the duration of the PWM signal in 1 millisecond units.

√ Save PwmTest.bs2 under the name PwmCommand.bs2.
√ Replace the **`Apply_Drive`** subroutine with the following:

```
Apply_Drive:
   FOR X = 0 TO 15
     PWM Fan, Duty * 255 / 100, 100
   NEXT
RETURN
```

√ Download the program and repeat the PWM test.

My BASIC Stamp is resetting! If it appears that your BASIC Stamp is resetting, you might try placing a 10 µF capacitor across the Vin and Vss terminals of your Board of Education, but BE VERY CAREFUL!

WARNING: Always observe polarity when connecting an electrolytic capacitor **– reversing the leads can cause it to explode. Safety goggles are recommended.**

- Disconnect the power before adding the capacitor to the board.
- Connect the positive terminal to Vin and the negative terminal to Vss.
- Remember, the negative terminal is the lead that comes out of the metal canister closest to the stripe with a negative (–) sign.
- Double-check your circuit before reconnecting power.

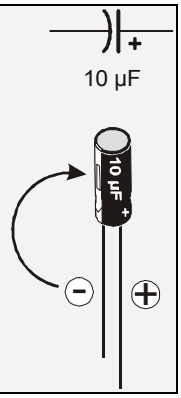


Figure 5-9 PWM Drive using the PWM Command

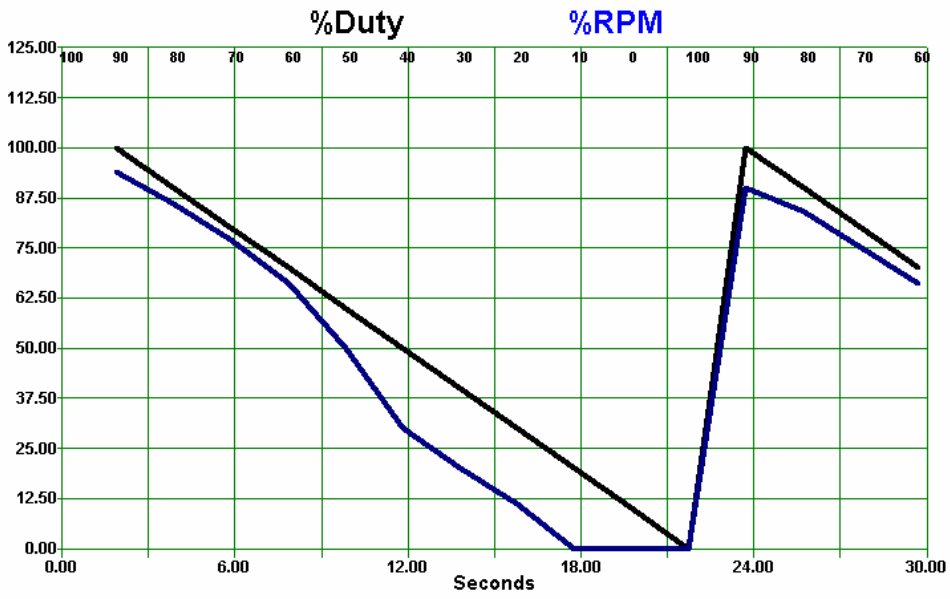


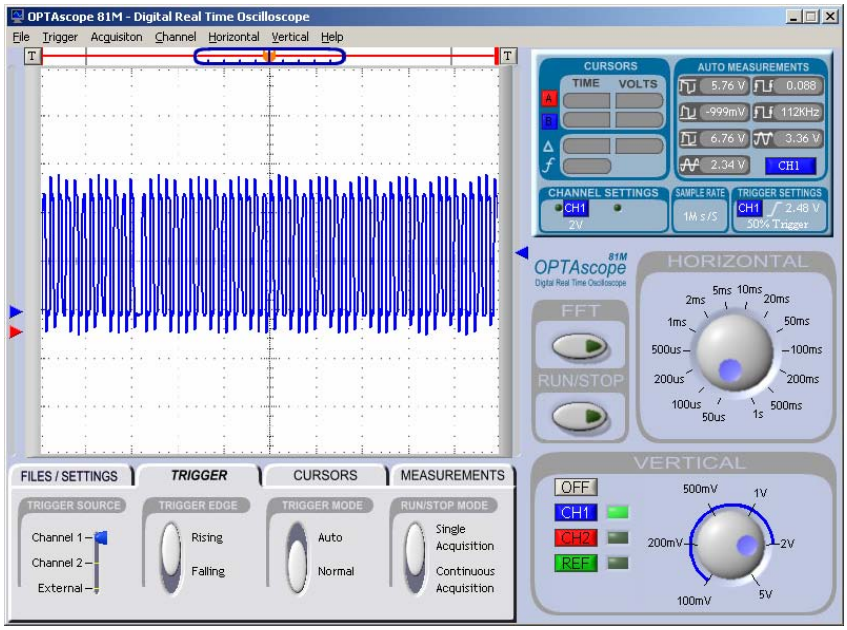
Figure 5-9 is a sample plot of the drive versus the RPM. The linearity and control is much better than the prior result, though the lower voltage limit of the fan is soon reached.

The PWM command controls the duty cycle for the given amount of time (up to 255 ms). As mentioned, the ***Duty*** argument range is 0 to 255. Since our drive is 0% to 100%, we must apply a transfer function to arrive at an appropriate value for the PWM instruction: ***Duty*** X 255 / 100.

Figure 5-10 is a capture of the waveform at the drive pin for a 50% duty cycle. While the BASIC Stamp does not produce as clean a duty cycle waveform as we have discussed, the average of the high time is still 50% as noted by the average voltage produced (2.34 is approximately 50% of 5 V). Also note the frequency of the waveform – 112 kHz. This very fast switching of levels produces the desired duty cycle.

A high-frequency duty cycle is not always desirable. Inductive loads, such as motors, limit how fast the current can change. With high-frequency duty cycles the current does not have time to build and will limit the response at lower duty cycle values.

Figure 5-10 Oscilloscope Capture of 50% Duty Cycle



Challenge 5-2: Modifying the Drive Loop

There are a few issues in driving the fan and obtaining good RPM reading in respect to drive. The fan has inertia, which limits how fast it can change speed. As drive is changed, the change in speed is not instantaneous. It takes a certain amount of time to bring the fan up to speed, and conversely to slow it down. In general, the greater the mass, the greater the inertia of the object. With a rotating element, the shape and diameter also comes into play when determining inertia.

Part A:
Currently in PwmCommand.bs2's **`Apply_Drive`** subroutine, the PWM is applied for 15 repetitions for a PWM time of 100 ms. This allows 1.5 seconds for the fan to reach a new speed prior to taking a reading.

- √ Save PWMTest.bs2 under a new name.
- √ Modify the **`Apply_Drive`** subroutine to apply PWM for 5 seconds prior to taking a measurement.

How does your plot change?

Part B:
Another issue in controlling and monitoring the fan's speed is the fact that drive is only applied while the PWM command is being executed. With this very short program, not much occurs outside of applying and measuring the drive. What if there was other control and monitoring action that had to occur? How would that affect the fan's speed?

- √ Modify the bolded lines of the PwmCommand.bs2's Main Routine:

```
DO
  Duty = 75
  DEBUG "!TEXT (PTIME),98A,0.8A,(BLACK),",DEC Duty,CR
  GOSUB Apply_Drive
  PAUSE 2000                                  'Simulated code delay time
  COUNT Opto_SW,100,opto_count
  Per_RPM = Opto_Count * 100 / Max_Count
  DEBUG DEC Duty, ",", DEC Per_RPM ,CR
LOOP
```

1. What happens to the fan's speed with the simulated delay time of 2 seconds?
2. If the delay in execution was required, but proper sampling of RPM after driving was needed, where would be a better place for the simulated code delay? Test your idea.

ACTIVITY #3: PWM FILTERING

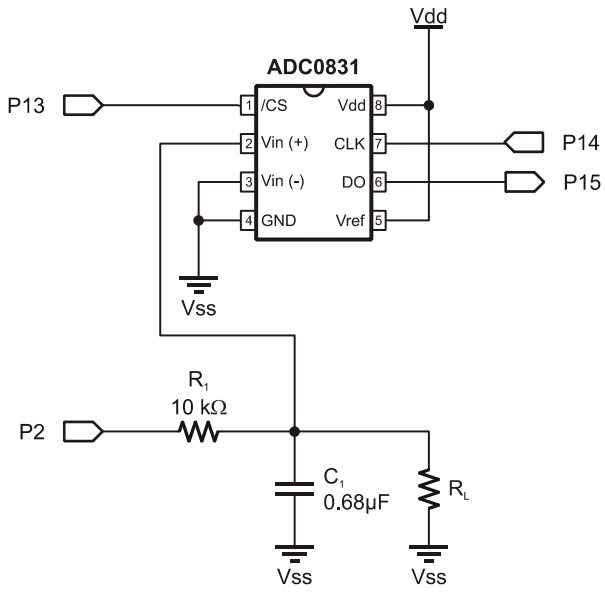
The inertia of the fan performed mechanical filtering, or damping, to run at various constant speeds with a quick cycling of high and low voltages. Your eye performed filtering of the LED blinking on and off rapidly to see a bright or dim LED. Electrical filtering of a changing voltage can be achieved by the use of a resistor-capacitor (RC) network.

A capacitor is used to hold a charge of electrons. An RC network, properly sized, can be used to filter the cycling PWM to a constant voltage.

Additional Parts Required

(1) 10 kΩ Resistors
(1) 100 kΩ Resistor
(1) 1 MΩ Resistor
(1) 0.68 µF Capacitor

- √ Remove the fan circuit connected to P2 (including its MOSFET transistor, LED, both 220 Ω resistors, and the fan itself).
- √ Replace it with the RC circuit as shown in Figure 5-11. **DO NOT** connect R_L initially.



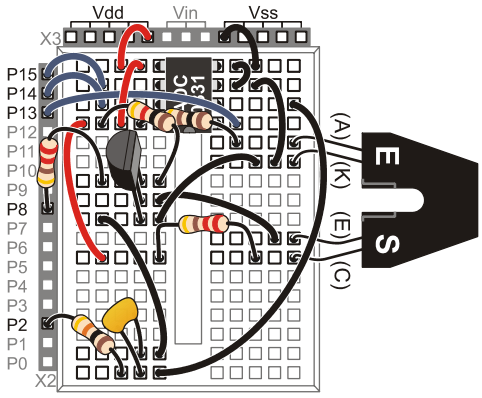


Figure 5-11
RC Filter of PWM Circuit

Top: Schematic

Bottom: Wiring Diagram

√ Enter, save and run the program PWMFiltering.bs2.
√ Open StampPlot macro sic_pc_pwm_filtering.spm
√ Connect and plot.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - PwmFiltering.bs2
' Control of PWM filtering circuits through StampPlot
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
ADC_DataIn   VAR Byte      ' Analog to Digital Converter data
PWM_Val      VAR Byte      ' Amount of PWM to apply as read from StampPlot
SampleAmount VAR Word      ' Amount of time to apply as read from StampPlot
SampleCount  VAR Word      ' Count of applied PWM

Fan          PIN 2         ' PWM Drive pin - For fan eventually
ADC_CS       PIN 13        ' ADC Chip Select pin
ADC_Clk      PIN 14        ' ADC Clock pin
ADC_Dout     PIN 15        ' ADC Data output

' -----[ Initialize ] ------------------------------------------------------
PAUSE 1000                 ' Connection stabilization

' -----[ Main Routine ]-----------------------------------------------------

DO
  GOSUB Read_SP             ' Read setting from StampPlot
  GOSUB DrivePWM            ' Drive output
  FOR SampleCount = 1 TO SampleAmount   ' Read ADC for number of samples
      GOSUB Read_ADC
      GOSUB PlotData        ' Plot data
  NEXT
LOOP

' -----[ Subroutines ]------------------------------------------------------
READ_SP:
  DEBUG "!READ (sldPWM)",CR        ' Request PWM slider val. from StampPlot
  DEBUGIN DEC PWM_Val              ' Accept value and store
  PAUSE 100
  DEBUG "!READ (sldSample)",CR     ' Request number for sample from StampPlot
  DEBUGIN DEC SampleAmount         ' Accept value and store
  PAUSE 100
RETURN

DrivePWM:
  DEBUG "^TEXT (PTIME),105A,0.8A,(BLACK),PWM\n",DEC PWM_Val,CR  ' Label plot
  PWM Fan, PWM_Val, 255            ' Drive PWM
RETURN

Read_ADC:                          ' Read ADC 0831
  LOW ADC_CS                       ' Enable chip
  SHIFTIN ADC_Dout,ADC_Clk, MSBPOST,[ADC_DataIn\9] ' Clock in data from ADC
  HIGH ADC_CS                      ' Disable ADC
RETURN
```

```
PlotData:                           ' Send voltage data to StampPlot
  DEBUG DEC ADC_DataIn ,",",
        "[",DEC ADC_DataIn,",*,0.0196]",CR
  PAUSE 50
RETURN
```

This program reads slider controls from StampPlot. One sets value the of the **`PWM_Val`** variable, which will be used as the **`PWM`** command's ***Duty*** argument, and the other sets the number of samples to measure (0-255) before re-applying the PWM drive.

- √ Test your control of the voltage by setting the PWM Value slider control in StampPlot to various positions.
- √ Set the "# Samples" slider to approximately 50.
- √ Adjust the PWM Value from 0 to maximum in increments of 25 as the plot progresses. Clicking to the far right of the slider will cause a change of 25.

Note the measured voltage as the duty cycle is increased as illustrated in Figure 5-12. The RC Network filters the PWM to the average of the voltage applied. What voltage is measured when the PWM Value control is set to 150? From the plot, the value is approximately 3 V. Does this make sense?

%Duty = PWM *Duty* **x** 100/255 = 150 **x** 100/255 = 58.8%
Vavg = Vmax x %Duty = 5 V **x** 58.8% = 2.94 V.

Or, looking at it in terms of PWM resolution, 5 V is covered over 255 steps of PWM, giving each PWM increment a resolution of:

5 V/255 = 0.0196 V.

At a **`PWM`** ***Duty*** of 150: 150 **x** 0.0196 = 2.94 V.

- √ Set the PWM Value control to 1 and note the voltage.
- √ Set the PWM Value to 2 and note the change in voltage.

The ADC used to measure voltage also has a resolution equal to that of the PWM, which may lead to results that are not reliable for true analysis. Measure the voltage with a separate voltmeter if possible.

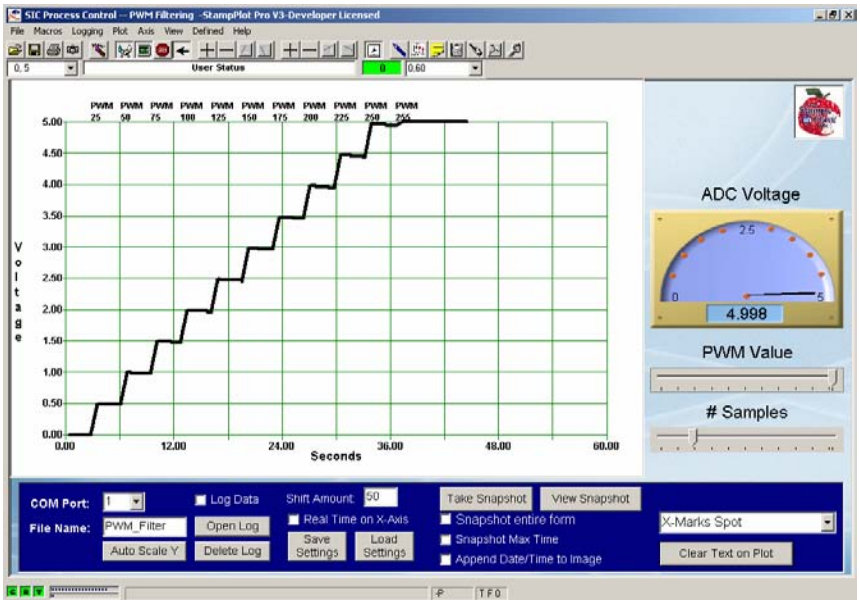
The resolution of a system can also be defined in percentage, where:

%Resolution = (Smallest Change/Total Range) x 100%
%Resolution = .0196 V/5 V x 100% = 0.392%

So, for a range of 5 V, the smallest change is 0.392% of 5 V, or .0196 V. Note also that this value is equal to the inverse of the number of steps in percent:

%Resolution = 1/255 x 100% = 0.392%

Figure 5-12 Measured Voltages and Various PWM Value Control Settings.



It is important to note that the voltage remains relatively constant during sampling. Once charged, the stored electrons have little place to go, but given sufficient time they will dissipate. Test the stability of the voltage by performing the following:

- √ Set the PWM Value control to midrange.
- √ Set the number of samples to maximum.
- √ Monitor the voltage over the 255 samples.

Does the voltage eventually start to decrease?

After the PWM has been applied, the BASIC Stamp output is high impedance, limiting the loss of electrons on the charged capacitor. The charge is also lost through the ADC sampling the voltage and leakage of the capacitor. Both of these are minimal because of the very high impedance of the ADC input and the dielectric, or insulation, used in the capacitor.

Can this average voltage be used to drive the fan even when the BASIC Stamp is busy performing other tasks? Unfortunately not. The high current draw (90 mA) will quickly bleed the charge off the capacitor. Let's test this using a 1 MΩ load, which at 5 V would have a current draw of 0.005 mA or 5 µA – MUCH less than the current draw of the fan.

- √ Connect the 1 MΩ resistor, R_L, as shown earlier in Figure 5-11.
- √ Repeat the stepped PWM test with a sample number of 25.

What occurs to the voltage with R_L in place? The voltage on the capacitor rapidly decreases as shown in Figure 5-11. The rate at which the voltage decreases can be calculated. The time-constant (τ) of an RC network is approximately 1.1 times the resistor value and the capacitor value or:

$$\tau = 1.1\ RC$$

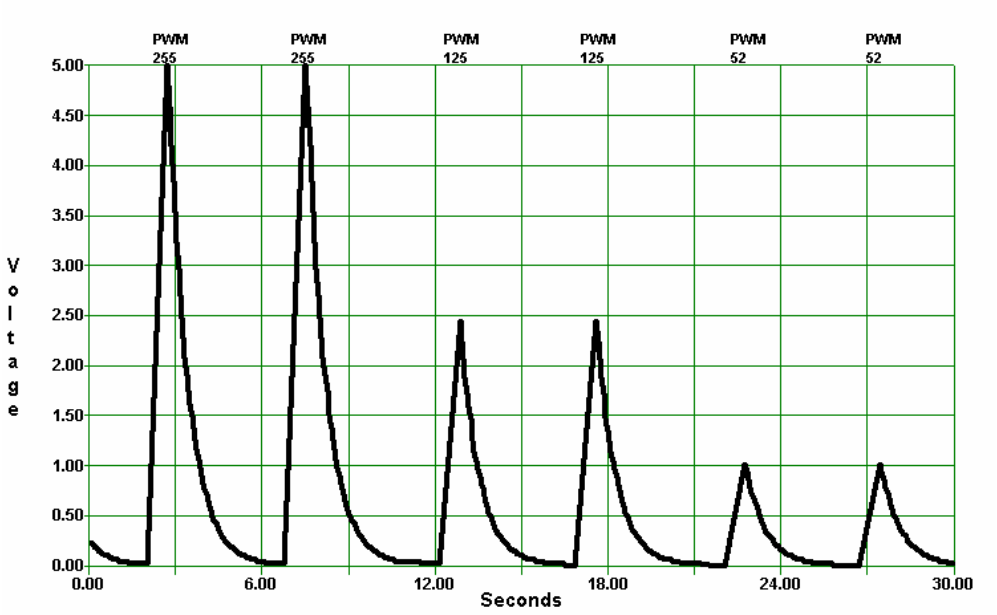
In this case,

$$\tau = 1.1(1\ M\Omega)(0.68\ \mu F) = 0.748 \text{ seconds.}$$

While we won't get much deeper into the math, the charge and discharge of a capacitor requires 5 time constants (5 τ) for the capacitor to nearly fully charge or discharge.
For this circuit:

$$5\ \tau = 5\ (0.748\ s) = 3.74 \text{ seconds.}$$

Figure 5-13 RC Network Discharge



√ Set the PWM Value control to 255 (100% duty cycle) and adjust the samples to measure the full time it requires the charge to dissipate to 0 V. It may be beneficial to log the data and read the times from the file for better accuracy, or the StampPlot Values window may be used to read the times under the mouse pointer (Menu View → Values).

As can be seen in Figure 5-13, the discharge time is approximately 4 seconds, which is not too far off from the predicted time, taking into account sampling rate and tolerances of components used. Also note that the discharge time, or time constant, of the RC network did not change significantly whether starting from 255 or 125. The response of the circuit is not dependent on where it started.

With the fan drawing 90 mA, how long will the charge last? It would last only an instant. While we are not able to drive the fan continuously, we are getting closer.

Challenge 5-3 : Modifying RC Values

Part A:
Based on the equation for the RC network time constants, what would be the result of changing R_L to 100 kΩ?

√ Calculate the discharge time.
√ Test your results.

Part B:
The capacitor is charged using the RC network of R_1 (10 kΩ) and C_1 (0.68 µF).

1. Based on this RC network, how long is required to fully charge the capacitor?

√ Replace R_1 with the 1 MΩ resistor.
√ Set the sample amount very low.
√ Cause large step changes in voltage. What occurs? Note that charging only occurs during that PWM burst, so times will not be accurate.

2. If R_1 were changed to 1 MΩ, would the capacitor fully charge with a PWM duration of 255 milliseconds? How long does it need?

√ When complete, replace R_1 with the 10 kΩ resistor.

ACTIVITY #4: OP-AMP BUFFER AND ACTIVE-FILTERS

An operational amplifier (op-amp) is a sophisticated network of transistors that has many applications when working with analog signals. The uses of op-amps are almost endless and entire texts have been written around configurations using these devices. We will explore several of the most popular configurations of op-amps beginning with the buffer.

Additional Part Required:

(1) LM358 Op-Amp

Some important characteristics of the Op-Amp pertinent to our discussion include:

- Extremely high input impedance (theoretically infinite).
- Extremely low output impedance (theoretically zero).
- The output will be driven in an attempt to drive the inverting input (−) voltage equal to the non-inverting input (+) voltage.

Figure 5-14 shows the schematic symbol for an op-amp, though most times the supply voltage connections are not shown. Figure 5-14 also shows the pins of the LM358 dual op-amp.

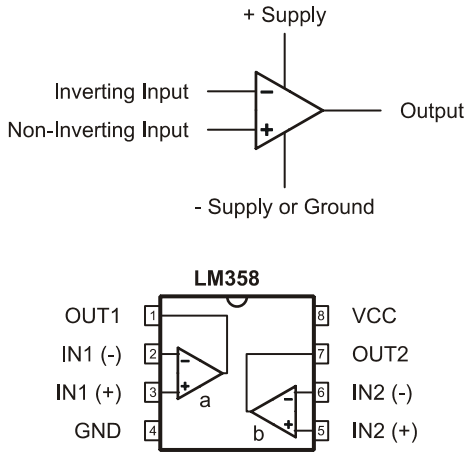


Figure 5-14
The LM358 Op-Amp

Schematic Symbol (top) and Pin Map (bottom)

Figure 5-15 is a buffer configuration. With $V_I = 2$ V on the non-inverting input, what must the output be to balance the inputs? With the output tied directly to the non-inverting input, the output must be $V_O = 2$ V to balance the inputs.

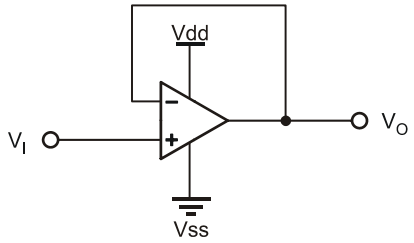
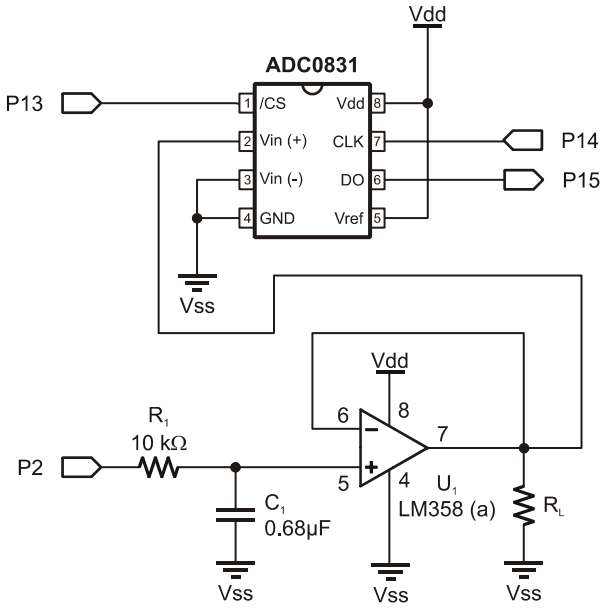


Figure 5-15
Typical Op-Amp Buffer Circuit

How can this be beneficial in our circuit? Since the inputs are of extremely high impedance, the op-amp configuration can be used to buffer the filtered PWM voltage before it is used to drive a device. The op-amp cannot be used to directly drive the fan because it can only supply about 20 mA, but we'll worry about that later. First, let's test an op-amp buffer circuit.

√ Add the op-amp to the circuit as shown in Figure 5-16.
√ Use the 100 kΩ resistor for R_L.
√ Run PwmFiltering.bs2.
√ Run StampPlot macro sic_pc_pwm_filtering.spm.
√ Monitor voltages and discharge rates for varying PWM values.



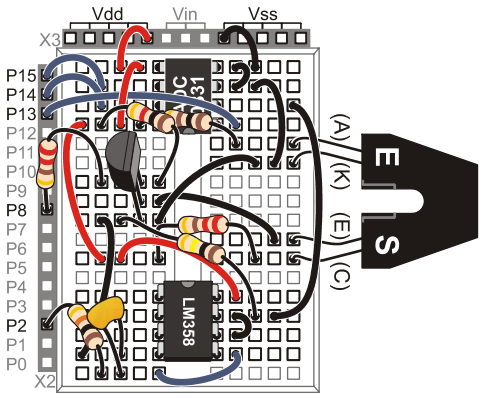


Figure 5-16
Filter with
Op-Amp Buffer

Schematic (top)
and Wiring Diagram
(bottom)

√ Adjust the drive, which should produce stable voltages.

You may note the voltage increasing slightly over time. Very small leakage through the non-inverting input is causing the capacitor to charge further. While not critical, if not refreshed regularly (occurs when StampPlot is not connected), the voltage at the output will rise to maximum over several minutes.

What occurs as the PWM is adjusted to maximum? Do you still get 5 V out? Chances are, you see a maximum voltage around 3.5 V. The output of the op-amp cannot be higher than the voltage supplied to it, and, in fact, will be 1 to 2 volts less than the rail or supply voltages. To overcome this problem, the op-amp can be supplied from a higher voltage – Vin on our boards.

√ Connect pin 8 of the op-amp (NOT P8 on the BASIC Stamp!) to Vin. Ensure this voltage is not connected to any other devices that may be damaged by it.
√ Retest the maximum voltage achieved.

Another name for the combination of filter and op-amp buffer is an active low-pass filter. If we could adjust the input frequency, the maximum frequency (cutoff frequency) allowed to be passed before being dramatically attenuated is $1/(2\pi RC)$.

$$f_{CUTOFF} = 1/(2\pi RC) = 1/(2 \times 3.14 \times 10\ k\Omega \times 0.68\ \mu F) = 23.4\ Hz$$

In our example, attenuation is good because it is being filtered to a DC level leaving no ripple or AC component. The PWM at 112 kHz is well above the cutoff frequency.

A high-pass filter, or one that blocks low frequencies while allowing high frequencies to pass, has the resistor and capacitor positions in the circuit reversed but uses the same formula for cutoff frequency.

ACTIVITY #5: OP-AMP NON-INVERTING AMPLIFIER

The buffer has a gain of 1, or unity gain. For each 1 V change in the input, there is a 1 V change in the output. With an input of $V_I = 2$ V, the output has to drive at $V_O = 2$ V to balance the inverting and non-inverting inputs.

What would happen if the output was not connected directly to the inverting input, but used a voltage divider to cut the output voltage in half before being fed to the inverting input as shown in Figure 5-17?

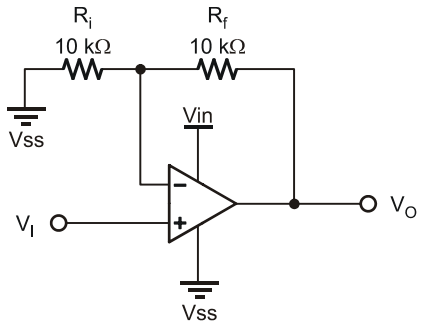


Figure 5-17
Example of Non-Inverting Op-Amp

With the non-inverting terminal set to V_I = 2 V, the output will drive in an attempt to balance the inputs. If the op-amp's output is V_O = 2 V, what voltage will be at the inverting input? The voltage divider of R_f and R_i cuts the voltage in half so that the inverting terminal senses 1 V. The output must go higher. At what voltage will the input sense 2 V? When the op-amp output is 4 V. What would the output be when the non-inverting input is 3 V? The output will be 6 V. For each change of 1 V_I, the output changes by 2 V.

The voltage gain (Av) of the circuit is 2 and is described by the formula:

$A_v = 1 + R_f/R_i$
$A_v = 1 + 10\ k\Omega/10\ k\Omega = 1+1 = 2$
$V_O = (A_v)(V_I) = 2(2\ V) = 4\ V$

Time to test it out!

Additional Parts Required

(2) 10 kΩ Resistors
(1) 4.7 kΩ Resistor
(2) 1 kΩ Resistors

- √ Reconfigure the op-amp as shown in Figure 5-19. Set the 4.7 kΩ resistor aside for the moment.
- √ Add R_{d1} and R_{d2} for the ADC input. These act as a voltage divider to cut the sensed voltage by 2 so that voltages over 5 V do not damage the ADC.
- √ Save PwmFiltering.bs2 under a new name: ModPwmFiltering.bs2.
- √ Change 0.0196 to 0.0392 in the following line of code

```
DEBUG DEC ADC_DataIn ,",",
"[",DEC ADC_DATAIn,",*,0.0392]",CR
```

  This will calculate voltage based on a 10 volt maximum (10 / 255 = 0.0392).
- √ Download your modified program.
- √ Run the StampPlot macro sic_pc_pwm_filtering.spm.
- √ Adjust the Y-Axis to monitor 0 – 10.
- √ Measure the output voltage for various values of PWM.

Now how much does voltage change for each increment of PWM value? With a gain of 2, each increment is 0.0196 V x 2 or 0.0392 V. However, since the total output swing is now 10 V instead of 5 V, the %Resolution is:

0.0392 V / 10 V x 100% = 0.392 %

The %Resolution has not changed, which makes sense since the number of possible steps (255) has not changed.

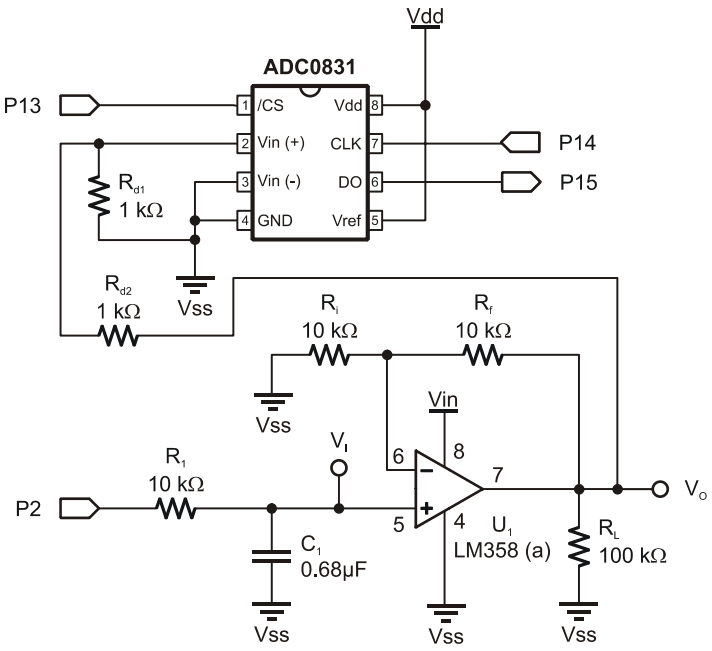


Figure 5-18
Filtered with Non-Inverting Gain

Schematic

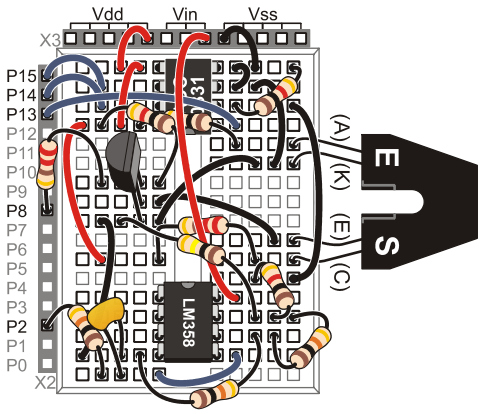


Figure 5-19
Filtered with Non-Inverting Gain

Wiring Diagram

Challenge 5-5: Changing the Gain

√ Replace R_f with a 4.7 kΩ Resistor.
√ Calculate:
  a. The gain.
  b. The maximum output voltage predicted for a PWM of 255.
  c. The output voltage for a PWM value of 25.
  d. The output step resolution.
  e. The %Resolution.
√ Test your results for a, c and d above. In testing the results, leave the **DEBUG** line for plotting with a value of 0.0392 since the ADC is still spanned for 10 V.

ACTIVITY #6: DRIVING THE FAN WITH THE OP-AMP

In previous activities, the PWM duty cycle from the BASIC Stamp was filtered, buffered, and amplified. The circuit can now produce voltages from 0 to 10 V in 0.0392 V increments. Can we now drive the fan with it? No, because the output current of the op-amp is insufficient.

Let's review two key principles:

- The op-amp will drive its output in an attempt to balance the inputs.
- The BJT transistor is a linear device and can conduct up to 200 mA dependent on base current.

Consider how Figure 5-20 brings these principles together. With the input to the non-inverting terminal at 2 V, what voltage will be at to R_L? The op-amp will drive to provide sufficient base current to reach a point where 4 V is sensed at R_L (2 V at the non-inverting input). The common-collector configuration is used to be able to measure the voltage across the load to ground.

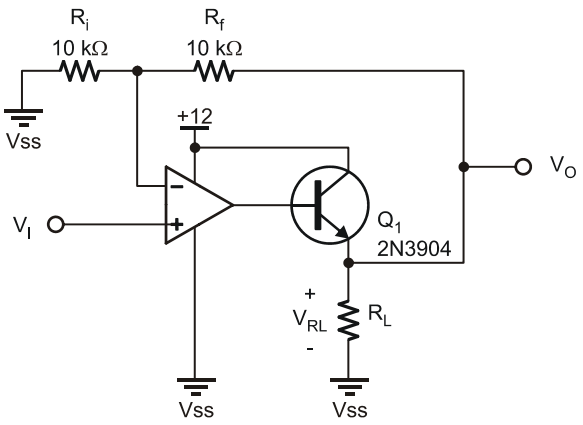


Figure 5-20
Example of High-Current Voltage-Regulated Drive

DO NOT BUILD

With a 12 V supply, and R_L being 100 Ω, the current through R_L will be 4 V/100 Ω = 0.04 A or 40 mA. The transistor is operating in the linear region to supply 40 mA, developing the 4 V required.

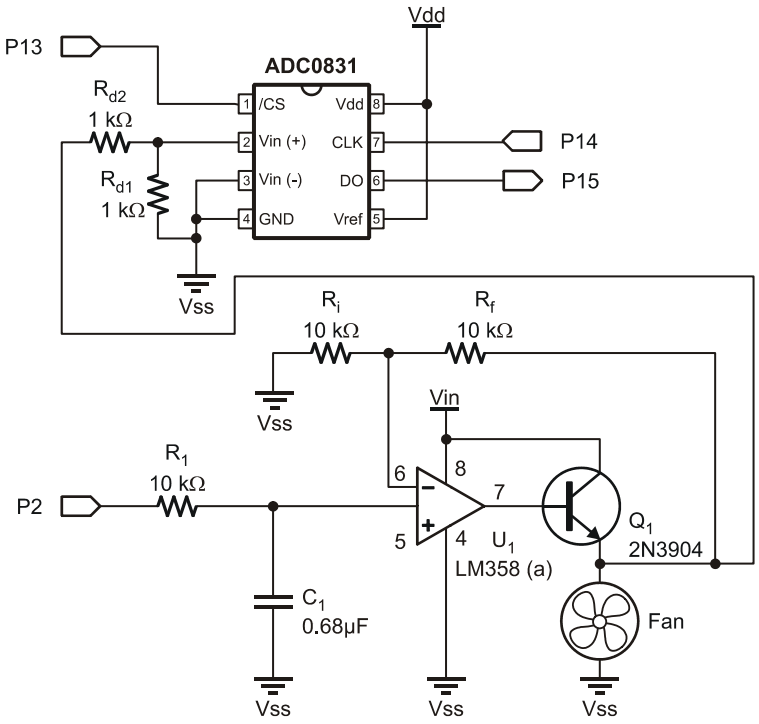
What happens if R_L decreases to 50 Ω? The resistor will require 80 mA to produce the 4 V the op-amp requires to balance the inputs. The output of the op-amp will drive to control the transistor until this condition is met.

The voltage at R_L is being controlled by the input to the op-amp's non-inverting input. With a gain of 2, V_{RL} will be twice the non-inverting voltage. If R_L is replaced with the fan, the DC voltage supply of the fan may be controlled.

Additional Parts Required

(1) 2N3904 Transistor
(1) Brushless DC Fan

- √ Modify your circuit to match Figure 5-21.
- √ Run ModPwmFiltering.bs2.
- √ Run StampPlot macro sic_pc_pwm_filtering.spm.
- √ Use StampPlot to control the voltage to the fan. Note its speed by listening to it. The fan requires at least 3.5 V and may require a small push to start rotation at voltages slightly above 3.5 V.



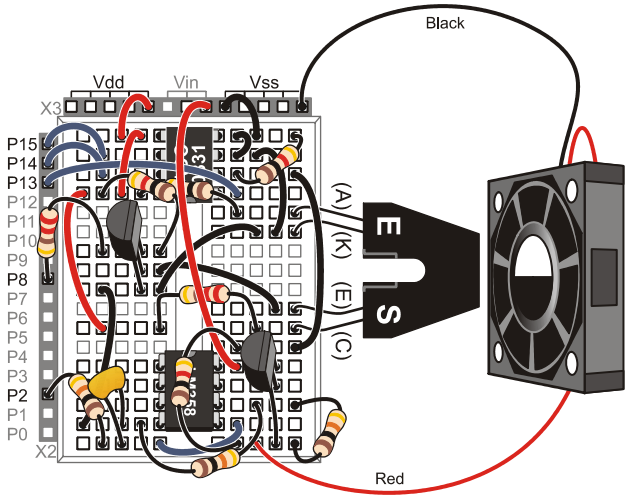


Figure 5-21
Filtered PWM Driving Fan

Schematic (top) and Wiring Diagram (bottom)

Challenge 5-6: Monitoring RPM

√ Save ModPwmFiltering.bs2 under a new name.

√ Modify the program to show the fan's RPM in the StampPlot 'User Status' box above the plot. Example code to update the text:

```
DEBUG "!STAT RPM= ", DEC RPM,CR
```

√ Add all variables, pin assignments, and code to count and scale to RPM. Measure the tachometer count for 1 second. Use pertinent code from Tachometer.bs2 for a starting point.

√ From 5 V to 10 V for fan voltage, calculate the fan's RPM per volt and fill in Table 5-2 . With long sample times, does the speed of the fan vary between PWM updates?

Table 5-2: Voltage vs. RPM

Voltage	RPM	RPM/Volt
5		
6		
7		
8		
9		
10		

ADDITIONAL DEVICES OF INTEREST

While we won't test these circuits and devices, they are worth discussing.

Darlington-Pair Drivers

Just as a Darlington-pair may be used to detect small signals, as explored with the opto-reflective switch, they may also be used as high current drives. The ULN2003A shown in Figure 5-22 is a popular device in that it contains seven drivers, each capable of sinking 500 mA at 50 V.

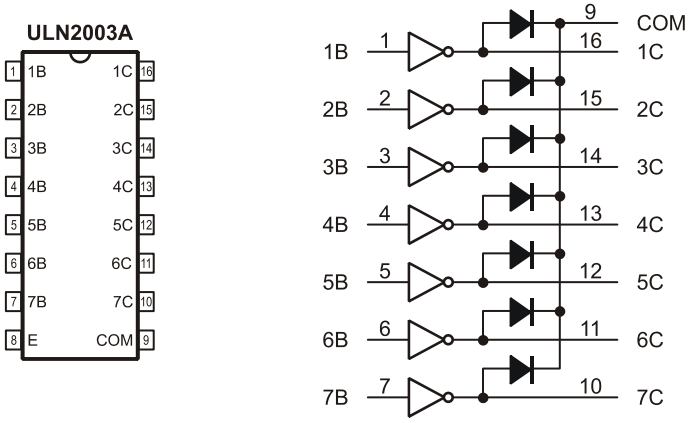


Figure 5-22
Example ULN2003A
High Current Driver

"Current sink" means that the output controls the ground, or negative supply, to power a device, as demonstrated in Figure 5-23. The BASIC Stamp controls the ULN2003, which energizes a 12 V, 100 mA relay coil. The relay coil is closing contacts to drive a 115 VAC fan. The electromagnetic relay draws considerable power. Frequent cycling causes chatter and wears out the moving contacts. An alternative is a solid-state relay discussed shortly.

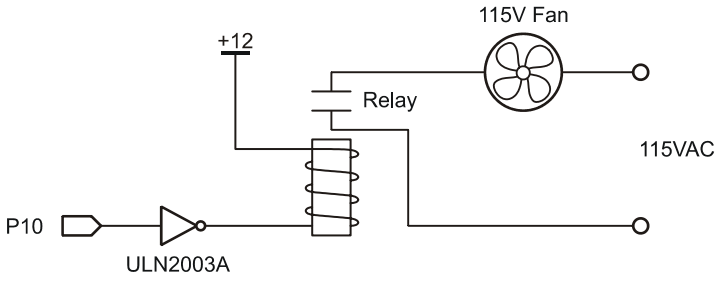


Figure 5-23
Example of Driving an Electromagnetic Relay for 115 VAC Control

Current source is where the output is the positive supply. A helpful phrase to remember is – "Current sinks to ground." By using a Darlington-pair, a very small base current can saturate the drive stage. Figure 5-24 is a schematic for one driver. Note that it contains an internal clamping diode for surges from inductive loads such as coils and motors.

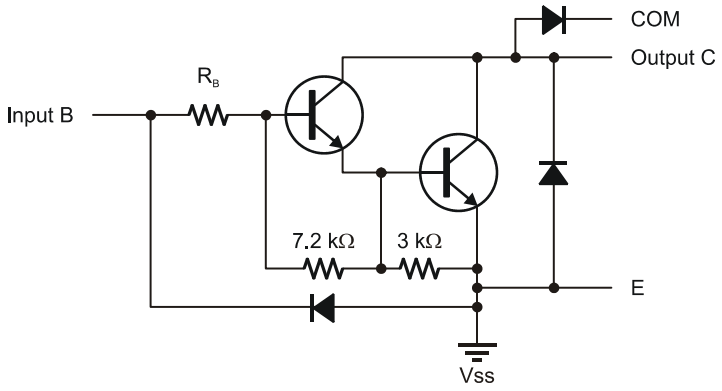


Figure 5-24
Example of ULN2003A Single Stage Darlington-Pair Configuration

H-Bridges

Using PWM control, it was seen that the speed of a DC motor might be controlled. How about direction? This can be performed using the H-Bridge as shown in Figure 5-25.

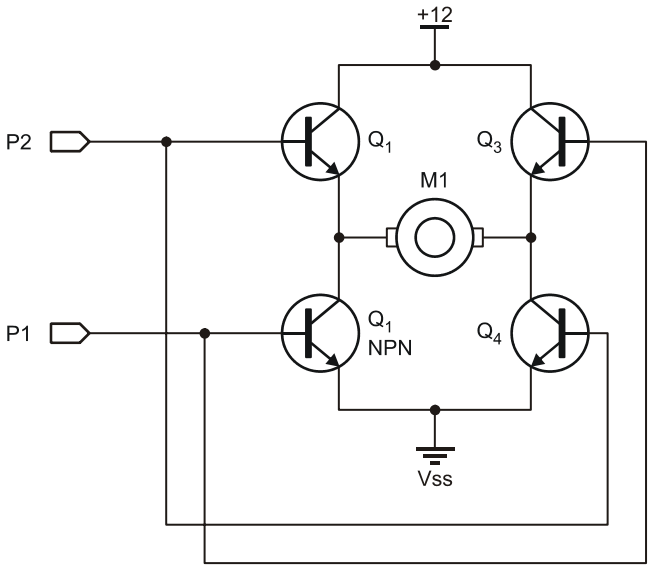


Figure 5-25
Example H-Bridge
Drive of DC Motor

If P2 applies PWM (unfiltered) to the bridge (with P1 low), which transistors will be cycled? Q_4 and Q_1. Following current flow from +12 to ground, the left terminal of the motor will be + and the right −. The motor will rotate in one direction at a speed based on the duty cycle applied.

What occurs when P2 is low and P1 applies the PWM? Q_2 and Q_3 are cycled applying current flow such that the right motor terminal is + and the left -. By reversing polarity to a standard DC motor, the direction of rotation is reversed. While Figure 5-25 uses BJT transistors, a better choice would be MOSFETs since the devices are being switched on and off and are not controlling an analog level. NOTE: The brushless DC fan in the Process Control Parts Kit is not a standard DC motor and will not operate with reversed polarity.

A Little More on Op-Amps

In the op-amp configurations, the signal was applied to the non-inverting input. An input voltage of 2 V produced 4 V at the output with the configuration shown in Figure 5-26. The non-inverting input, and the configuration, is named such because a positive voltage in produces a positive voltage out, or, more specifically, the output rises when then input rises and the same is true for decreasing voltages.

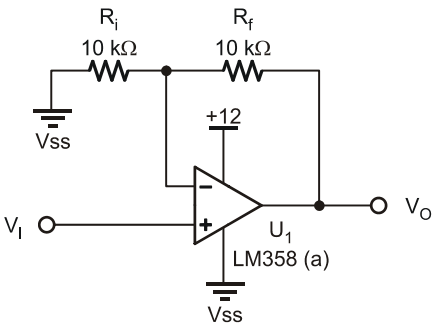


Figure 5-26
Example Non-Inverting Amplifier Configuration

Consider the alternative – the Inverting Amplifier shown in Figure 5-27. With an input of 2 V to the inverting terminal, the output will be −2 V.

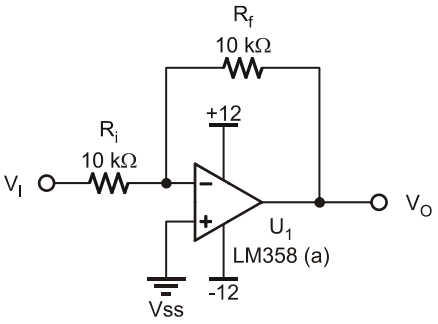


Figure 5-27
Example of Inverting Amplifier configuration

The op-amp still wants to balance the inputs. If the inverting terminal is 2 V higher than the non-inverting (which is 0 V), the output wants to drive the inverting terminal down to balance the inputs, or subtract off those 2 V, so −2 V is needed.

The gain formula for this configuration is:

$$AV = -R_f/R_i$$

or

$$V_O = (-1)(V_I)(R_f/R_i)$$
$$V_O = (-1)(2\text{ V})(10\text{ k}\Omega/10\text{ k}\Omega)$$
$$= -2\text{ V}$$

If the input increases to 3 V, the output will decrease to −3 V. The output is inverted from the input. Note the supply voltage: +12 V and −12 V. Unlike previous circuits, this one requires a negative supply because the output must go negative. Not all op-amps are the same! Many op-amps require a dual supply, or + and − supply voltages, such as the popular LM741. The LM358 was chosen for this text because it requires only a single supply, or +voltage and ground.

One final op-amp configuration to consider is the differential amplifier in Figure 5-28. In this configuration, neither input is ground. The output is dependent on the difference between inputs.

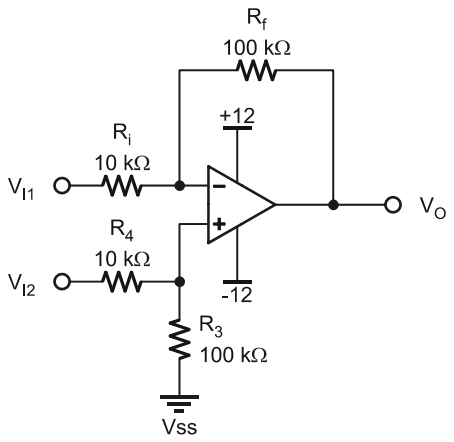


Figure 5-28
Example of Differential Amplifier configuration

DO NOT BUILD

The direction of the output relies on the principle of balancing inputs. If the inverting input is higher than the non-inverting input ($V_{I2}>V_{I1}$), the output will go negative in an attempt to drive down voltage. If $V_{I2} < V_{I1}$, the output will be positive to drive V_{I2} up to V_{I1}. The resistor networks of both sides must be equally sized.

The gain equation is:

$$R_f/R_i$$

or

$$V_O = (V_{I1}-V_{I2})(R_f/R_i) = (3\ V-2\ V)(100\ k\Omega/10\ k\Omega) = (1\ V)(10) = 10\ V$$

A special class of a differential amp with no feedback resistor network is called a comparator and will be explored in Chapter 6.

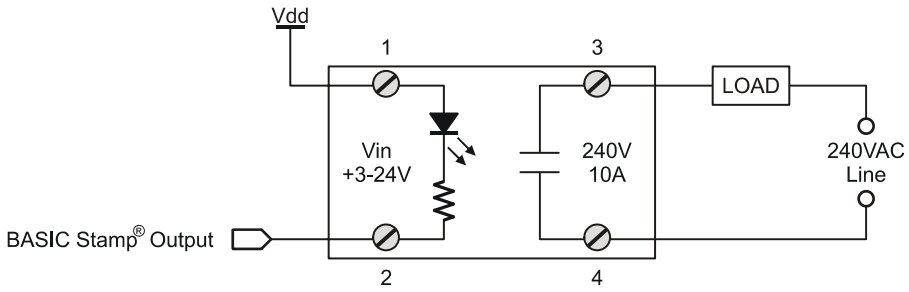
Digital to Analog Converter

In this section a low-pass active filter was used to develop an analog voltage from the PWM of the digital controller. Another popular means of producing the same effect is through the use of a Digital to Analog Converter (DAC). Just as the Analog to Digital converter allows an analog voltage to be converted to binary and then used by the controller, the DAC accepts a digital value from the controller and converts it to an analog voltage.

Solid State Relays

Unlike an electromagnetic relay, which uses an electromagnetic coil to move electrically isolated contacts to energize a large load, the Solid State Relay (SSR) has no moving parts and may be controlled with a very small current. The BASIC Stamp controlling a high voltage load is illustrated in Figure 5-29.

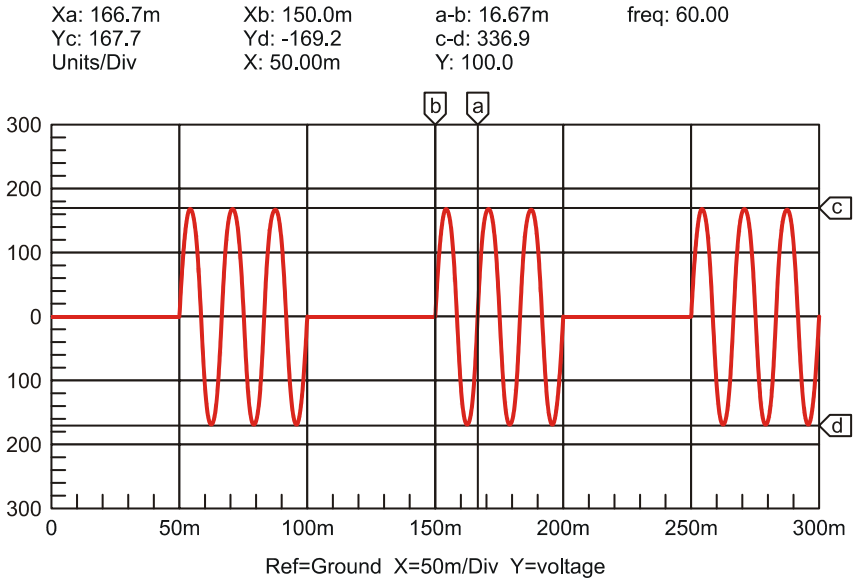
Figure 5-29 Solid State Control of Load



The BASIC Stamp is simply energizing an LED in the SSR. The LED's light energizes a current switching device, a TRIAC, to control 240 VAC to the load. The SSR uses optical coupling and therefore complete electrical isolation between the low voltage BASIC Stamp circuit and high voltage load circuit.

The **`PWM`** command should not be used to regulate the power to the load. The frequency of household electricity is 60 Hz, and the period of a single wave equals 16.67 ms, as shown by the "a" and "b" labels in Figure 5-30. The PWM pulse is around 9 µs (1/112 kHz), which would not provide effective control of the SSR. Using **`HIGH`** and **`LOW`** commands to control the SSR, as was done in the first part of Activity #2, will provide better control. Figure 5-30 illustrates energizing the AC output with a 50% duty cycle of 50 ms on and 50 ms off.

Figure 5-30 PWM of AC with 50% Duty and Period of 100 ms



CONCLUSION

Controlling higher voltage or current output devices is often a necessity when interfacing real-world process control. The transistor (BJT or FET) is a popular and inexpensive means of controlling DC devices.

Using PWM, the power to a motor or device may be varied. The percentage of time HIGH as compared to the total time or period is called duty cycle. Duty cycle values can range from 0% to 100%. The PBASIC command **`PWM`** provides a means to control the duty cycle, but uses values of 0 to 255 to define 0% to 100% duty cycle. A transistor controlled by the PWM maybe used to control a higher current or voltage DC device.

PWM may be directly applied in many instances, but it is in effect only while the instruction is being executed. Through the use of RC filters, the duty cycle may be converted to a voltage from 0 V to 5 V. With the use of operational amplifiers the voltage can be buffered or amplified. This provides a very high-impedance input from the RC filter and a very low impedance output to drive the device or a transistor.

Beyond what is explored through experimentation, there exist a wide range of devices and means of driving devices such as Darlington pairs, coil relays and solid-state relays.

SOLUTIONS TO CHAPTER 5 CHALLENGES

Challenge 5-1 Solution

The higher the HIGH time as compared to LOW time, the faster the fan will run. With HIGH times 3 ms and under the fan may not rotate due to low voltage.

The LED should be noticeably dimmer with long LOW times as compared to HIGH times.

Challenge 5-2 Solution

Part A:
The fan's speed drops off faster below 70% duty because of the increased time allowing the fan to slow more.

Part B:

1. With the simulated delay of 2 seconds, the fan slows noticeably, or almost stops, before the RPM is measured.
2. The simulated delay should occur after measuring and plotting the speed, just prior to `LOOP` for best control and monitoring.

Challenge 5-3 Solution

Part A:

By changing the value of R_L to 100 kΩ, the discharge will be faster due to higher current draw.

$1\ \tau = 1.1(100\ k\Omega)(0.68\ \mu F) = 0.0748$ seconds

$5\ \tau = 5\ (0.0748$ Seconds) = 0.374 seconds = Discharge time

Part B:

1. $5\ \tau = 1.1(100\ k\Omega)(0.68\ \mu F) = 0.0374$ seconds
2. No. It would require:
   $5 \times 1.1(1\ M\Omega)(0.68\ \mu F) = 3.74$ seconds

Challenge 5-5 Solution

With a 4.7 kΩ value of R_f:

a. Gain $= 1+R_f/R_i = 1 + 4.7\ k\Omega/10\ k\Omega = 1.47$

b. Maximum output: At capacitor: 5 V out with 255 PWM Value.

Output of Op-Amp = $V_I(Av) = 5\ V(1.47) = 7.35$ V

c. For PWM value of 25:

$25/255 \times 7.35\ V = 0.72$ V

d. Output step resolution: $= 7.35\ V/255 = 0.0288$ V

e. %Resolution is still 0.392% since the number of steps is still 255.

Challenge 5-6 Solution

√ To the declarations add:

```
Opto_SW       PIN     8
Opto_Count    VAR     Word    ' Count from opto-reflective switch
RPM           VAR     Word    ' Calculated RPM
CyclesPerRev  CON     4       ' Number of pairs on encoder used
```

(Note: Adjust the value of **`CyclesPerRev`** accordingly for the encoder wheel used.)

√ In the main **`DO...LOOP`** add:

```
GOSUB ReadTach
```

√ Add a subroutine:

```
ReadTach:
   COUNT Opto_SW, 1000, Opto_Count    ' Measure counts per unit time
   RPM = Opto_Count*60/CyclesPerRev   ' Calculate RPM
RETURN
```

√ To **`PlotData`** add:

```
DEBUG "!STAT RPM=", DEC RPM, CR
```

Chapter 6: Open Loop Continuous Process Control

Continuous process control involves maintaining desired process conditions. Heating or cooling objects to a certain temperature, rolling a thickness of aluminum foil, or setting a flow rate of material into a vat in to order maintain a constant liquid level are all examples of continuous process control. The condition we want to control is termed the "process variable." Temperature, thickness, 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.

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.

For a simple example of open-loop control, consider 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.

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.

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 a 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.

ACTIVITY #1: TESTING THE LM34

When bringing analog data, such as temperature, into the BASIC Stamp, it is important to understand the transfer functions involved to properly condition the analog voltage and to use software to convert that data into something meaningful. The LM34 temperature sensor supplied with the kits converts a temperature to a voltage with a transfer function of 0.01 V/°F. At 100 °F, the output will be 1 V. At 80 °F, the output will be 0.8 V and so on. The LM34-DZ style has a linear output range of 32 to 212 °F. The BASIC Stamp for monitoring and controlling our system must read this voltage, representing temperature.

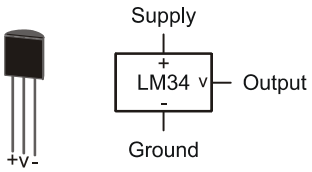


Figure 6-1
The LM34 Temperature Sensor

Parts Required

(1) ADC0831
(1) LM34 Temperature Sensor
(2) 220 Ω Resistors
(1) LED

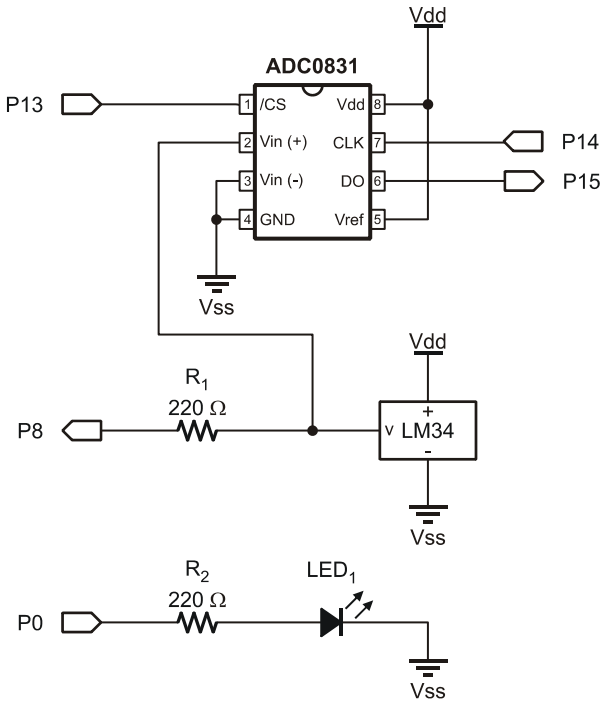


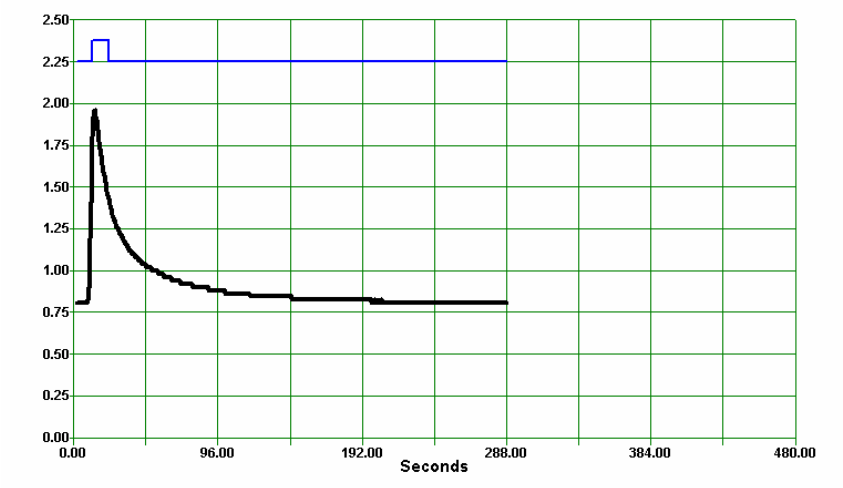
Figure 6-2
Monitoring the LM34 Temperature Sensor

Schematic

- √ Construct the circuit in Figure 6-2.
- √ Run the BASIC Stamp program DataMonitoring.bs2 from Chapter 3, page 46.
- √ Open StampPlot macro sic_pc_data_monitoring.spm (also from Chapter 3).
- √ Adjust the analog (Y-axis) scale to 2.50.
- √ Increase the time (X-axis) scale to 480 seconds.
- √ Connect and plot.
- √ Use a lighter or other heat source to apply heat to the LM34, not to exceed 2.0 V (200 °F).
- √ Monitor temperatures noting the minimum change in voltage.

Figure 6-3 is a plot of the heating and cooling. Have you seen a curve of this shape previously? Recall the discharge plot of the RC network in Chapter 5. This is, or is close to, what is called a First Order Response Curve and is seen in many simple systems that have a transfer of energy. In this case the energy is that of temperature. The mathematical value e, the natural log, is based on this response curve.

Figure 6-3 Plot of Heating and Cooling



The shape of the curve stems from the fact that when there exists a large difference in energy, there is a fast transfer of energy. As the two systems' energy states become closer, the transfer rate decreases exponentially. The voltage across a capacitor and the

temperature difference between the sensor and air are just two examples of energy transfer.

Just as the RC network required 5 time constants (5 τ) to fully discharge, the LM34 requires 5 τ to fully cool. Based on the curve, what is the value of 1 τ for this device? Approximately 200 seconds/5 = 40 seconds.

The LM34 has a very large mass that must be heated and cooled when responding to temperature changes. Other temperature sensors, such as thermistors that change resistance based on temperature, may have very small mass in order to quickly respond to temperature changes.

Challenge 6-1: Testing Another Temperature

Part A:

- √ Heat the sensor to approximately ½ the temperature in Activity #1.
- √ Allow the sensor to return to room temperature.
- √ Determine the value of 5 τ and 1 τ.

Was there a significant change in the time constants using the two temperatures? Why or why not?

Part B:

- √ Being careful not to get the electronics wet, cool the system to 50° F using ice or a cold can.
- √ Monitor the graph of cooling and returning to room temperature.

1. Explain the graph of the warming based on your understanding of energy transfer rates.
2. Determine the value of 5 τ and 1 τ for warming to room temperature.

ACTIVITY #2: SIGNAL CONDITIONING

The analog to digital converter resolves an analog value to a digital value. The ADC0831, as configured, coverts an input voltage of 0 to 5 volts to an 8-bit value (byte) of 0 to 255, respectively, as shown in Figure 6-4. The reason for the value of 255 is that the ADC0831 is an 8-bit ADC. The maximum value of 8 bits is 255. When looking at a binary value, such as 11111111, each position from right to left is a higher power of 2, with the Least Significant Bit (LSB – right most) having a value of 2^0, or 1. The Most Significant Bit (MSB – left most) has a value of 2^7 or 128. A binary value of 11111111 would be the sum of each position with a 1 or:

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

A binary value of 10011001 converted to decimal would be:

$$128 + 0 + 0 + 16 + 8 + 0 + 0 + 1 = 153$$

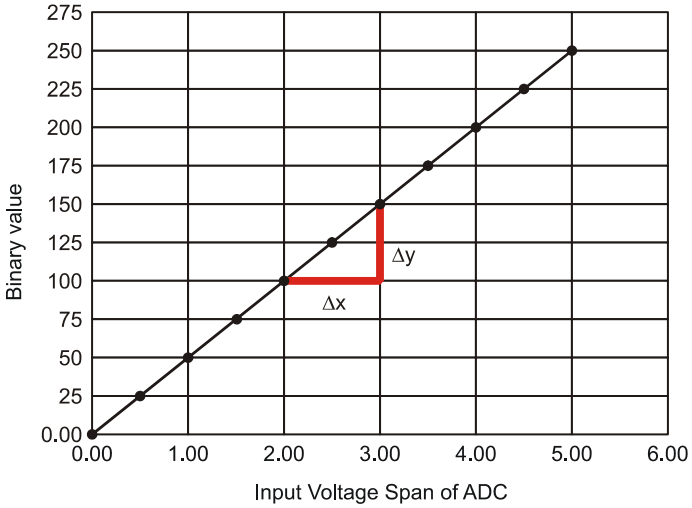


Figure 6-4
Voltage to Binary Conversion Transfer Function

Figure 6-4 shows that the input voltage (0-5 V) is resolved to a binary value (0-255). This linear transfer function can be written as a line equation:

$$y = mx + b$$

Where m is the slope of the line, and b is the y-intercept (where the line crosses the Y-axis), which is 0 in this case.

The slope, m, is the change in y for a given change in x. $m = \Delta y/\Delta x$. In Figure 6-4 the denoted Δy is around 50 for a given Δx of 1 V, for a slope of 50/1 V. While we are only approximating the changes from the graph, any two points on the linear line will yield the same slope. It's best to choose two known values. We know that a voltage of 0 V in will result in a binary value of 0, and a voltage of 5 V in will result in a binary value of 255. Given these two points:

$$m = \Delta y/\Delta x = (255-0)/(5\text{ V}-0\text{ V}) = 51/\text{V}$$

This provides a general equation of:

$$y = (51/\text{V})x$$

Testing the input value of 2 V:

$$y = (51/\text{V})2\text{V} + 0 = 102$$

Another way of stating the general line equation is as a transfer function for our specific system:

$$\text{Binary Value} = (\Delta\text{Output}/\Delta\text{Input})\text{Input}$$
$$\text{Binary Value} = (\Delta\text{Binary Value}/\Delta\text{Vin})\text{Vin}$$
$$\text{Binary Value} = (51/\text{V})\text{Vin}$$

Δy is ΔOutput since that is how much the ADC binary value will change for a given Δx, or change in input to the ADC.

Since there are only 255 possible steps between 00000000 and 11111111, the ADC is limited to how finely it can resolve a voltage. What would be the bit value for voltages of 1.05 V and 1.06 V? 53.55 and 54.06. But since only integer values can be used, they would both have byte counts of 54, rounding to the nearest integer. In terms of temperature, this change between 105 °F and 106 °F would not be measurable.

When the BASIC Stamp processes the binary value, the process is reversed where the byte value is converted into a temperature using code. In performing the conversion, a

line equation can again be used where the change in output temperature will be 0-500 °F (based on the transfer function of 0.01 V/°F) for the change in input value of 0 to 255.

Temp = m(Bit Value) + b where b = 0
m = ΔOutput/ΔInput = ΔTemperature Span/ΔByte Value = 500 °F/255 = 1.96 °F
Temp = 1.96 °F x Byte Value

Each change of 1 in byte value will signify a temperature change of 1.96. Since the ADC is limited on resolution, the output will have distinct steps as you have probably noted in many experiments.

Example Program: AdcSpanOffset.bs2

√ Using the same circuit as Activity #1, enter, save and run AdcSpanOffset.bs2.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - AdcSpanOffset.bs2
' Tests the spanning and offset input range of the ADC 0831 using PWM
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------

ADC_ByteValue  VAR   Byte       ' Analog to Digital Converter data
V_Offset       VAR   Byte       ' Offset voltage read from StampPlot
V_Span         VAR   Byte       ' Span voltage read from StampPlot
TempF          VAR   Word       ' Calculated temp in hundredths of degree F

ADC_CS      PIN 13              ' ADC Chip Select pin
ADC_Clk     PIN 14              ' ADC Clock pin
ADC_Dout    PIN 15              ' ADC Data output

ADC_VRef    PIN 10              ' Pin for PWM to set ADC voltage span
ADC_Vminus  PIN 11              ' Pin for PWM to set ADC Offset

' -----[ Initialization ]---------------------------------------------------
PAUSE 1000                      ' Allow connection stabilization

' -----[ Main Routine ]-----------------------------------------------------
DO
  GOSUB ReadSP
  GOSUB SetADC
  GOSUB ReadADC
  GOSUB CalcTemp
  GOSUB UpdateSP
```

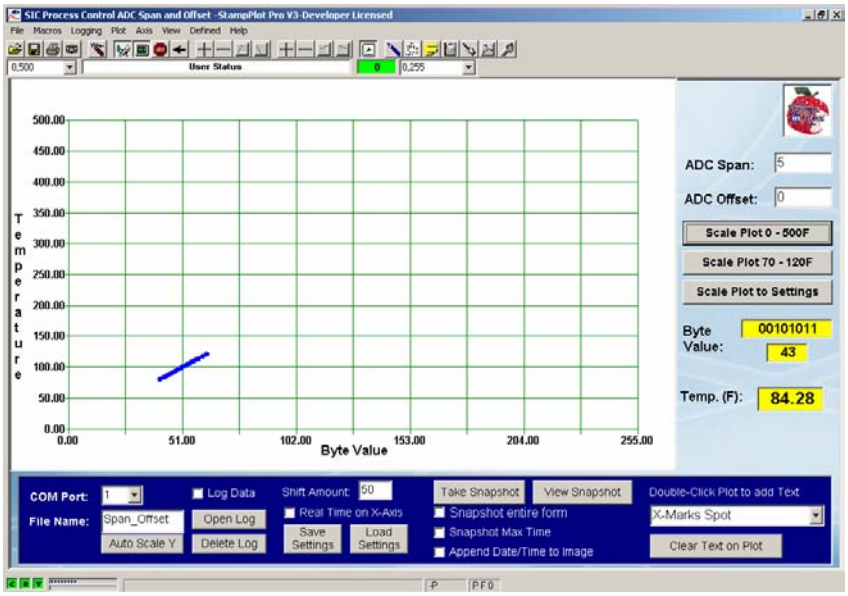
```
  GOSUB PlotPoint
  PAUSE 100
LOOP

' -----[ Subroutines ]------------------------------------------------------
ReadSP:
  DEBUG CR,"!READ [(txtADCoffset),*,10]",CR ' obtain offset volt. in tenths
  DEBUGIN DEC V_Offset
  PAUSE 50
  DEBUG "!READ [(txtADCspan),*,10]",CR      ' obtain span voltage in tenths
  DEBUGIN DEC V_Span
  PAUSE 50
  RETURN

SetADC:
  PWM ADC_Vminus, V_Offset * 255/50,100     ' Apply PWM to set offset volt.
  PWM ADC_Vref, V_Span * 255/50,100         ' Apply PWM to set span voltage
  RETURN

ReadADC:                ' Read ADC 0831
  LOW ADC_CS            ' Enable chip
  SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_ByteValue\9] ' Clock in ADC data
  HIGH ADC_CS           ' Disable ADC
  RETURN

CalcTemp:   ' y = mx + b
            ' where y=temp,
            ' m = (change in output)/(change in input) = voltage Span/255
            ' x = ADC value read, b = offset, y = temperature in hundredths
            ' temperature = (Span/255)Byte + Offset
  TempF = (V_Span * 1000)/255 * ADC_ByteValue + (V_Offset * 1000)
  RETURN

UpdateSP:
  DEBUG "!O txtByteBin = ", BIN8 ADC_ByteValue,CR, ' Update w/ binary ADC val
        "!O txtByte = ", DEC ADC_ByteValue,CR      ' Update w/ decimal ADC val
  DEBUG "!O txtTemp = [", DEC TempF,",/,100]",CR   ' Update w/ temperature/100
  RETURN

PlotPoint:
  DEBUG "!FCIR (txtByte),(txtTemp),0.3A,(WHITE)",CR  ' Plot white circle at
  PAUSE 100                                          '  byte, Temp as X,Y
  DEBUG "!FCIR ,,,(BLUE)",CR                         ' Plot again in blue
  RETURN
```

- √ Load StampPlot macro sic_pc_adc_span_offset.spm.
- √ Verify that "ADC Span" is set to 5 and "ADC Offset" is set to 0 in StampPlot.
- √ Connect and plot. A small blinking dot will appear on the screen, marking the temperature.

- √ Read down from the dot to determine the byte count read from the ADC. The display is not a ".vs time" plot, so if the temperature doesn't change much, you will only see a few dots.
- √ Pinch the LM34, or briefly apply a heat source, to change the temperature reading.

Figure 6-5 is a plot of the byte count read from the ADC and the calculated temperature. Over a range of 0 °F to 500 °F, there appears to be very good resolution based on how close the individual points are.

Figure 6-5 ADC Byte Count vs. Calculated Temperature - Full Range

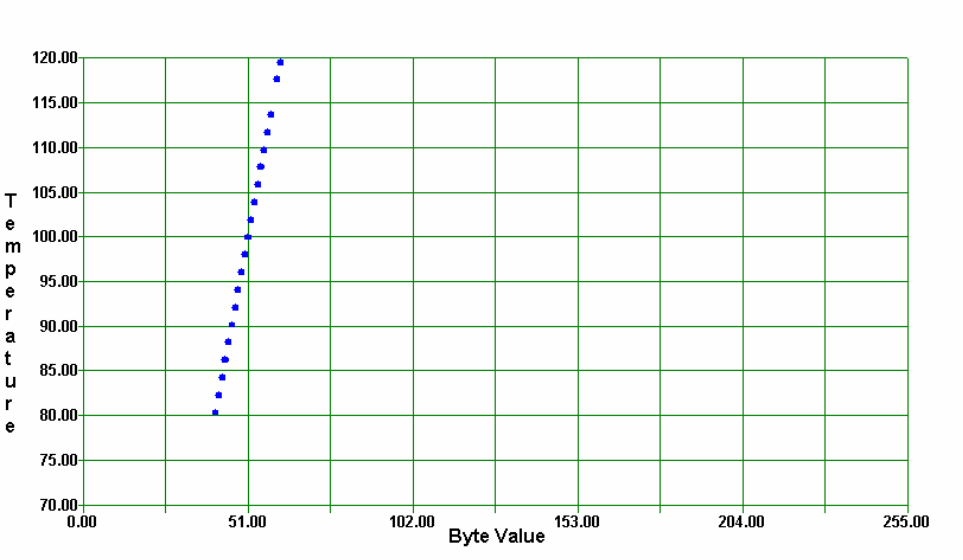


If we were monitoring and controlling temperature over 0 °F to 500 °F, this may be adequate resolution. But our temperatures of focus will be in the 70 °F to 120 °F range. How finely can the system monitor and control over this range?

- √ Click the "Scale Plot 70−120F" button on the interface.

Figure 6-6 is a capture of the plotted data for this range. Doesn't look so good now, does it? With the current resolution and span, the system can only resolve temperatures to 1.96 °F. This isn't very good for precise temperature monitoring and control.

Figure 6-6 ADC Byte Count vs. Calculated Temperature - Narrow Range



For better resolution the system needs to be modified to focus on the temperatures of interest. One means to do this would be amplify the voltage before converting to a digital value.

Consider (but do not build) the schematic in Figure 6-7. What effect would it have on the resolution?

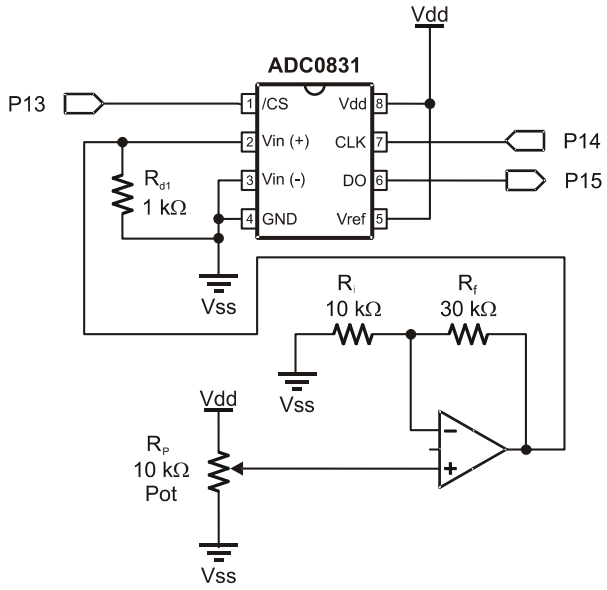


Figure 6-7
Voltage Monitoring With Gain

DO NOT BUILD

Recall that the op-amp is configured in a non-inverting configuration where the gain is:

$$R_f/R_i + 1$$

For the given values, the gain is:

$$30\ k\Omega/10\ k\Omega + 1 = 3+1 = 4$$

For the potentiometer range of 0 to 5 volts, this would provide an output of 0 to 20 V (providing that the op-amp was powered from a voltage above 20 V). The ADC is still converting the voltage range of 0 to 5 V to a byte count of 0 to 255. Consider what occurs to the slope of the graph in Figure 6-8 of sensor voltage to voltage ADC and byte value when gain is applied.

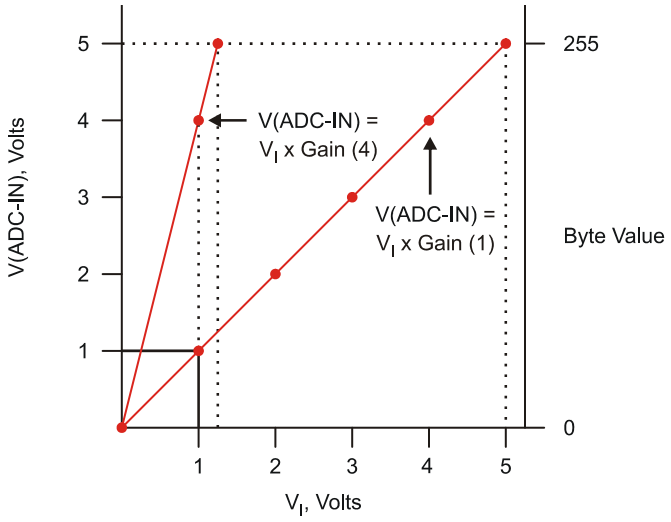


Figure 6-8
Vin vs. ADC Voltages

With no gain, or actually a gain of 1 or unity gain, the voltage to the ADC is the voltage sensed at the ADC. When the op-amp is used with a gain of 4, an input voltage of 1.25 V will provide 5 V to the ADC. Note the slope of the line:

$$m = \Delta Output/\Delta Input = (5-0)/(1.25-0) = 5/1.25 = 4$$

The slope of the line is also the gain, giving a line equation of:

$$VADC = 4(ADC\text{-}IN) + 0$$

What is the resolution of the circuit now? In comparing the applied voltage to the bit count: 1.25 V/255 = 0.0049 V/bit value. In terms of temperature:

$$125\ °F/255 = .49\ °F/bit\ value$$

...which is 4 times the resolution previously measured.

In terms of byte value, the resolution is 255/1.25 V or 204/V. Our high temperature of interest, 120 °F (1.2 V), would provide a byte value of 1.2 V x 204 = 244.8 or 245. The low temperature of interest, 70 °F (0.7 V), would provide a byte value of 0.7 x 204 = 142.8 or 143. So the temperature range of interest covers a byte value change of 245−143 = 102. The majority of our available byte values (0 to 101 and 246 to 255) are still outside the temperature range of interest.

What line equation would best serve the needs of the monitoring and control system? Figure 6-9 is the ideal line needed to convert the temperature range of interest to a digital value for measurement. The temperature range of 70 °F (0.7 V) to 120 °F (1.2 V) is amplified and covers 0 to 5 volts into the ADC for byte values of 0 to 255.

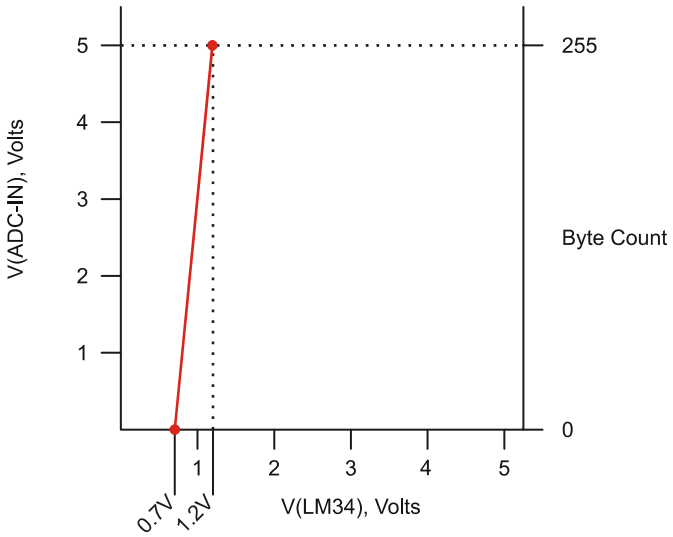


Figure 6-9
Ideal Conversion for
70 °F-120 °F

Let's analyze the line and develop the line equation.

$$y = mx + b$$
$$m = \Delta Output/\Delta Input = (5-0\ V)/(1.2-0.7\ V) = 5\ V/0.5\ V = 10$$

Since the line does not cross at (0,0), b can be calculated given any single known point, such as 0.7 V and 0 V or 1.2 V and 5 V.

$5\ V = 10(1.2\ V) + b$
$5\ V - 12\ V = b$
$b = -7\ V$
$y = 10(x) - 7\ V$ or $VADC = 10(VIN) - 7\ V$

From this equation we can see that a gain of 10 is required and 7 must be subtracted from the product. There are op-amp configurations that can add or subtract voltages, called summing amplifiers. Figure 6-10 illustrates an op-amp circuit which performs the equation of $V_O = 10(V_I) - 7\ V$.

This circuit utilizes a two-stage op-amp configuration to perform the equation or transfer function. Stage 1 provides the gain using an inverting amplifier. The Stage 2 amplifier is configured as a summing amplifier.

Stage 1:

$$V_{S1} = (-R_{f1}/R_{i1})V_I$$
$$= (-100\ k\Omega/10\ k\Omega)V_I$$
$$V_{S1} = -10\ V_I$$

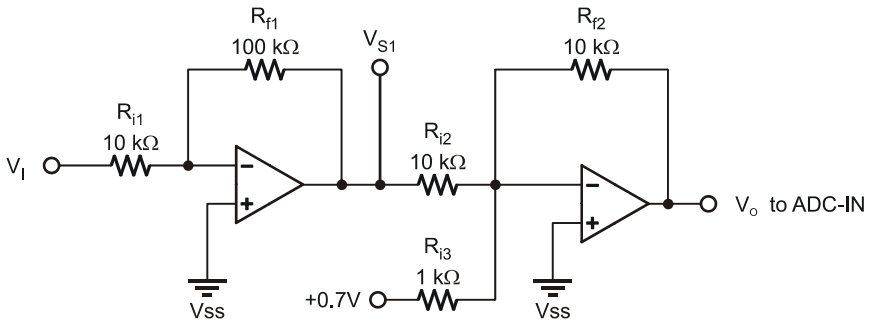
Stage 2:

$$V_O = V_{S1}(-R_{f2}/R_{i2}) + V_{offset}\ (-R_{f2}/R_{i3})$$
$$= V_{S1}\ (-10\ k\Omega/10\ k\Omega) + 0.7\ (-10\ k\Omega/1\ k\Omega)$$
$$= V_{S1}\ (-1) + 0.7\ (-10)$$
$$V_O = -V_{S1} - 7$$

Combining the results of the two stages provides the final voltage into the ADC.

$$V_O = -V_{S1} - 7$$
$$= -(-10\ V_I) - 7$$
$$V_O = 10\ V_I - 7$$

Figure 6-10 Amplify and Offset Signal with Op-Amps (*DO NOT BUILD*)



Testing it out with a voltage of 1.2 V at V_I representing 120 °F:

Stage 1: $-10(1.2\text{ V}) = -12\text{ V}$

Stage 2: $-(-12\text{ V}) - 7\text{ V} = 5\text{ V}$

Of course, accomplishing this requires supply voltages in excess of +/− 12 V for the op-amps. Another option is to offset the input prior to amplifying the voltage.

$$V_O = 10(V_I - 0.7)$$

Consider the differential amplifier in Figure 6-11. Recall that this configuration amplifies the difference between the two input voltages.

The formula for this configuration is:

$$V_O = (V_{I1} - V_{I2})(R_f/R_i)$$

where V_{I1} is the voltage from the LM34 and V_{I2} is the offset voltage of 0.7 V. For the voltage of 1.2V from the sensor, relating to a temperature of 120 °F:

$$V_{ADC} = (1.2\text{ V} - 0.7\text{ V})(100\text{ k}\Omega/10\text{ k}\Omega) = (0.5\text{ V})(10) = 5\text{ V}$$

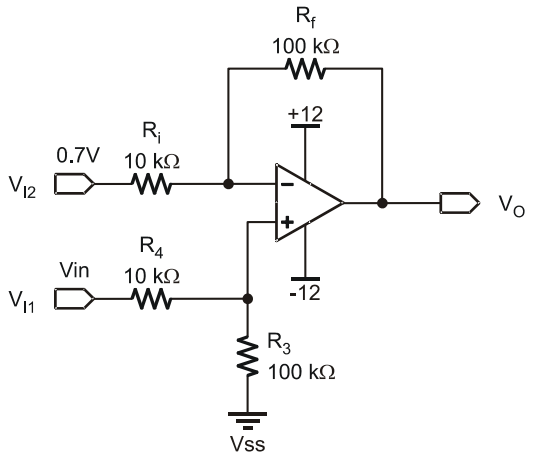


Figure 6-11
Offset and Gain using Differential Configuration

DO NOT BUILD

6

How would the 0.7 V be set? One way would be to use a potentiometer that is adjusted to 0.7 V as the input for V_{I2}. In both configurations we are setting the offset and span for the voltage range of interest.

Fortunately, the ADC0831 converter can directly perform the spanning and offsetting. Consider the pin-out of this device as shown in Figure 6-12.

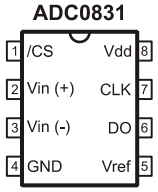


Figure 6-12
ADC0831 Pin Out

Up until now Vin(-) has been connected to Vss (0 V) and V_{ref} has been connected to Vdd (5 V). These two inputs set the offset and span voltages over which to convert. If the ADC0831's Vin(-) is set to 0.7 V and Vref is set to 0.5 V, the ADC will convert the range of 0.7 V to 1.2 V to a byte value of 0 to 255 respectively.

One method of using potentiometers to set these two pins is shown in Figure 6-13.

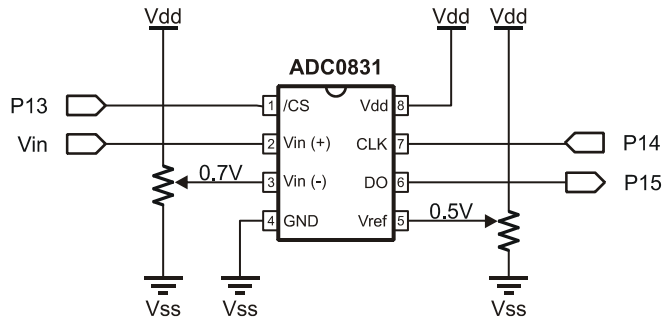


Figure 6-13
Span and Offset of ADC0831 Using Potentiometers

DO NOT BUILD

Instead of using potentiometers, which must be manually adjusted for different ranges, can you think of a simple way to have the BASIC Stamp set the offset and span voltage of the ADC directly?

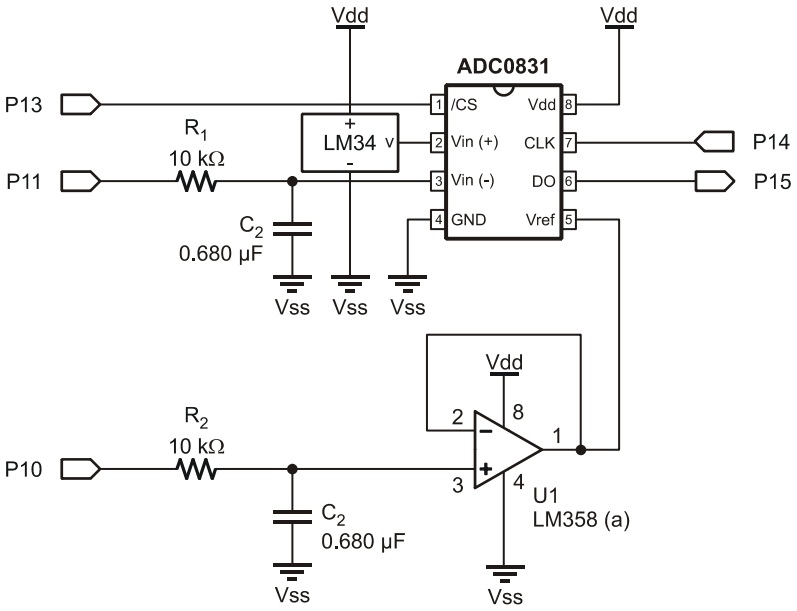
Recall how filtered PWM was used in Chapter 5 to set the voltage of the fan. The same principle may be applied here as shown in Figure 6-14. Due to the low impedance of the Vref input, the filtered PWM must be buffered prior to being applied. The ADC0831's Vin(-) is higher impedance and does not require buffering.

Additional Parts Required:

(1) LM358 Op-Amp
(2) 0.68 µF Capacitors
(2) 10 kΩ Resistors

- √ Construct the circuit in Figure 6-14.
- √ In StampPlot, set "ADC Span" to 0.5 and "ADC Offset" to 0.7.
- √ Connect and plot.
- √ Heat the LM34 again.
- √ Click the "Scale Plot 70-120F" button on StampPlot to cover the selected range.

Figure 6-14 Span and Offset of ADC Using PWM - *Schematic*



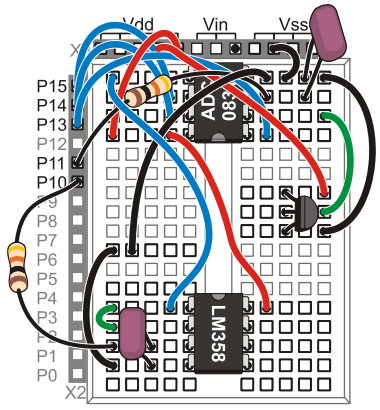


Figure 6-15
Span and Offset of ADC Using PWM

Wiring Diagram

6

Note the new resolution as shown in Figure 6-16 compared to previous tests shown in Figure 6-4. Gaps between points are temperatures that were not sampled. Note that temperature resolves to .196 °F/bit by observing the change in voltage.

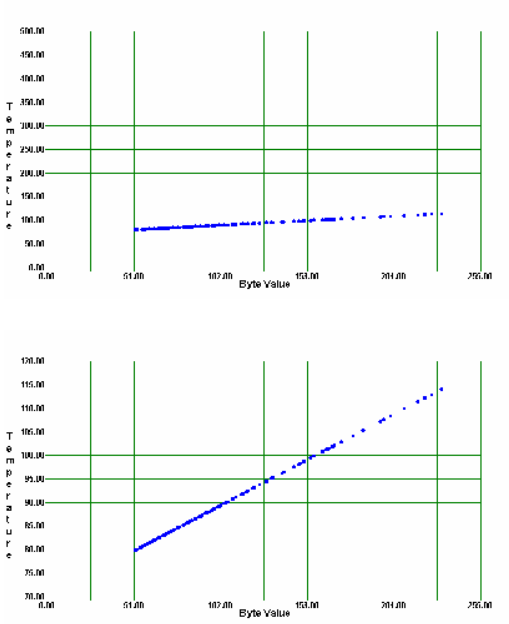


Figure 6-16
ADC Measurement
0 to 500 °F and
70 to 120 °F

The PWM style of setting offset and span may not be 100% accurate but is adequate for our testing. Offset values below 0.5 V should be avoided for better accuracy. Combined offset and span voltages used to set the upper voltage limit (Vspan + Voffset) should be 3.5 V or less since the op-amp is only powered from 5 V.

Program Discussion

The **`ReadSP`** subroutine reads the values of offset and span from StampPlot. These values are used in calculations and in setting the ADC. Since the BASIC Stamp does not use floating-point math (values with decimal points), the values are multiplied by 10 so that 0.5 is stored as 5, or tenths of a volt.

```
DEBUG CR,"!READ [(txtADCoffset),*,10]",CR
DEBUGIN DEC V_Offset
```

The **SetADC** subroutine uses the span and offset values to control the Vref and Vin(-) pins respectively. The settings are scaled accordingly to apply 0-5 V to these pins using PWM. Note that it is again using our transfer function line equation, y = mx + b where b= 0, though slightly rearranged.

```
PWM ADC_Vminus, V_Offset * 255/50,100
```

If 255 were divided by 50 first, the slope would have been 5 and not 5.1. By multiplying first, while ensuring 65535 is not exceeded, we have better resolution and accuracy.

PWM value = (voltage desired in 10ths) x (byte value span)/(max. voltage in 10ths)
PWM value = (voltage in tenths) x 255/50

The **ReadADC** subroutine reads the ADC and stores the byte collected in **ADC_ByteValue**. After enabling the IC using the Chip Select (CS) pin, the data from the ADC is shifted in using a clock pulse on the clock (CLK) line and data is collected from the Data Out (DO) pin. The **\9** means that 9 bits are collected, though the first is not used and is discarded.

```
LOW ADC_CS
SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_ByteValue\9]
HIGH ADC_CS
```

In the **CalcTemp** section, the values of **V_Span** and **V_Offset** are multiplied by 1000. Working with the original ADC values, a span of 0.5 would have related to 50 °F. If we used 50 in our equation for 50/255 this would equal 0.196, but since the BASIC Stamp does not perform floating-point math, this would be 0! By multiplying by 1000, the result is 196 providing a reading of 8007 for a temperature of 80.07. Again, a precaution is that 65535 is not exceeded for any intermediary calculation, such as with a span of 5.0 V.

```
TempF = (V_Span * 1000)/255 * ADC_ByteValue + (V_Offset * 1000)
```

The **UpdateSP** subroutine updates the **txtByte** and **txtByteBin** text boxes with the byte value in decimal and binary respectively, as read from the ADC0831, and also updates **txtTemp** with the current temperature divided by 100.

```
DEBUG "!O txtByteBin = ", BIN8 ADC_ByteValue,CR,
      "!O txtByte = ", DEC ADC_ByteValue,CR

DEBUG "!O txtTemp = [", DEC TempF,",/,100]",CR
```

In the **PlotPoint** subroutine, StampPlot's Filled Circle (**!FCIR**) instruction is used to plot a point in white based on the X coordinate of **ADC_ByteValue** and the Y coordinate of the temperature with a size of 0.3 absolute. After a 100 ms pause, the point is plotted using the same parameters in blue to give the effect of a blinking point.

```
DEBUG "!FCIR (txtByte),(txtTemp),0.3A,(WHITE)",CR
PAUSE 100
DEBUG "!FCIR ,,,(BLUE)",CR
```

Challenge 6-2: Spanning and Scaling for an Air Conditioner System

Instead of eventually controlling an incubator, assume the system under control is an air conditioning system for an equipment room. The temperature needs to be monitored and controlled over a range of 50 °F to 90 °F.

1. What values of **V_Span** and **V_Offset** would be appropriate?
2. What would be the resolution in degrees Fahrenheit for this range of temperature?
3. Draw a graph of byte value vs. temperature.
4. What would be the equation for this line?
5. At 72 °F, what would be:
    a. The output of the LM34?
    b. The byte value of the ADC0831?

ACTIVITY #3: MANUAL CONTROL OF INCUBATOR

Have you ever ridden in a vehicle with someone who controls the cabin temperature by cycling the heater or air conditioner on and off? Too hot – turn on the AC. Too cold – turn off the AC. Every several miles they switch again to keep the cabin comfortable. Manual control of a system with a small allowable range can be time consuming (and a little frustrating to other passengers).

In this activity you will construct the incubator circuit used in the remainder of this text. We will begin by manually controlling the incubator and testing the response. Our incubator is a closed system containing a resistor acting as a heater and the LM34 for temperature monitoring.

The resistor is 100 Ω ½ watt. The "power rating", ½ watt, means how much heat it can dissipate before becoming hot enough to damage it with long-term continuous use. With a 9 V supply, the resistor will be dissipating more heat than it is rated for:

$$P = V^2/R = (9V)^2/100\ \Omega = 0.81\ W.$$

This is almost twice the rated power of the device. As such, the resistor will get very hot and may discolor with prolonged use.

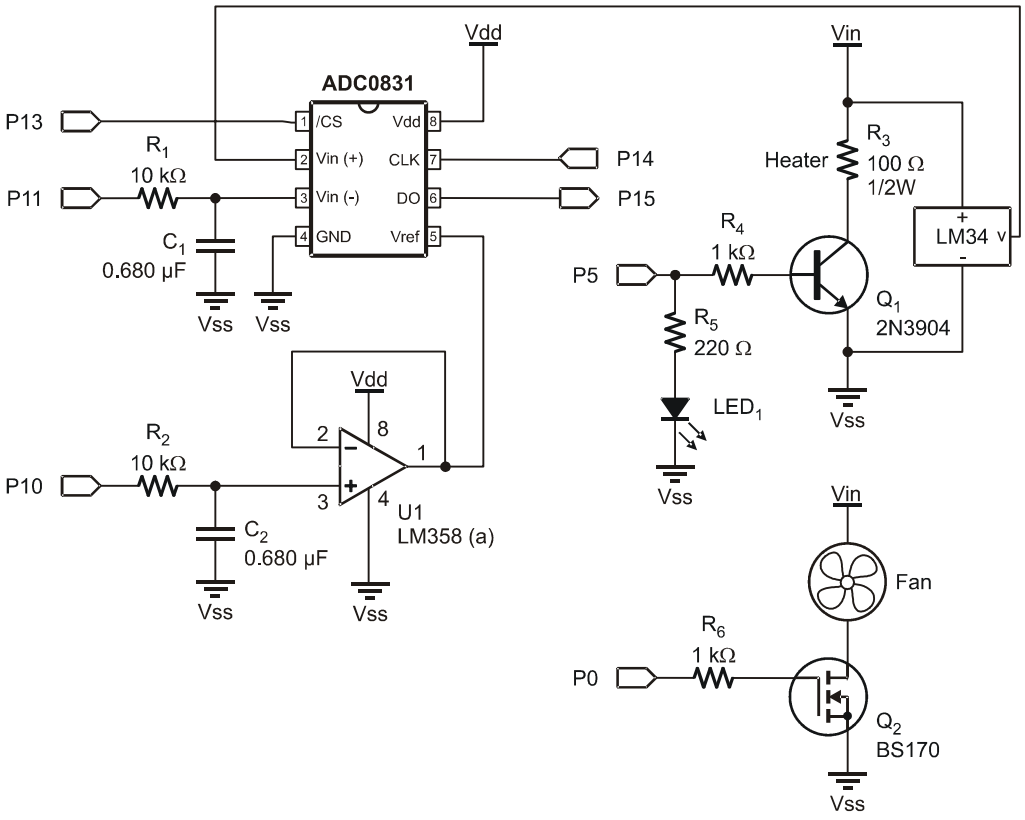
WARNING: Do not use an unregulated 9 V wall-mount power supply for this activity. Use a fresh 9 V battery (this activity will drain the battery). The resistor will reach temperatures high enough to melt soft plastic. Use care when constructing the incubator circuit. Do not allow the test tube to touch the resistor directly; even so, it may warp. NEVER LEAVE POWERED CIRCUITS UNATTENDED!

Additional Parts Required

(1) 2N3904 Transistor - BJT
(1) BS170 Transistor – FET
(1) 220 Ω Resistor
(2) 1 kΩ Resistors
(1) LED
(1) 100 Ω ½ W Resistor (Heater)
(1) 12 V Fan
(1) Polystyrene Test Tube
(1) 9 V battery (not included)

- √ If you are using a wall-mount power supply, disconnect it from your board
- √ Construct the circuit in Figure 6-17. The heating resistor (100 Ω ½ watt, R3) needs to have one lead tightly bent in a U-turn so it can stand vertically up from the board.
- √ Place the plastic test tube on the breadboard directly over R_3 and the LM34.
- √ Place the fan approximately 3 inches from the incubator so the fan will blow towards the incubator, as shown in Figure 6-19. (The fan is facing the opposite direction than it was in previous chapters.)
- √ Connect the 9 V battery to your board.

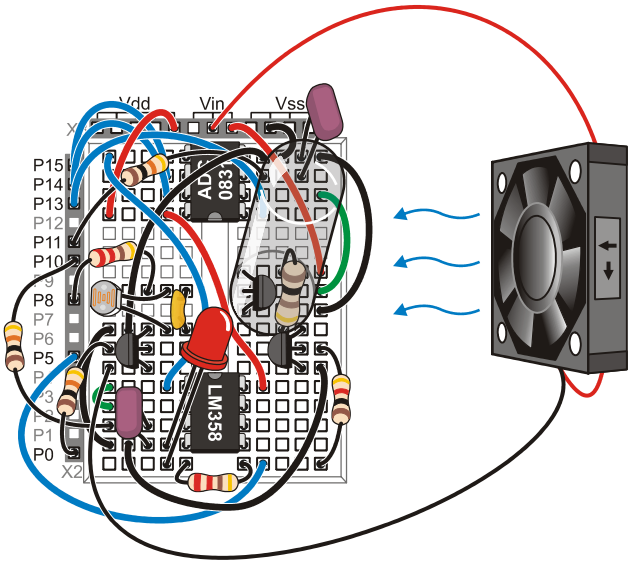
Figure 6-17 Incubator Control Circuit – Schematic



REPEAT WARNING: Do not use an unregulated 9 V wall-mount power supply for this activity. Use a fresh 9 V battery (this activity will drain the battery). The resistor will reach temperatures high enough to melt soft plastic. Use care when constructing the incubator circuit. Do not allow the test tube to touch the resistor directly; even so, it may warp. NEVER LEAVE POWERED CIRCUITS UNATTENDED!

Figure 6-18 Incubator Control Circuit – Wiring Diagram

NOTE: Use a 9 V battery as the power supply for this activity.



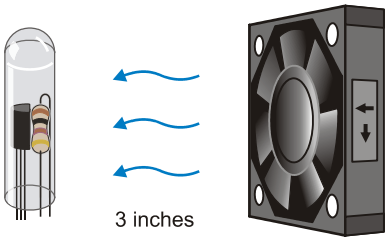


Figure 6-19
Incubator and Fan Positioning

Example Program: IncubatorManual.bs2

√ Enter, save and run the BASIC Stamp program IncubatorManual.bs2.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - IncubatorManual.bs2
' Allows manual control of incubator heater and fan via StampPlot
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
ADC_ByteValue  VAR   Byte       ' Analog to Digital Converter data
V_Offset       VAR   Byte       ' Offset voltage read from StampPlot
V_Span         VAR   Byte       ' Span voltage read from StampPlot
TempF          VAR   Word       ' Calculated temp. in hundredths of degree F

ADC_CS      PIN 13              ' ADC Chip Select pin
ADC_Clk     PIN 14              ' ADC Clock pin
ADC_Dout    PIN 15              ' ADC Data output

ADC_VRef    PIN 10              ' Pin for PWM to set ADC voltage span
ADC_Vminus  PIN 11              ' Pin for PWM to set ADC Offset
Heater      PIN 5               ' Pin for heater control
Fan         PIN 0               ' Pin for Fan control

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

LOW Heater                      ' Heater off
LOW Fan                         ' Fan off
PAUSE 1000                      ' Connection stabilization
GOSUB ReadSP_Temps              ' Get temp span and offset from StampPlot

' -----[ Main Routine ]----------------------------------------------------
DO
  GOSUB ReadSP_Controls
  GOSUB SetADC
  GOSUB ReadData
  GOSUB CalcTemp
  GOSUB PlotTemp
  PAUSE 500
LOOP

' -----[ Subroutines ]------------------------------------------------------
ReadSP_Temps:
  DEBUG CR,"!READ (txtTMin)",CR            ' obtain min temperature (offset)
  DEBUGIN DEC V_Offset
  PAUSE 50
  DEBUG "!READ [(txtTMax),-,(txtTMin)]",CR ' obtain temperature span
  DEBUGIN DEC V_Span
  PAUSE 50
RETURN
```

```
ReadSP_Controls:
   DEBUG CR,"!READ (swHeat)",CR            ' obtain state of heater control
   DEBUGIN DEC Heater
   PAUSE 50
   DEBUG "!READ (swFan)",CR                ' obtain state of fan control
   DEBUGIN DEC Fan
   PAUSE 50
RETURN

SetADC:
    PWM ADC_Vminus, V_Offset * 255/500,100 ' PWM ADC offset voltage
    PWM ADC_Vref, V_Span * 255/500,100     ' PWM ADC Span voltage
RETURN

ReadData:               ' Read ADC 0831
  LOW ADC_CS            ' Enable chip
  SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_ByteValue\9] ' Clock in ADC data
  HIGH ADC_CS           ' Disable ADC
RETURN

CalcTemp:                                  ' Calculate temp in hundredths
  TempF = (V_Span * 100)/255 * ADC_ByteValue + (V_Offset * 100)
RETURN

PlotTemp:
  DEBUG "[", DEC TempF,",/,100]",CR         ' Plot temperature/100
  DEBUG IBIN Heater,BIN Fan,CR              ' Plot state of fan as digital
RETURN
```

- √ Close the Debug Terminal.
- √ Open StampPlot macro sic_pc _incubator_manual.spm.
- √ Connect and plot.
- √ Monitor the temperature to ensure it is reading correctly at room temperature.

The StampPlot macro sic_pc _incubator_manual.spm has the following features:

- Control of the ADC resolved temperature range – Upper and Lower. Note: The BASIC Stamp reads these only in the initialization section of code. If changed, press the RESET button on the Board of Education, or disconnect and connect on StampPlot.
- Control of the heater and the fan.
- Indication of current minimum and maximum temperatures.
- Button to reset the minimum and maximum temperatures.

Now it is time to complete the testing operation:

- √ Test operation of the heater control by turning it on and watching for a rising temperature.
- √ Test operation of the fan control.
- √ Turn off the heater.
- √ Allow the incubator to cool back down to room temperature.
- √ Turn off the fan.
- √ Reset the plot.
- √ Note that minimum and maximum temperatures were recorded in the text boxes.
- √ Click the "Clear Min/Max" temperature button.
- √ Turn on the heater and monitor the heating of the incubator until stable.
- √ Turn off the heater and allow cooling to below 120 °F.

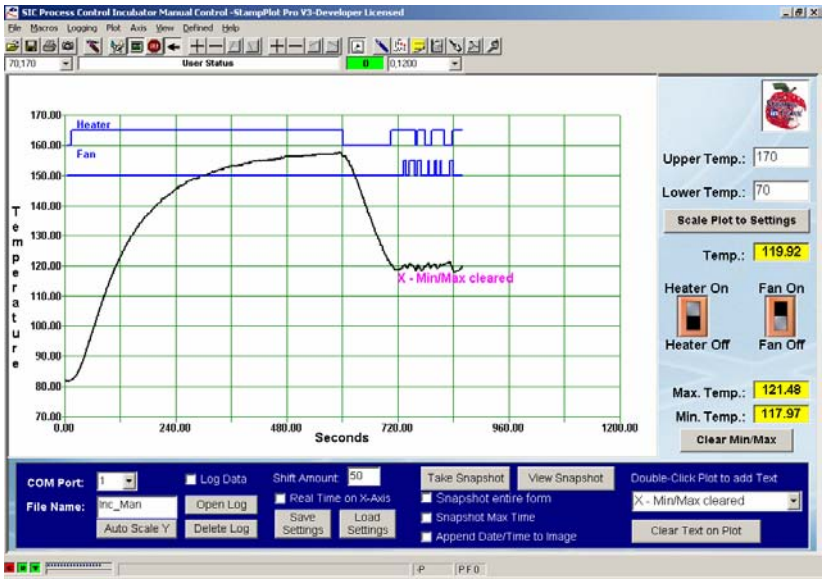
Ready to test your process control skills?

- √ Energize the heater.
- √ When temperature reaches 120 °F, click the "Clear Min/Max" button on StampPlot.
- √ Control the heater and fan for 2 minutes (1 grid division) while controlling temperature as close to 120 °F as possible.
- √ After 2 minutes disconnect on StampPlot. Based on your minimum and maximum temperatures, how did you do?
- √ Disconnect power to your Board of Education to ensure the heater is off.

Figure 6-20 is a screen capture of our testing. The system has a fairly slow response. Note that the dynamics are not that of a first order system on heating. Multiple components of the system come into play. The heater requires time to come up to temperature. This heat must be transferred to the air, and then transferred to the LM34. On cooling, residual heat from the resistor must be dissipated, and energy of the system is lost through the test tube walls in addition to the LM34 giving up its stored heat.

With manual temperature control of the system, it wasn't real easy to maintain it at 120 °F. There was lag time in response, causing overshooting and undershooting of the target setpoint, making control difficult. After a while, and with practice, the user can become accustomed to the system response and gauge more easily the control amounts needed. If you were like us, shortly your concentration drifts and control becomes erratic. Precise process control is best left up to microcontrollers, as they can stay 'focused' on the task.

Figure 6-20 Manual Control of Incubator



Program Discussion

The **`ReadSP_Temps`** subroutine is run when the program is initialized. This routine retrieves data from StampPlot, which is used in setting the ADC span and offset based on desired temperature range. In this program, values are read and used slightly differently than in the previous activity. The **`V_Offset`** variable would be 50 (for 50 °F) in this case instead of 5 for 0.5 V as it was in Activity #2. This affects the math for calculating temperature and PWM. The span is calculated by having StampPlot return the difference between the upper and lower temperature settings.

```
DEBUG "!READ [(txtTMax),-,(txtTMin)]",CR
DEBUGIN DEC V_Span
```

The Fan and Heater controls are read as a 1 (on) or 0 (off), based on whether the virtual switch is up or down. These values are used to directly control the BASIC Stamp outputs. For example, **`Heater`**, in the code below, is assigned to Pin 5, providing control of the BJT powering the resistor.

```
DEBUG CR,"!READ (swHeat)",CR
DEBUGIN DEC Heater
```

Temperature is calculated to hundredths by the BASIC Stamp program, so that 149.22 degrees is stored as 14922. The **PlotTemp** subroutine has StampPlot divide by 100 prior to using the value:

```
DEBUG "[", DEC TempF,",/,100]",CR
```

A typical string sent to StampPlot would be: [14922,/,100]. StampPlot will perform the division prior to plotting or using the value in other ways.

Challenge 6-3: Monitoring Average Temperature

StampPlot maintains multiple values for each of 10 analog channels it can plot. The macro code handles updating the textboxes. The BASIC Stamp could also use them to show the current value in the status box above the plot:

```
DEBUG "!STAT Current Temp: (AINMIN0)",CR
```

StampPlot formatting may be used to limit and pad the number of decimal places used:

```
DEBUG "!STAT Current Temp: [(AINMIN0), FORMAT, 0.00]",CR
```

AINMIN0 is the minimum value of analog channel 0.

Other values of interest:

AINVAL0 – Last analog value for channel 0.
AINMAX0 – Maximum analog value for channel 0.
AINAVE0 – Average analog value for channel 0.

The "Clear Min/Max" button on StampPlot clears the stored values by issuing a "**!CLMM**" instruction (Clear Min Max).

- √ Add code to the **PlotTemp** routine in IncubatorManual.bs2 to display the average temperature in the status box. Format the value for 3 decimal places.
- √ Have StampPlot graph the average temperature by adding it as a second analog value to be plotted. Replace the first line in **PlotTemp** with:

```
DEBUG "[", DEC TempF,",/,100], (AINAVE0)",CR
```

ACTIVITY #4: OPEN LOOP PWM CONTROL

The simplest form of process control is open loop. The block diagram in Figure 6-21 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.

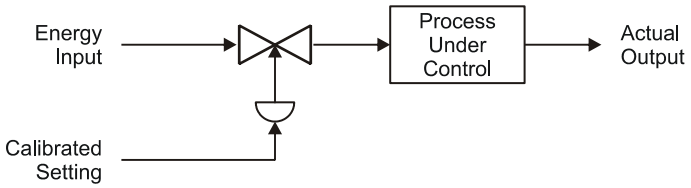


Figure 6-21
Open-Loop Block Diagram

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 6-22. The fluid being drawn from the tank must be kept at constant temperature. Obviously, this will require adding a certain amount of heat to the fluid. (The drive on the transistor determines the power delivered to the heating element.) The question becomes "How much heat is necessary?"

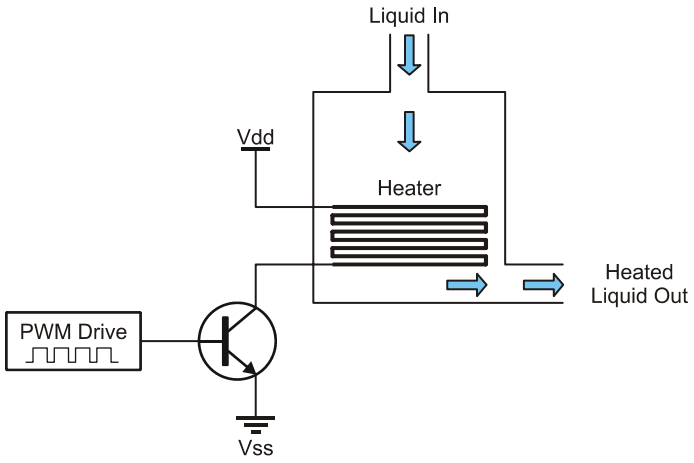


Figure 6-22
Liquid Heating System

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.
- 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 was 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 system at a constant temperature. A good real-world example is that of an incubator used to hatch eggs. To hatch chicken eggs, it is important to maintain a 101.5 °F environment.

Turning on the heater will warm up the interior of the test tube. 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, you will not need to have full power applied to the

resistor. Through a little testing, you can determine just what drive level is needed to yield the correct temperature.

Recall how PWM was used in previous activities to control the speed of the fan. In this activity PWM is used to control the temperature of the incubator. The 100 Ω resistor-heater dissipates 0.81 W of power as heat. If a PWM of 50% is used, how much power would be dissipated? The resistor would be at 0.405 W of course, or 50% of 0.81 W.

At the correct setting of PWM, the heat into the system from the resistor will equal the heat lost from the system at 101.5 °F, to maintain a constant temperature. As long as conditions do not change, the temperature will remain constant.

Parts Required

Same Circuit as Activity #3 (page 203)

WARNING: Do not use an unregulated wall-mount power supply greater than 9 V for the remainder of the activities in this text. A 7.5 V unregulated or 9 V regulated supply is recommended. Fresh 9 V batteries may be used, but they will drain very quickly. The heater resistor will get hot. Care should be taken when touching the resistor and in construction of the incubator. Do not allow the test tube to touch the resistor directly; even so, it may warp. Do not set temperature limits above 120 °F. **NEVER LEAVE POWERED CIRCUITS UNATTENDED!**

Example Program: IncubatorOpenLoop.bs2

√ Enter, save and run the BASIC Stamp program IncubatorOpenLoop.bs2.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - IncubatorOpenLoop.bs2
' Drive incubator with a user defined PWM drive
'
' {$STAMP BS2}
' {$PBASIC 2.5}
'
' -----[ Declarations ]-------------------------------------------------------
ADC_ByteValue  VAR   Byte     ' Analog to Digital Converter data
V_Offset       VAR   Byte     ' Offset voltage read from StampPlot
V_Span         VAR   Byte     ' Span voltage read from StampPlot
TempF          VAR   Word     ' Calculated temp in 100ths of degree F
PWM_Drive      VAR   Byte     ' Amount of PWM for heater
```

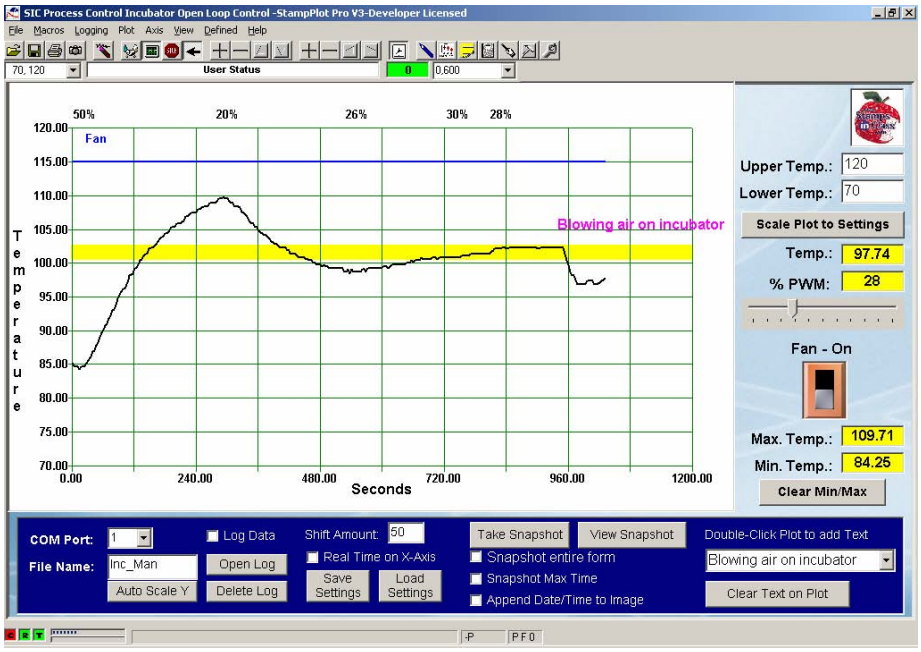
```
x              VAR   Byte     ' General counting variable

ADC_CS         PIN   13       ' ADC Chip Select pin
ADC_Clk        PIN   14       ' ADC Clock pin
ADC_Dout       PIN   15       ' ADC Data output

ADC_VRef       PIN   10       ' Pin for PWM to set ADC voltage span
ADC_Vminus     PIN   11       ' Pin for PWM to set ADC Offset
Heater         PIN   5        ' Pin for heater control
Fan            PIN   0        ' Pin for Fan control

' -----[ Initialization ]----------------------------------------------------
LOW Heater                    ' Heater off
LOW Fan                       ' Fan off
PAUSE 1000                    ' Connection stabilization
GOSUB ReadSP_Temps            ' Get temp span and offset from StampPlot


' -----[ Main Routine ]-------------------------------------------------------
DO
  GOSUB ReadSP_Controls
  GOSUB Drive_Heater
  GOSUB SetADC
  GOSUB ReadADC
  GOSUB CalcTemp
  GOSUB PlotTemp
  PAUSE 500
LOOP

' -----[ Subroutines ]--------------------------------------------------------
ReadSP_Temps:
  DEBUG CR,"!READ (txtTMin)",CR   ' Request and store minimum temp (offset)
  DEBUGIN DEC V_Offset
  PAUSE 50
  DEBUG "!READ [(txtTMax),-,(txtTMin)]",CR ' Request and store temp span
  DEBUGIN DEC V_Span
  PAUSE 50
  RETURN

ReadSP_Controls:
  DEBUG CR,"!READ (sldPWM)",CR   ' Request and store state of heater control
  DEBUGIN DEC PWM_Drive
  PAUSE 50
  DEBUG "!READ (swFan)",CR       ' Request and store state of fan control
  DEBUGIN DEC Fan
  PAUSE 50
  RETURN

Drive_Heater:
  FOR x = 0 TO 20                         ' Drive heater for 20 repetitions
    PWM Heater, PWM_Drive * 255/100, 100  ' %Duty converted to 0-255 for PWM
```

```
  NEXT
  RETURN

SetADC:
  PWM ADC_Vminus, V_Offset * 255/500,100 ' PWM ADC offset voltage
  PWM ADC_Vref, V_Span * 255/500,100     ' PWM ADC Span voltage
  RETURN

ReadADC:                                 ' Read ADC 0831
  LOW ADC_CS                             ' Enable chip
  SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_ByteValue\9] ' Clock data from ADC
  HIGH ADC_CS                              ' Disable ADC
  RETURN

CalcTemp:                                  ' Calculate temp in 100ths
  TempF = (V_Span * 100)/255 * ADC_ByteValue + (V_Offset * 100)
  RETURN

PlotTemp:
  DEBUG "[", DEC TempF,",/,100]",CR        ' Plot temperature/100
  DEBUG IBIN Fan,CR                        ' Plot state of Fan as Digital
  DEBUG "!O txtPWM = ", DEC PWM_Drive,CR   ' Update PWM text
  RETURN
```


- √ Close the Debug Terminal.
- √ Open StampPlot macro sic_pc_incubator_open_loop.spm.
- √ Connect and plot.
- √ Adjust the PWM drive slider to achieve a stable temperature in the desired band of 101.5 °F +/− 1 °F (yellow band in plot).
- √ Once stable for 2 minutes, lightly blow continuously on the incubator test tube. The fan may also be used if moved the farthest possible distance away. The fan will also draw down the power supply voltage, causing heat output of the resistor to drop in addition to increased losses.

Figure 6-23 is our test of the incubator using open-loop control. Note that we allowed temperature to stabilize somewhat prior to changing the drive. With your experience of the system curve you can begin to approximate whether drive is too much or too little after a short while.

Figure 6-23 PWM Open-Loop Control



Around a PWM of 28%, the temperature was in the desired band and fairly stable. At this point, the energy into the system equaled the energy lost for the temperature. What happened when air was blown across the enclosure? There was greater energy lost, and the 28% drive was insufficient to maintain temperature in the band. Given a constant disturbance and time, the incubator would have stabilized at a lower temperature. In this BASIC Stamp program, the temperature is simply acquired and displayed. It is not used for control of the system beyond proper spanning of the ADC.

Consider the earlier analogy of controlling the cabin temperature of an automobile. A typical temperature control is simply a dial or slider that is adjusted to supply air from cold to hot. After driving for a while, you have 'tweaked' the temperature control to your comfort on a hot summer day. Suddenly, a bank of clouds rolls in, blocking the sun. Within a few minutes your cabin temperature is suddenly too chilly. The removal of heat energy via the air conditioner is in excess of the energy entering the cabin from the sun and engine. This is an open-loop system in that the car's controller is not regulating the temperature but simply supplying a set amount of heating or cooling. As losses or gains

change, an open-loop system cannot automatically compensate because there is no feedback informing the system what the actual process variable is doing.

Challenge 6-4: Stand-Alone Control

The BASIC Stamp program and StampPlot interact to control the system. The controller receives input from the computer interface for the settings of the ADC offset and span based on temperature range and the amount of PWM for proper regulation of the incubator.

If StampPlot is not running, the BASIC Stamp will cease to run as it awaits data that never arrives. This leads to excessive expense and possible unreliability of the system. The setting of PWM to control temperature was found by manual control. The setting can also be hard-coded into the BS2 program to run stand-alone.

- √ In the initialization section of the code, set the ADC voltage variables to appropriate levels to control the ADC. This allows monitoring of the system by StampPlot when desired.
- √ In the initialization section of the code, set the PWM drive variable to the appropriate setting, found through testing, to maintain temperature in the 'yellow' under normal conditions.
- √ Comment out (place an apostrophe in front of) any `GOSUB` commands in the program that read values from StampPlot.
- √ Save your program as StandAloneIncubator.bs2.
- √ Disconnect from StampPlot.
- √ Run your code.
- √ Connect briefly with StampPlot to ensure proper temperatures are read and that the PWM control has no effect on the BASIC Stamp (as the slider is moved, the % PWM will still be marked at the top of the plot but should be disregarded).
- √ Disconnect on StampPlot.
- √ After several minutes, connect on StampPlot again to check on the control of the incubator.
- √ Discuss your code changes and results.

CONCLUSION

The ability to measure temperature, or other analog parameters, is dependent on the resolution of the system. Using the ADC0831, an 8-bit analog-to-digital converter, a span of voltages from 0 to 5 V can only be resolved to 255 steps. To increase the resolution, the incoming signal may be amplified and offset to narrow in on the range of voltages of interest. The ADC0831 has built in features to set the span and offset using its Vin(-), Vin(+) and Vref pins. By setting the voltages on these pins, the ADC can be configured to convert a defined range to byte values of 0 to 255. Using filtered PWM, the BASIC Stamp can be programmed to automatically set these pins and adjust the range of conversion for the ADC0831. Combining this device with the LM34 temperature sensor, which has an output of 0.01V/°F, systems temperatures can be monitored to a high degree of resolution.

Another means to increase the resolution would be to use an ADC with higher resolution, such as 12-bit ADC. This type of ADC resolves a range of temperatures to 4095 steps ($2^{12} - 1$). Over the range of 0 to 500 °F (0 to 5 V), this would provide a resolution of 0.122 °F. Another popular style of ADC converts the analog voltage to 10-bits for a range of 0 to 1023.

PWM can also be used to control the power supplied to the control element, a resistive heater in these experiments. Using PWM, the transistor is cycled on quickly, effectively controlling the power supplied to the heater. Recall from Chapter 5 that the filtered PWM may also be used to control the voltage to the element, but the transistor will be controlled in the linear region, causing it dissipate large amounts of power, causing it to heat.

By adjusting the power to the heater, a value of PWM can be found that will control the temperature in the band at or near the setpoint. A drawback to this manual control is that as conditions change, the setting may no longer be appropriate for control at the setpoint. In the next chapter, we will study control scenarios to automatically control a system at the setpoint.

SOLUTIONS TO CHAPTER 6 CHALLENGES

Challenge 6-1 Solution

Part A:
The values of 1 τ and 5 τ should not have varied by very much. The time constant is the same for any step change in temperature.

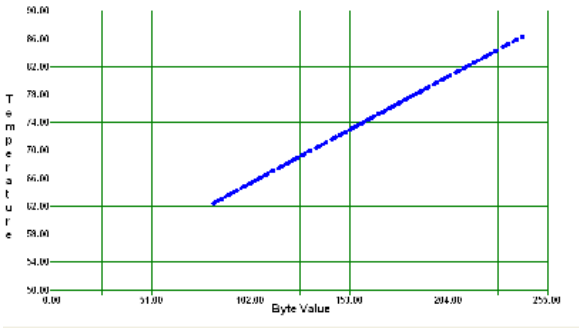
Part B:

1. The plot should have risen quickly then slower its rise as time went on. Just as in cooling to room temperature there is a rapid heat transfer initially. As the temperatures come closer to one another, the transfer slows.

2. If the sensor was dry with nothing around it, the time constants should have been close to previous results for cooling. If there was plastic and or/water around it, the time constants may have slightly longer due to increased mass.

Challenge 6-2 Solution

1. For a range of 50 to 90 °F, an offset of 0.5 V (50 °F) and a span of 0.4 V (0.9 − 0.5).
2. Resolution: 40 °F/255 = .156
3. Your graph may look like this:



4. y = mx + b = (40/255)x + 50 = 0.156(Byte Value) + 50 (Not in BASIC Stamp compatible format.)

5.

a. At 72 °F, LM34 = 0.72 V
b. 72 = 0.156(Byte Value) + 50 (Add −50 to both sides)
   22 = 0.156(ByteValue) (Divide both sides by 0.156 and round)
   141 = Byte Value

Challenge 6-3 Solution

Add this code to **`PlotTemp`**:

```
DEBUG "!STAT Average Temp:[(AINAVE0,FORMAT,0.000)]",CR
```

Challenge 6-4 Solution

Additions/Changes are in bold.

```
' -----[ Initialization ]------------------------------------------------
LOW Heater               ' Heater off
LOW Fan                  ' Fan off
PAUSE 1000               ' Connection stabilization
' GOSUB ReadSP_Temps      ' Get temp span and offset from StampPlot

V_Offset = 70            ' Offset in degrees (hundredths of volt)
V_Span = 50              ' Span in degrees (hundredths of volt)
PWM_Drive = 28           ' Set stand-alone PWM drive


' -----[ Main Routine ]---------------------------------------------------
DO
'  GOSUB ReadSP_Controls
   GOSUB Drive_Heater
```

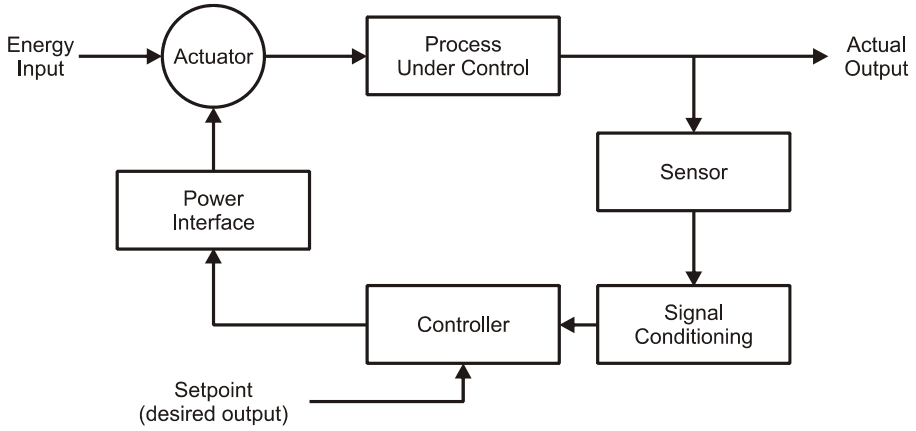
Chapter 7: Closed Loop Process Control

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.

In Chapter 6, the temperature was read by the BASIC Stamp, but only for the purpose of monitoring it. This could have been done as easily with a simple voltmeter. Although you were able to find a PWM drive setting that would yield the desired temperature in Chapter 6, when a long-lasting disturbance changed the system dynamics, 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 7-1.

Figure 7-1 Closed Loop Control Block Diagram



In this diagram, an appropriate sensor is measuring the controlled variable. 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. Chapter 5 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.

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 → 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 7-1.

Table 7-1: 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.

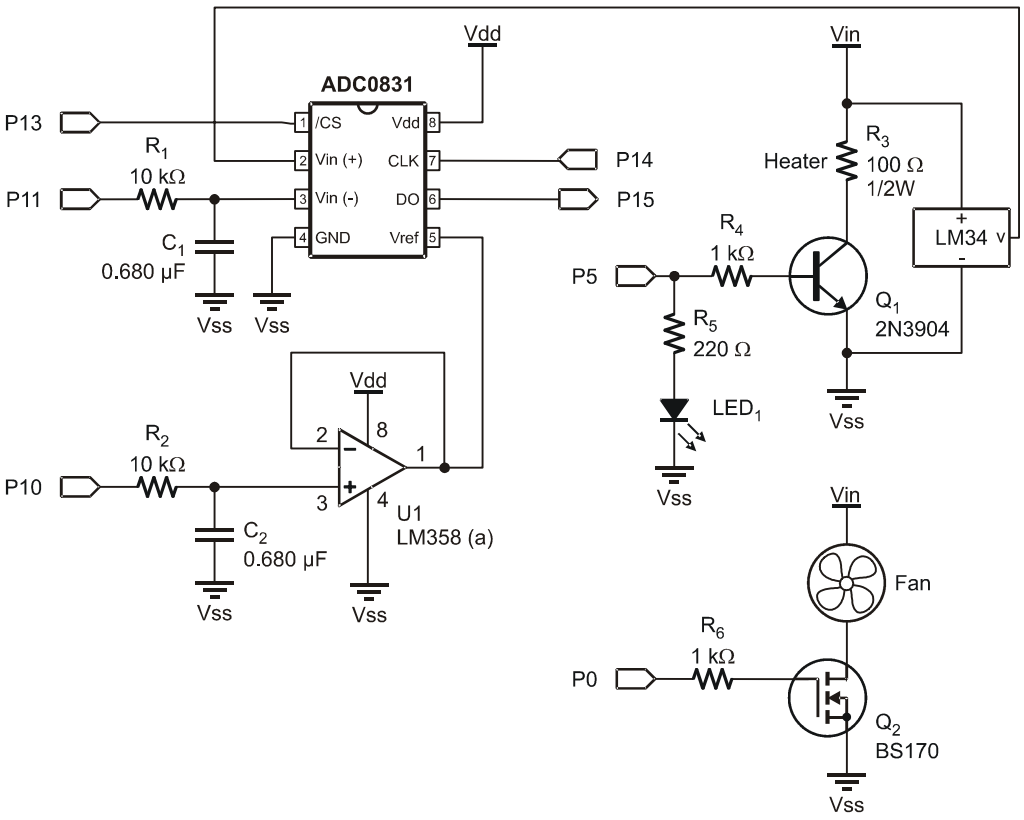
ACTIVITY #1: ON-OFF CONTROL

This activity will focus on converting the open-loop temperature control system of Chapter 6 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 determine how suitable a particular control mode will be.

Parts Required

Same circuit as Chapter 6, Activity #3 (page 203)

Figure 7-2 Incubator Monitoring and Control Circuit (same as Figure 6-17)



We will use the same circuitry to overview and apply proportional, integral, and derivative control modes. Figure 7-2 (identical to Figure 6-17) is a schematic of the circuitry necessary for the next two exercises. The test tube provides the environment we wish to control. The heater drive provides full power for developing heat in the resistor.

Let's assume it is our objective to maintain temperature within the test tube at 101.50 °F within 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 °F is perfect, but the temperature could go up to 102.50 °F or down to 100.50 °F without damage to the embryos. Although it may be hard to imagine an incubator when you look at your test tube, 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.

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 prior programs will 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. Figure 7-3 is a partial flowchart of the control action.

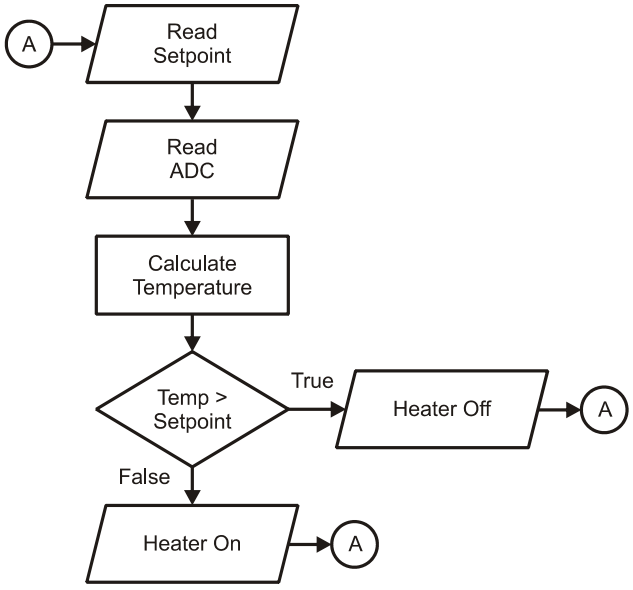


Figure 7-3
Partial Flowchart of On-Off Control

Example Program: IncubatorOnOff.bs2

√ Enter, save and run BASIC Stamp program IncubatorOnOff.bs2
√ Close the Debug Terminal.

```
' -----[ Title ]-----------------------------------------------------------
' Process Control - IncubatorOnOff.bs2
' Comment: Process Control On-Off Control of incubator

' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------
ADC_ByteValue  VAR   Byte     ' Analog to Digital Converter data
V_Offset       VAR   Word     ' Temperature/voltage for ADC offset
V_Span         VAR   Word     ' Temperature/voltage for ADC Span
TempF          VAR   Word     ' Calculated temperature in hundredths
SetPoint       VAR   Word     ' Heater control setpoint

ADC_CS      PIN 13            ' ADC Chip Select pin
ADC_Clk     PIN 14            ' ADC Clock pin
ADC_Dout    PIN 15            ' ADC Data output
ADC_VRef    PIN 10            ' Filtered PWM for ADC Vref
```

```
ADC_Vminus  PIN 11            ' Filtered PWM for ADC Vminus

Heater      PIN 5             ' Incubator heater
Fan         PIN 0             ' Incubator cooling fan

' -----[ Initialization ]---------------------------------------------------
LOW Heater                    ' Ensure heater off
LOW Fan                       ' Ensure fan off
PAUSE 1000                    ' Allow connection to stabilize with StampPlot
GOSUB ReadSP_Span             ' Read StampPlot for ADC

' -----[ Main Routine ]------------------------------------------------------
DO
  GOSUB ReadSP_Controls
  GOSUB SetADC
  GOSUB ReadADC
  GOSUB CalcTemp
  GOSUB Drive_Heater
  GOSUB PlotTemp
  PAUSE 500
LOOP

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


ReadSP_Span:                              ' On connection, get ADC range
  DEBUG CR,"!READ (txtTMin)",CR           ' Request/store offset
  DEBUGIN DEC V_Offset
  PAUSE 50
  DEBUG "!READ [(txtTMax),-,(txtTMin)]",CR ' Request/store Span (Max-Min)
  DEBUGIN DEC V_Span
  PAUSE 50
  RETURN

ReadSP_Controls:
  DEBUG "!READ [(txtSetPoint),*,100]",CR    ' Obtain setpoint in hundredths
  DEBUGIN DEC SetPoint
  PAUSE 50
  DEBUG "!READ (swFan)",CR                  ' Request/control Fan state
  DEBUGIN Fan
  PAUSE 50
  RETURN

SetADC:
  PWM ADC_Vminus, V_Offset * 255/500,100    ' Set Voltage for ADC Vminus
  PWM ADC_Vref, V_Span * 255/500,100        ' Set voltage for ADC Vref
  RETURN

ReadADC:                ' Read ADC 0831
  LOW ADC_CS            ' Enable chip
  SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_ByteValue\9] ' Clock in ADC data
```

```
  HIGH ADC_CS           ' Disable ADC
  RETURN

CalcTemp:               ' Transfer function for temperature in hundreths
  TempF = (V_Span * 100)/255 * ADC_ByteValue + (V_Offset * 100)
  RETURN

Drive_Heater:
  IF TempF > SetPoint THEN                  ' Determine heater state
    LOW Heater                              ' Above setpoint - Heat off
  ELSE
    HIGH Heater                             ' Below setpoint - Heat on
  ENDIF
  RETURN

PlotTemp:
  DEBUG "[", DEC TempF,",/,100]", ",",      ' Plot temperature
        "[", DEC SetPoint, ",/,100]",CR     ' and setpoint
  DEBUG "!O txtTemp=(AINVAL0)",CR           ' Update temperature reading
  DEBUG IBIN Heater, BIN Fan,CR             ' Plot fan as digital
  RETURN
```


- √ Open StampPlot macro sic_pc_incubator_on_off.spm
- √ Verify the Lower and Upper Temperatures for monitoring are 70 to 120 respectively.
- √ Verify the setpoint is 101.5.
- √ Connect and Plot
- √ Note the action of the incubator as it rises to the setpoint.
- √ Once temperature has stabilized, clear the min/max temperatures and monitor for several minutes.

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.

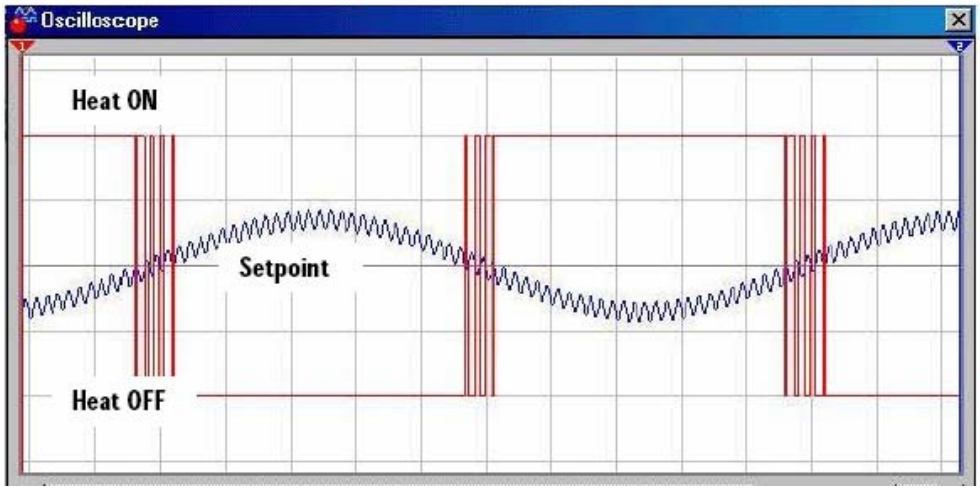
Overshoot becomes large when the heat energy contained in the mass of the resistor is large, relative to the heat already in the test tube. The test tube is small, but the mass of

the half-watt resistor also is small. As a result, the overshoot of your system will probably only be several degrees.

When the temperature does turn around and begins to fall, as it passes the setpoint, the heater is once again turned on. Undershoot may 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 overshoot depends 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, note the minimum and maximum overshoot values in the minimum and maximum temperature boxes.

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 7-4 represents this problem.

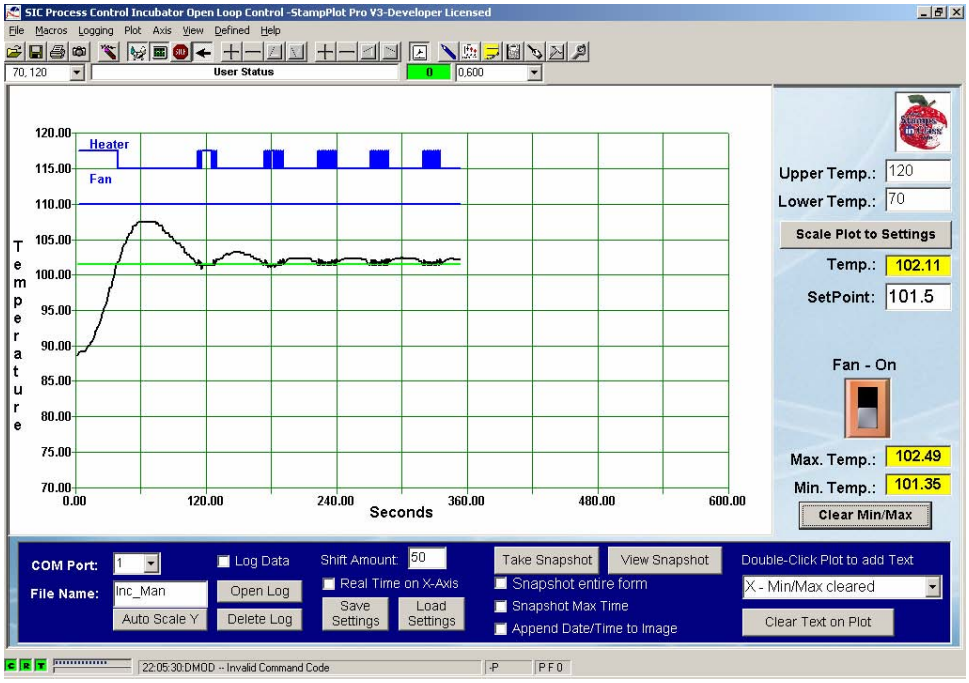
Figure 7-4 Effects of Noise on Signal Causing Heater Cycling



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 result 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?

You may note the temperature fluctuating slightly as the heater cycles as shown in Figure 7-5. The heater draws significant current, dropping the supply voltage. This in turns affects the PWM voltages controlling the ADC. As the heater cycles on and off, the data from the ADC representing temperature varies slightly. This is another form of noise caused by cycling of high-current loads.

Figure 7-5 Temperature Cycling Around Setpoint with On-Off Control



Additional Information

The op-amp can be employed here to control an on-off system without the expense of a microcontroller, ADC, and all the additional hardware. While we won't ask you tear down your circuit for this, you may wish to try it at a later time. Consider the schematic in Figure 7-6.

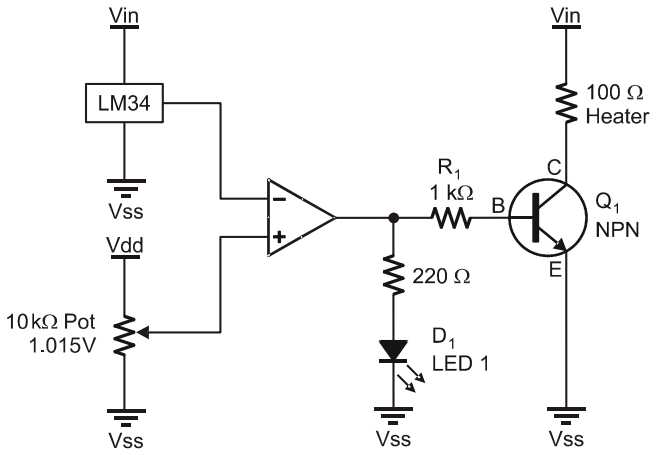


Figure 7-6
Op-Amp On-Off Control Circuit

DO NOT BUILD

Recall the output of the op-amp attempts to drive the inverting input (−) towards the non-inverting input (+). In this configuration, if the temperature is lower than the setpoint (1.015 V = 101.5 °F), the op-amp output will go high in an attempt to drive the inverting input down to the non-inverting. But the output is NOT connected to the inverting input, so the op-amp will drive fully to the positive supply-voltage rail in a futile attempt at affecting the inverting terminal. This causes the transistor to go into saturation, turning on the heater.

Once temperature rises above the setpoint, the op-amp will drive to the other rail voltage (ground/Vss) in an attempt to lower the inverting input towards the non-inverting input. This will have the effect of turning off the heater. Thus, the heater will cycle around the setpoint.

This op-amp configuration is called a comparator because its output is based on a comparison of the two inputs. A digital signal output is based on comparing two analog voltages.

Challenge 7-1

Part A: Stand-Alone Control with Fan

- √ Save IncubatorOnOff.bs2 under the name IncubatorOnOffFan.bs2.
- √ Modify your program to control temperature at 101.5 °F with no interactive control with StampPlot. Refer back to the Stand-Alone Control Challenge on page 217 for hints on accomplishing this. Use StampPlot for monitoring only to ensure proper operation.
- √ Modify the program to energize the fan should temperature exceed the setpoint by 3 degrees. It should turn off once the temperature falls below that limit.
- √ Write your modified code, and capture a plot of the control action. To achieve 104.5 °F, you may either rely on overshoot from a low temperature or remove the incubator canister and briefly use a lighter.

Part B: Op-Amp Temperature Setpoint Detection

Consider the On-Off op-amp configuration in Figure 7-6. Instead of using an ADC for measuring temperature, how could this configuration be used to simply indicate to the BASIC Stamp if a temperature is above or below a given value?

- √ Draw a schematic and show control code for this.

ACTIVITY #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. Good examples are the air conditioners or heaters in your home. Your furnace does not kick on just above your thermostat setting, and off just below it. Whether using a mercury switch or an electric thermostat, there is a control gap. If it is set for 68 °F, it may kick on at 66 °F and off at 70 °F. This allows a 4 degree gap for control to prevent the unit from continually cycling on and off over a very short period of time, which wastes energy and stresses the mechanical components. For ideal hatching of eggs with the incubator, it is allowable to have a 0.5-degree variance from the setpoint allowing control between 101.0 °F and 102.0 °F.

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 7-7 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.

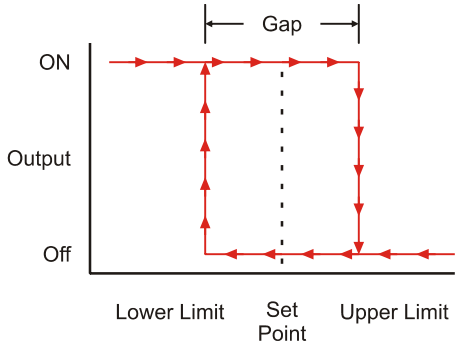


Figure 7-7
Differential Gap (Hysteresis) Control Graph

Notice how the diagram in Figure 7-8 differs from the earlier one depicting noisy data in a simple on-off control mode.

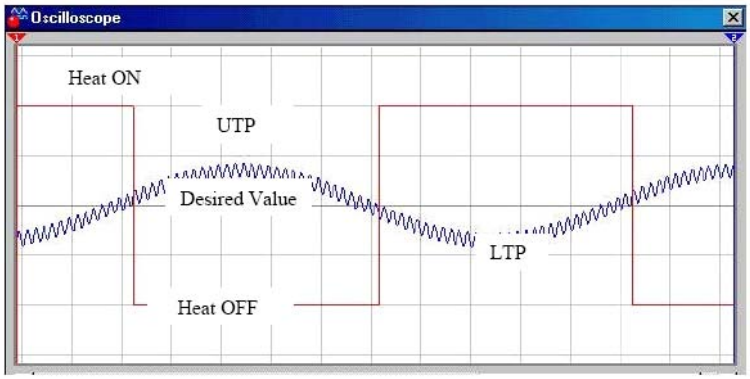


Figure 7-8
Effects of Noise on Control Using Differential-Gap

Because the differential gap is wider than the effect of the noise, rapid cycling is eliminated. 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

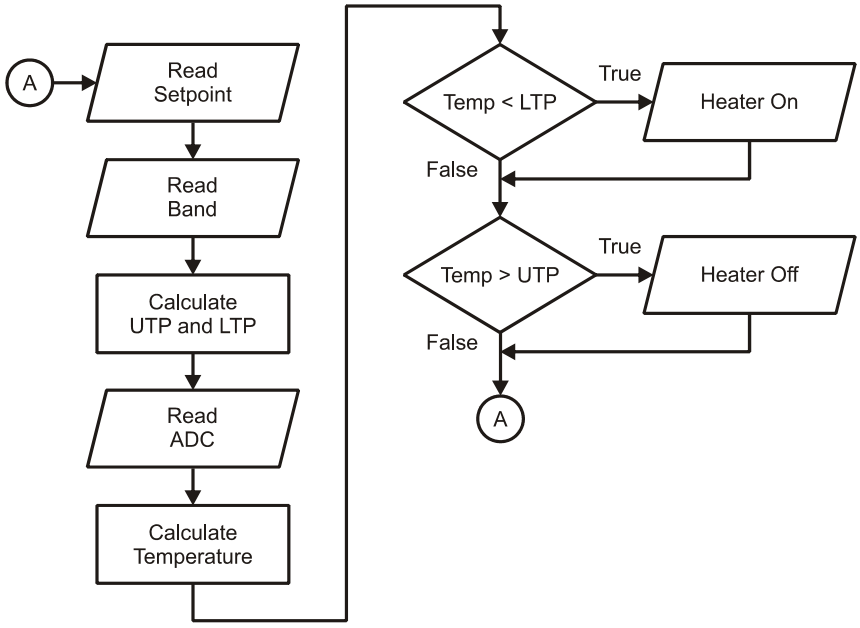
still result in some amount of overshoot and undershoot. We are accepting a wider variance in temperature. When processes allow this variance, differential-gap control is usually preferred over simple on-off control.

Parts Required

Same as previous Activity

Modifying the program and StampPlot only requires a little code to adapt the BASIC Stamp program and the StampPlot macro. Figure 7-9 is a flowchart representing the control action. The allowable band must be read from StampPlot, values for Upper Trip Point (UTP) and Lower Trip Point (LTP) calculated, and the heater controlled based on these setpoints.

Figure 7-9 Representative Flow Chart of Differential-Gap Control



Example Program: IncubatorDiffGap.bs2

- √ Enter, save and run BASIC Stamp program IncubatorDiffGap.bs2.
- √ Close the Debug Terminal.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - IncubatorDiffGap.bs2
' Controls using differential gap.
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------

ADC_ByteValue VAR Byte ' Analog to Digital Converter data
V_Offset      VAR Word ' Temperature/voltage for ADC offset
V_Span        VAR Word ' Temperature/voltage for ADC Span
TempF         VAR Word ' Calculated temperature in hundredths
SetPoint      VAR Word ' Heater control setpoint
Band          VAR Byte ' Setpoint band +/-
UTP           VAR Word ' Upper trip point based on band
LTP           VAR Word ' Lower trip point based on band

ADC_CS        PIN 13   ' ADC Chip Select pin
ADC_Clk       PIN 14   ' ADC Clock pin
ADC_Dout      PIN 15   ' ADC Data output
ADC_VRef      PIN 10   ' Filtered PWM for ADC Vref
ADC_Vminus    PIN 11   ' Filtered PWM for ADC Vminus
Heater        PIN 5    ' Incubator heater
Fan           PIN 0    ' Incubator cooling fan

' -----[ Initialization ]-----------------------------------------------
LOW Heater             ' Ensure heater off
LOW Fan                ' Ensure fan is off
PAUSE 1000             ' Allow connection stabilization
DEBUG CR,"!O lblBand.Delete",CR           ' Delete control if exists

' Create label control "lblBand"
DEBUG "!O oLabel.lblBand=82.5,64,13,3,(+/-):,,0,10,0",CR
DEBUG "!O lblBand.Font=Arial,10,1,0",CR   ' Set Font of lblBand
DEBUG "!O txtBand.Delete",CR              ' Delete control if exists

' Create text box control "txtBand"
DEBUG "!O oText.txtBand=92,64,7,5,1,15,0,11",CR
DEBUG "!RSET",CR                        ' Reset StampPlot
GOSUB ReadSP_Span                       ' Get settings for ADC

' -----[ Main Routine ]-------------------------------------------------
DO
  GOSUB ReadSP_Controls
  GOSUB CalcTripPoints
```

```
  GOSUB SetADC
  GOSUB ReadADC
  GOSUB CalcTemp
  GOSUB Drive_Heater
  GOSUB PlotTemp
  PAUSE 500
LOOP
' -----[ Subroutines ]------------------------------------------------------
ReadSP_Span:
  DEBUG CR,"!READ (txtTMin)",CR         ' Obtain value for ADC Offset
  DEBUGIN DEC V_Offset
  PAUSE 50
  DEBUG "!READ [(txtTMax),-,(txtTMin)]",CR ' Obtain value for ADC Span
  DEBUGIN DEC V_Span
  PAUSE 50
  RETURN

ReadSP_Controls:
  DEBUG "!READ [(txtSetPoint),*,100]",CR   ' Obtain setpoint in hundredths
  DEBUGIN DEC SetPoint
  PAUSE 50
  DEBUG "!READ [(txtBand),*,100]",CR    ' Obtain band in hundredths
  DEBUGIN DEC Band
  PAUSE 50
  DEBUG "!READ (swFan)",CR              ' Obtain fan based on state
  DEBUGIN Fan
  PAUSE 50
  RETURN

CalcTripPoints:
  UTP = SetPoint + Band                 ' Calculate UTP
  LTP = SetPoint - Band                 ' Calculate LTP
  RETURN

SetADC:
  PWM ADC_Vminus, V_Offset * 255/500,100   ' Set Voltage for ADC Vminus
  PWM ADC_Vref, V_Span * 255/500,100       ' Set voltage for ADC Vref
  RETURN

ReadADC: ' Read ADC 0831
  LOW ADC_CS ' Enable chip
  SHIFTIN ADC_Dout, ADC_Clk, MSBPOST,[ADC_ByteValue\9] ' Clock in ADC data
  HIGH ADC_CS ' Disable ADC
  RETURN

CalcTemp:             ' Transfer function for temperature in hundredths
  TempF = (V_Span * 100)/255 * ADC_ByteValue + (V_Offset * 100)
  RETURN

Drive_Heater:
  IF (TempF < LTP) THEN
```

```
    HIGH Heater       ' Heat On If less than LTP
  ENDIF

  IF (TempF > UTP) THEN
    LOW Heater        ' Heat off if greater than UTP
  ENDIF
  RETURN

PlotTemp:
  DEBUG "[", DEC TempF,",/,100]", ",", ' Plot current temp,
  "[", DEC SetPoint, ",/,100]",",",    ' and setpoint,
  "[", DEC UTP, ",/,100]",",",         ' and UTP
  "[", DEC LTP, ",/,100]",CR           ' and LTP
  DEBUG "!O txtTemp=(AINVAL0)",CR      ' Update current temp text
  DEBUG IBIN Heater, BIN Fan,CR        ' Plot heater as digital
  RETURN
```

- √ Open StampPlot macro sic_pc_incubator_on_off.spm.
- √ Verify the Lower and Upper Temperatures for monitoring are 70 to 120 respectively.
- √ Verify the setpoint is 101.5.
- √ Connect and Plot.

Note that the code adds a band control (+/−) to StampPlot to adjust the control band. Shortly, we will discuss how this was performed.

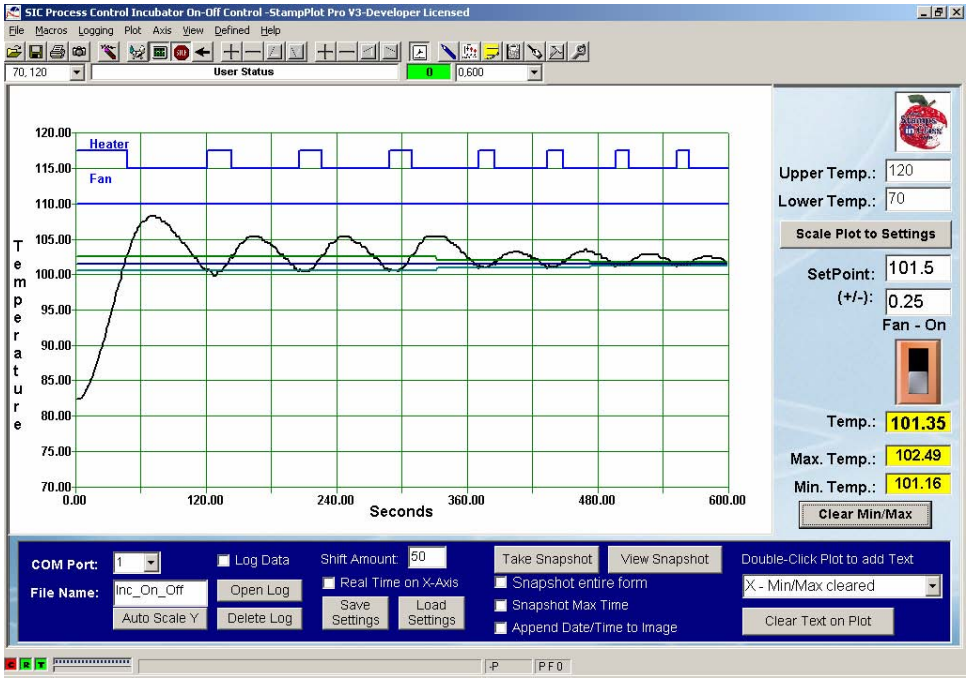
- √ Verify the control band is set to (+/−) 1.
- √ Note the action of the incubator as it rises to the setpoint.
- √ Once temperature has stabilized after several cycles, clear the min/max temperatures and monitor for several minutes.
- √ Adjust the band (+/−) to achieve the smallest control band for minimizing overshoot or undershoot while preventing actuator cycling due to noise.

The smallest possible value is 0.01 since the BASIC Stamp is controlling in hundredths of degrees. A gap of 0.01 °F may not be a realistic value depending on the resolution of the ADC. To find the smallest temperature change, divide the resolved temperature span (120 − 70 = 50 by default) by the 255 to obtain the resolution.

Figure 7-10 is a plot of our results. Note that the UTP and LTP are plotted along with the setpoint and current value. Are the eggs harmed when temperature overshoots 105 °F? Probably not. Remember that the transfer of energy is not instantaneous. The eggs, based on their mass and heat conduction properties, have a time constant. A short rise

above the limits would be insufficient to raise the egg's temperature appreciably. If this were an actual incubator, the response of the system would be much slower. Consider what would happen if the temperature began to drop due to a disturbance, such as opening the incubator for a short period of time. As the incubator's air temperature dropped, the heat stored in the eggs would be transferred to the air, helping to stabilize the temperature. The greater the mass, the longer the time constant of the system.

Figure 7-10 Differential-Gap Control



Program Discussion

While most of the code should be fairly easy to follow, the ability to add controls to StampPlot from the BASIC Stamp is something new. Note that the code begins by deleting the named control we are about to create. This is done to ensure a duplicate is not created if you reset the BASIC Stamp and the code is run again.

```
DEBUG CR, "!O lblBand.Delete",CR            ' Delete control if exists
```

A control on StampPlot can be created by defining what type it will be, naming it, and providing configuration parameters such as coordinates, size, text, etc.

In the code a label (**`oLabel`** – Object Label) is first created called **`lblBand`** and its parameters are set:
X coordinate (82.5), Y Coordinate (64), width (13, height (3), text in it (+/−), color (0-Black), font size (10) and bordered or not (0, 1 = border).

```
' Create label control "lblBand"
DEBUG "!O oLabel.lblBand=82.5,64,13,3,(+/-):,,0,10,0",CR
```

The font is set using the given name where the 1 parameter in it sets the font to bold.

```
DEBUG "!O lblBand.Font=Arial,10,1,0",Cr  ' Set Font of lblBand
```

Next a text box (**`oText`** – Object Text) called txtBand is created using similar parameters. For more information on the different controls, please see the StampPlot help files. Note that the code begins by deleting the named control. This is done to ensure a duplicate is not created if you reset the BASIC Stamp and the code is run again.

```
DEBUG "!O txtBand.Delete",CR             ' Delete control if exists

' Create text box control called txtBand
DEBUG "!O oText.txtBand=92,64,7,5,1,15,0,11",CR
```

Once created, the BASIC Stamp may read or update these controls by name.

Challenge 7-2

Part A: Adding a Heater-Run Control
Currently, we have no way short of turning off the BASIC Stamp to de-energize the heater. The following StampPlot code will create a checkbox control for allowing the user to turn off the heater manually:

```
!O oCheck.chkHeatRun=82.,53.,8.,8.,Heater Run,0,(BLUE),(WHITE),11
```

√ Modify the BASIC Stamp program to:
- Create this control on initialization.
- Read this control as part of the **`ReadSP_Controls`** subroutine and store into a bit sized variable.
- Use the new variable to determine if the heater should be energized when temperature is below the lower trip point. (HINT: Review the use of Boolean operators in Chapter 3).

Part B: Op-Amp Differential Gap Temperature Detection
Instead of the ADC being used to monitor temperature, how could two op-amps be used as comparators to inform the BASIC Stamp when the actual temperature is above or below the upper and lower trip points of 102.0 and 101.0 respectively?

√ Draw a schematic and show control code for this.

CONCLUSION

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 a 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.

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

Increased cycle time and noise immunity around 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 exercises of Chapter 6, a value of drive between fully-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. Chapter 8 will investigate controls that take an appropriate amount of action based on an evaluation of the measurement. Proportional, Integral, and Derivative control theory can be employed to maximize the effectiveness of the control system.

SOLUTIONS TO CHAPTER 7 CHALLENGES

Challenge 7-1 Solution

Part A:

√ In the Initialization section of your program, comment out **GOSUB READSP_Span**, then set the temperature setpoint in hundredths.

```
'GOSUB ReadSP_Span          ' Read StampPlot for ADC
SetPoint = 10150
```

√ Also in the Initialization section, set the values for **V_Offset and V_Span**:

```
V_Offset  =    70
V_Span    =   50
```

√ In the main **DO...LOOP**, comment out **GOSUB ReadSP_Controls**

```
'  GOSUB ReadSP_Controls
```

√ Add a subroutine to control the fan.

```
Drive_Fan:
 IF TempF > (SetPoint + 300) THEN  ' Check fan state for setpoint +3F
   HIGH Fan                        ' Above - Fan on
 ELSE
  LOW FAN                          ' Below - Heat off
 ENDIF
 RETURN
```

√ Add a subroutine call in the **DO...LOOP**:

```
GOSUB Drive_Fan
```

Part B:

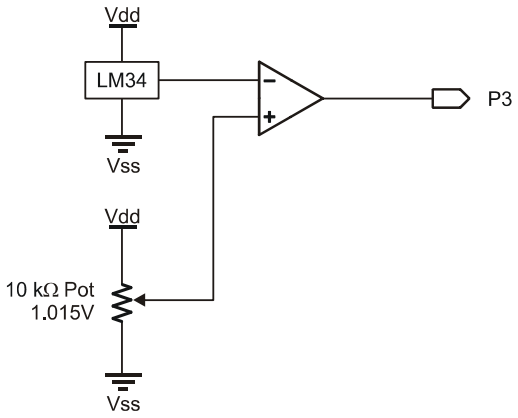


Figure 7-11
Solution for High/Low Temperature Sensing

In the schematic for Figure 7-11, if the V(LM34) > V(Setpoint), the output will be low (0). If V(LM34) < V(Setpoint), output will be high (1). Assumes remainder of current code present for monitoring.

```
Temp  PIN 3

Drive_Heater:
  IF Temp = 0 THEN     ' If Op-Amp low, Voltage LM34>Voltage Setpoint
    LOW Heater
  ELSE
    HIGH Heater
  ENDIF
  RETURN
```

Challenge 7-2 Solution

Part A:

√ In the Initialization section add:

```
DEBUG "!O chkHeatRun.Delete",CR       ' Delete duplicate if it exists

' Create control
DEBUG "!O oCheck.chkHeatRun=82.,53.,8.,8.,Heater Run,0,(BLUE),(WHITE),11", CR
```

√ In the **`ReadSP_Controls`** routine, add:

```
 DEBUG "!READ (chkHeaterRun)",CR   ' Request/control fan based on state
   DEBUGIN HeaterRun
   PAUSE 50
```

√ Add this variable in the Declarations section:

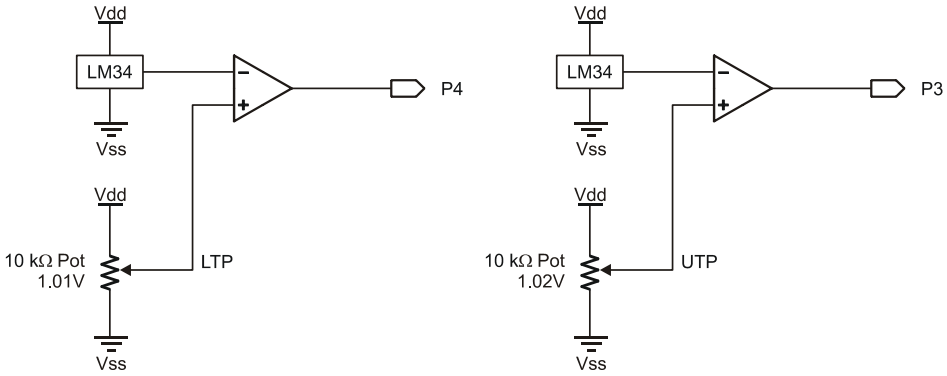
```
HeaterRun   VAR  bit
```

√ Modify **`Drive_Heater`** routine:

```
Drive_Heater:
  IF (TempF < LTP) AND (HeaterRun = 1) THEN
    HIGH Heater           ' Heat On If less than LTP and heater enabled
  ENDIF

  IF (TempF > UTP) THEN
    LOW Heater            ' Heat off if greater than UTP
  ENDIF
RETURN
```

Part B:

Figure 7-12 Solution: Op-Amp Configuration for Differential-Gap Control



- If V(LM34) > V(UTP), P3 will be Low
- If V(LM34) > V(LTP), P4 will be Low

- If V(LM34) < V(UTP), P3 will be High
- If V(LM34) > V(LTP), P4 will be Low

- If V(LM34) < V(UTP), P3 will be High
- If V(LM34) < V(LTP), P4 will be High

We need to turn off the heater when above UTP and off when below LTP. Assumes remainder of current code present for monitoring.

```
Temp_UTP   PIN  3
Temp_LTP   PIN  4

Drive_Heater:
  IF (Temp_UTP = 1) THEN
      HIGH Heater
  ENDIF

  IF (Temp_LTP = 1) THEN
      LOW Heater
  ENDIF
RETURN
```

Chapter 8: Proportional-Integral-Derivative Control

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

One objective of a process control method is to hold a system constant. In the previous chapter we used various means of cycling the heater of our incubator to maintain a desired temperature. Using differential gap, we created an allowable band, or range of temperatures, in which the heater would cycle, causing the temperature to cycle on and off, above and below the desired setpoint. In Chapter 6, we saw how we could use pulse-width modulation (PWM) to add variable amounts of energy to our system in duty cycles between 0% (fully off) and 100% (fully on). While these types of control have their advantages and disadvantages, PID control allows greater control of a system but can be more difficult to implement and tune (or adjust) for optimum performance. Holding a process constant involves continually adding energy that exactly equals the system energy losses. If the system losses were constant, then 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 energy losses due to a disturbance is not felt immediately. Furthermore, an instant change in drive does not create an instant output response. Process control can be as much an art form as it is a science. The first step to understanding PID control is to realize that every system has both gains and losses of energy.

$$E_{System} = E_{In} - E_{Out}$$

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

$$\text{Equilibrium: } E_{In} = E_{Out}$$

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. Conditions of the system will change the energy loss, such as room temperature changing, air movement changes around the

tube, sunlight falling on the tube, or the resistor aging. The amount of heat added and removed is seldom constant over long periods of time.

In an oil-flow system, the drive element, the pump, is controlled to maintain a desired oil flow rate. A sudden change in the system may be a valve closing, blocking one path for oil flow. A slow change in the system may be a filter gradually clogging or the oil temperature changing affecting the viscosity or 'thickness'. The pump needs to adjust in order to compensate for these changes.

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 the air blowing over the car. When the car is maintaining a constant speed, the system is in equilibrium and our foot keeps the accelerator in a constant position. When conditions change, such as the car climbing a hill, the car's velocity changes. The forces increase on the car, removing 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 energy losses equal energy input, and a new equilibrium is reached.

Throttle position directly relates to the power demanded from the engine. This power moves the mass of the car and overcomes friction, wind resistance, gravity due to road incline, etc. If all of these conditions (disturbances) were constant, our car would stabilize as some constant speed. When you want to go 55 mph on level highway, this would correlate to a specific throttle position. If all conditions stay the same, lock in this throttle position and your speed will not change.

Now, think carefully about how you might react in an effort to keep the car going at a constant speed. Consider how you would respond in pressing the gas pedal relative to continually watching the speedometer. The speedometer is the measuring device (sensor) and it is providing you feedback as to how your process is going. We will relate your "natural" reaction to the individual principles of a PID controller as you attempt to drive a steady 55 miles per hour (mph).

First, let's consider your simple response to noticing that you are going 5 mph slower than you desire. You respond by pressing the gas pedal a certain amount. If you had been going 10 mph slower than desired, your reaction would naturally be more forceful. And, likewise if you had been only slightly below the target speed your tendency would

be to only change the pedal position a small amount. Your control reaction is proportional to the absolute magnitude of the mph error.

Next, consider that your car is going up a long continuous hill and your speed drops to 45 mph. At 10 mph below your desired setpoint, the proportional reaction above was enough to overcome some of the effects of the incline but, due to the long hill, your car stabilizes at 50 mph. What would you do? Understanding your objective is to go 55 mph, you would naturally press down a little harder. And, if that brought the speed up to 52 mph, would you stop pressing? No, you would press a little more and a little more until finally you had integrated exactly enough control action to compensate for the steepness of the hill and be at 55 mph. As a smart controller, you would not allow long-term continuous error to persist in your process.

Finally, consider how you might react to driving up a steep hill versus the last long continuous hill. Remember that in our example you are reacting to what you see the speedometer doing – it is providing you process feedback information. If all of a sudden you notice your speed is dropping quickly, what do you assume? Right, you have just gotten onto a very steep hill. What will happen if you do not respond quickly and forcefully to this rapid change in your speed? Things may only get worse, right? Having an element of control in which you respond relative to the rate at which your process is changing can help buffer the affects of rapidly occurring, high magnitude disturbances. Responding to the rate of change at which an error is occurring is the function of derivative control. When you work with derivatives in mathematics you are usually analyzing the slope at points along a changing curve. Derivative control action responds solely to the slope of the error, or the speed at which the value is changing.

Hopefully, as you think of yourself as an intelligent PID controller in our driving example, you can see how a blended response to the absolute magnitude of error (P), the duration that an error persists (I), and the rate that error is occurring (D) can provide very efficient and effective process control. Determining the appropriate blend of the individual control modes must take into account the dynamics of the entire system. As with all continuous process control scenarios, keeping your car at a steady speed is dynamic. There is a dynamic to the magnitude, duration, and rate at which road conditions change. The time required for individual drivers to notice, evaluate, and react to speedometer changes is dynamic. And, the weight, engine torque, and transmission ratio are only a few of the dynamic variables of your car that defines its response to a throttle change. As a "smart controller" behind the wheel you will find yourself quickly adapting the blend of your PID response based on the characteristics of your vehicle as

well as the driving conditions. Keep this example in mind as you continue through this chapter.

Conditions (or disturbances on our system) can change very rapidly; such as when the car suddenly encounters a hill. Or may be very slow; such as tire wear reducing efficiency. 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 for which an error remains.
3. How quickly an error occurs in the system, or the "rate" of change.

The sum of these three evaluations comprises the output drive in an attempt to maintain a system in equilibrium. Figure 8-1 illustrates the evaluation and control of a system for PID control.

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

$$Co_{PID} = Bias + K_P E + K_I \int Edt + K_D \frac{dE}{dt}$$

or

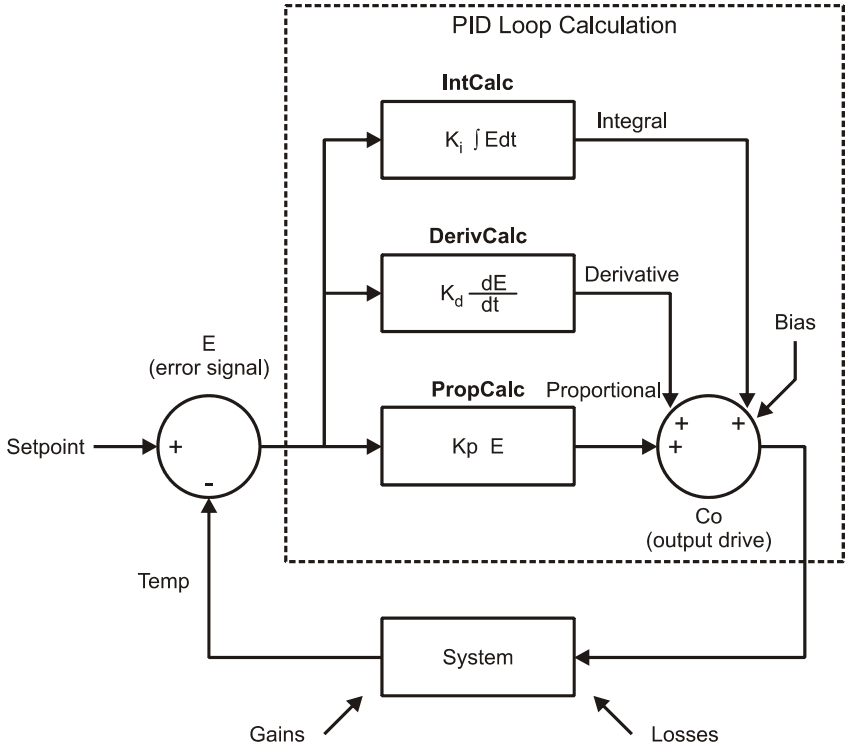
$$DRIVE_{TOTAL} = DRIVE_{BIAS} + DRIVE_{PROPORTIONAL} + DRIVE_{INTEGRAL} + DRIVE_{DERIVATIVE}$$

Where:

Co = controller output or drive
Bias = User defined drive
E = Amount of Error*
t= Time
K_P = proportional gain
K_I = integral gain
K_D = derivative gain
dE = Change in error
dt = Change in time

*(Earlier E was used for Energy. Symbols have multiple uses).

Figure 8-1 Evaluation and Control of PID



Breaking down the control in English:

- The controller will deliver a level of drive based on the calculations to try and maintain the system at the desired setpoint.
- **Bias Drive** is the amount of signal needed to drive the system at the setpoint under normal conditions. In Chapter 6, the PWM drive to the heater was manually adjusted until the incubator's temperature was in the operating band. Under constant conditions, approximately 30% of output drive was necessary to keep the system in the band. But, as we saw, when air was blown over the incubator, conditions changed, the drive remained constant and the system dropped below our defined operating limit.

- **Error** (E) is the measurement from the desired setpoint to the actual system condition, such as a difference between the desired temperature and the actual temperature.
- **Proportional Drive** continuously evaluates error and adds or subtracts output drive in an attempt to drive the system back to the setpoint. Too low? More drive. Too high? Less drive. The gain is used to adjust how much action is taken based on the error.
- **Integral Drive** measures the duration and magnitude of the error and adds or subtracts to eliminate the long lasting error. The longer the error and the greater the amount of error, the greater the integral drive will be. Gain is used to adjust the amount of action based on the duration and magnitude of error.
- **Derivative Drive** measures how quickly error changes in respect to time. The faster the error changes, the more action that will be taken to oppose the change to bring the system back under control. The gain once again is used to adjust the amount of action based on this calculation.

With each element in PID, the control action is based on the error, either current, accumulated over time, or change with respect to time. Mathematically, integrals and derivatives are based on an infinite number of points in the time continuum. Consider Figure 8-2 showing an actual reading oscillating around a setpoint. Data is measured and action taken at every possible point. The integral evaluation provides the area under the curve. As an example, if the error resulted in a certain amount of power being expended, the integral of the error would provide the total power used over time. The mathematical result would provide a very precise value for the area under the curve. Note that if a value above the setpoint were considered a positive error, data below the setpoint would be a negative error. If the areas A and B were equal and these two areas were integrated, the result would be 0.

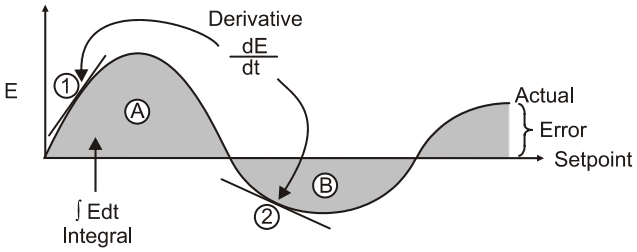


Figure 8-2
Continuous Measurement for PID

The derivative evaluation provides a measurement of the change in value with respect to time (the faster the error changes, or the slope at that instant, the higher the magnitude of the error.) At point 1 of Figure 8-2, the value is changing very rapidly with a positive slope (rising). At point 2, the value is changing slowly with a negative slope (decreasing).

Analog circuits using op-amps can provide true proportional, integral and derivative drive where the waveform is processed on a continuous basis. At any given instant, the error is measured and processing of that error is made. In microcontrollers, instantaneous measurement is not possible. Samples are taken at intervals, and approximations are made to provide PID control. Before we test the microcontroller performing PID control, let's first look at a system controlled with analog PID.

Electronic Systems Technologies students, at Southern Illinois University Carbondale, regularly construct and test the floating-ball project as shown in Figure 8-3. An infrared beam is used to detect the position of the magnetic ball. Drive to the electromagnet is used to control the position of the ball. The setpoint is the desired voltage representing the light falling on the detector based on the ball partially in the beam. The actual ball position is detected based on the amount of light striking the phototransistor and producing an output at the collector. The error is the difference between the actual and desired voltages. As the ball begins to drop (light level increases), drive to the magnet must increase, and as the ball gets pulled up (light level decreases), the drive to the magnet must be decreased. When properly adjusted, or tuned, the ball will maintain a relatively stable position, "floating" in the beam.

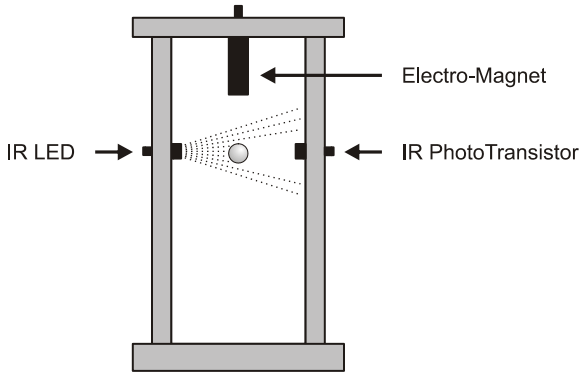
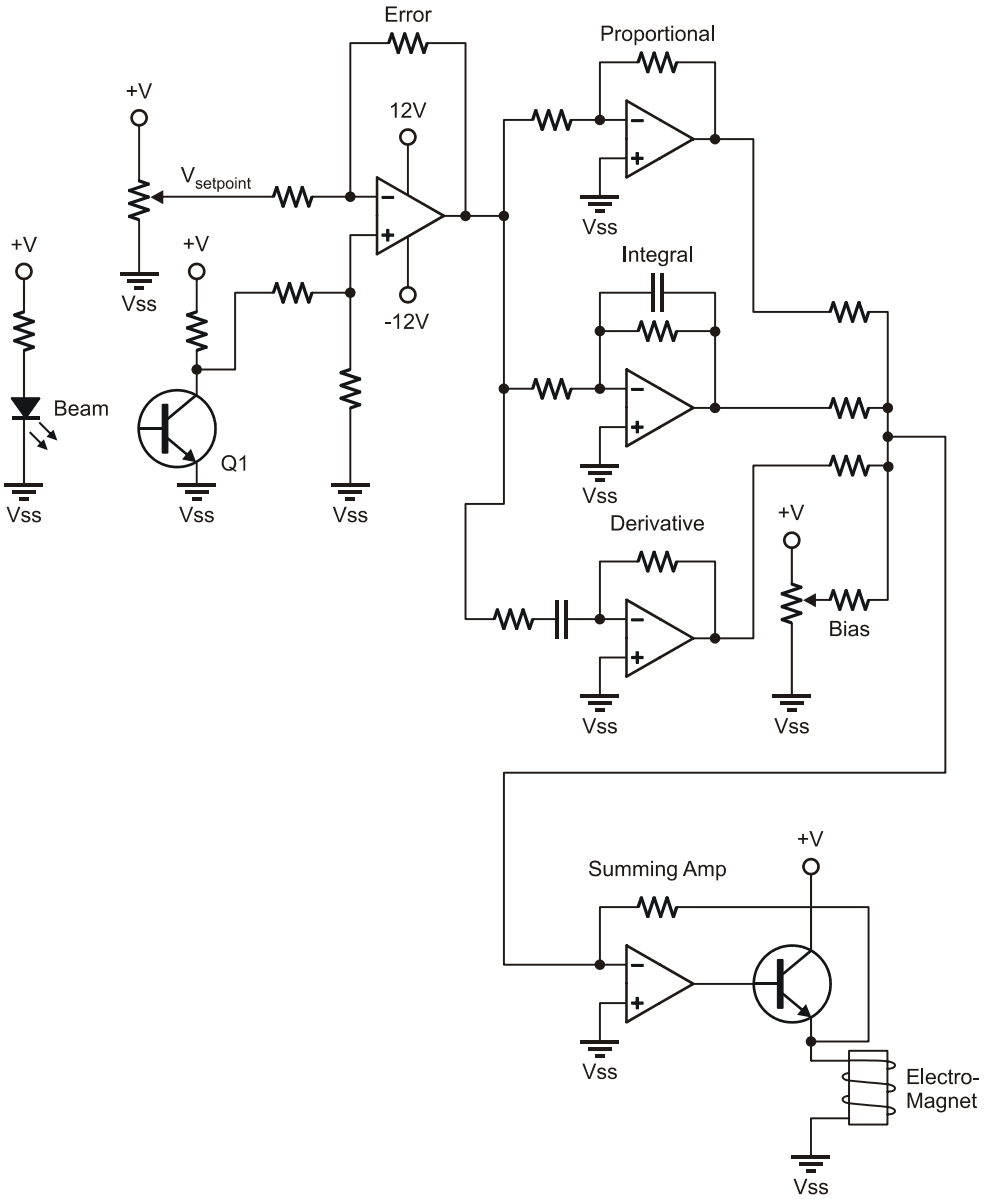


Figure 8-3
Floating Ball Project

Figure 8-4 Typical Op-Amp PID Control Circuit for the Floating Ball – *DO NOT BUILD*



By using a circuit similar to the one in Figure 8-4 to perform the measurements, calculations, and drive, the position of the ball is continuously evaluated and drive is updated nearly instantaneously. The difference between setpoint and actual voltages is determined (Error) and this signal is processed with three op-amp configurations to amplify the error (proportional), measure the error over time (integral) and the change in error in respect to time (derivative). The three signals and a bias voltage are added together (summing) to provide a drive voltage. The error instantaneously causes a change in the output. By adjusting the resistances for each op-amp stage, the gain, and consequently the contribution to the total output voltage, of the individual P, I and D blocks may be adjusted to achieve stable operation.

While newer advances in digital technology, such as Digital Signal Processors (DSP) can come very close to continuous measurement and control of the analog control circuit in Figure 8-4, programmable microcontrollers such as the BASIC Stamp are limited in the speed in which the value may be measured, calculations made, and action taken.

Consider the flowchart in Figure 8-5. Code must be processed at each step, which takes time. Sampling can only be performed at discrete intervals of time instead of continuously. The speed of the sampling and evaluation needed is very dependent on the dynamics of the system under control. Keeping a ball floating requires relatively fast sampling and control (around 30 updates per second). Other systems, such as our incubator, have lower demands.

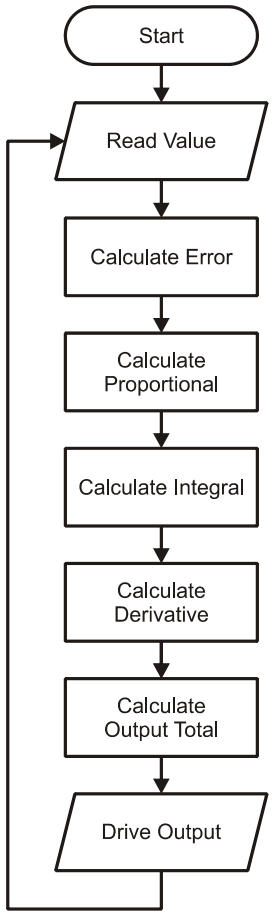


Figure 8-5
Programmatic PID Processing Flowchart

Because of the discrete sampling, the actual error and calculations will need to be performed as shown in Figure 8-6. The value is measured at time intervals and P, I and D calculations are based on those error samples. Since the signal value is only measured at discrete intervals, the proportional drive is only updated at each sample.

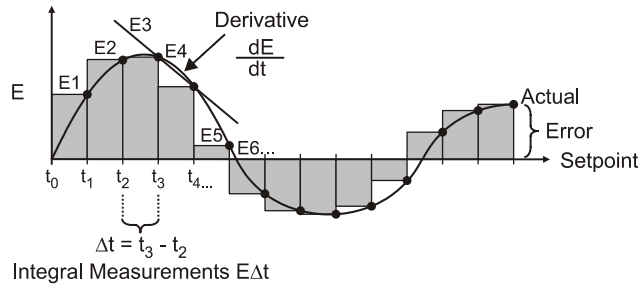


Figure 8-6
Discrete PID measurements

Derivate drive is based on the rate of change of the error – how quickly the error changed between samples:

$$\text{DERIVATIVE} = (E_4\text{-}E_3) / (t_4 – t_3) \quad = \Delta E/\Delta t$$

Using discrete PID measurements, the PID formula becomes

$$Co_{PID} = \text{Bias} + K_P E + K_I \Sigma E\Delta t + K_D \Delta E/\Delta t$$

Since we are on the subject of using light levels for PID control as used in the Floating Ball project, let's experiement with PID evaluation using light. In the next activity, we'll build a light-sensing system and use it to experiment with the effect of proportional (P), integral (I) and derivative (D) control, both singularly and in combination. The system consists of a photoresistor to sense the amount of light, and your hand, to either cast shade or allow more light onto the sensor. Your hand, then, creates disturbances to the system, and the photoresistor senses the system variable, in this case, light intensity. This system will demonstrate the PID evaluations of a system, but it is not actually representative of the response of a closed-loop system, as there is no electro-mechanical system in place to control the light level based on the PID evaluations.

Consider, though, if the light level was influenced by the amount of light coming in through a window, and we had a motor connected to windows blinds. As the light level increased during the day, the motor would close the blinds in an attempt to maintain light level at the setpoint. In such a system, the output drive values would directly control the motor. For a Parallax Standard Servo motor and the BS2, reasonable drive values would range 500 to 1000. However, for this activity, we are going to simply output values that are in the same range as those of the light sensor, about 0 to 2000 or so. Our output values will not be values that are needed for actual control.

Our main focus will be on watching how changes in the sensor cause changes in the output values, even though the output values are not in themselves meaningful in the physical world. Furthermore, the activity is set up so that as the photoresistor reading increases, the output drive will also increase. This is the opposite of our incubator model – as the temperature readings increase, the heater output must decrease to keep the temperature at the setpoint.

The materials needed for this activity are listed below.

ACTIVITY #1: BIAS DRIVE

Parts required:

(1) 10 kΩ Photoresistor
(1) 0.1 µF Capacitor
(1) 220 Ω Resistor

√ Add the light sensor circuit in Figure 8-7 to your board, leaving all the other components in place. The wiring diagram shows the recommended parts placement. We will be using the full heater circuit again shortly.

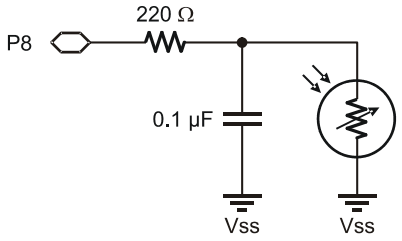
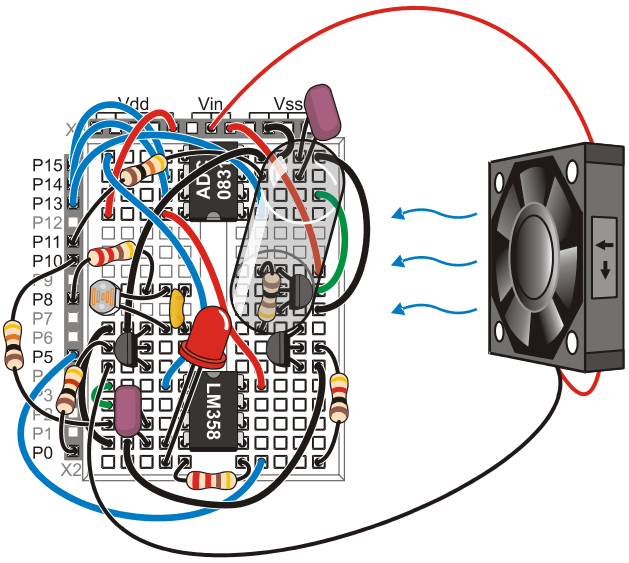


Figure 8-7
Light Sensor Schematic (left) and Wiring Diagram (below)

Leave the other parts on the board.



√ Enter, save and run the program PIDEval.bs2 and close the Debug Terminal.

```
' -----[ Title ]-------------------------------------------------------------
' Process Control - PIDEval.BS2
' Demonstrated P, I and D evaluations with StampPlot
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------
SetPoint    VAR     Word  ' Setpoint value
Photoval    VAR     Word  ' Value of photo RC
Error       VAR     Word  ' Caclulated Error
Prop        VAR     Word  ' Calculated proportional output
IntegSample VAR     Word  ' Sample Integral value (Edt)
Integ       VAR     Word  ' Calculated total integral value (SumEdt)
LastError   VAR     Word  ' Last error for deriv calcs
Deriv       VAR     Word  ' Calculated derivative output (dE/dt)
ErrorChange VAR     Word  ' Change in error for deriv calcs (dE)
Total       VAR     Word  ' Total output

SignBit     VAR     Bit   ' Holds sign for math
Counter     VAR     Byte  ' Count of samples
Samples     VAR     Byte  ' Number of samples per evaluation

Delay       CON     100   ' Delay to obtain approx times between evaluations
```

```
Photo PIN 8               ' Photo RC Network

' -----[ Initialization ]---------------------------------------------------
' ***** Reads values from StampPlot on reset
PAUSE 1000                ' Allow connection to stabilize

' Read time between evaluations (seconds * 4)
DEBUG CR,"!READ [(drpTime),*,4]",CR
DEBUGIN DEC Samples
PAUSE 50

' Read setpoint
DEBUG CR,"!READ (txtSetP)",CR
DEBUGIN DEC SetPoint
PAUSE 50

' -----[ Main Routine ]------------------------------------------------------

DO
 ' Adjust delay to approximate real time between evaluations
 ' Take samples and plot
 IF Samples > 2 THEN
    FOR Counter = 1 TO Samples
      GOSUB ReadPhoto
      GOSUB PlotData
      IF Samples > 4 THEN PAUSE Delay
    NEXT
  ELSE
      GOSUB ReadPhoto
      GOSUB PlotData
  ENDIF

  ' Evaluate and plot
  GOSUB CalcError
  GOSUB CalcProp
  GOSUB MarkProp
  ' GOSUB CalcInteg
  ' GOSUB MarkInteg
  ' GOSUB CalcDeriv
  ' GOSUB MarkDeriv
  GOSUB CalcTotal
  GOSUB MarkTotal
LOOP

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

ReadPhoto:
  HIGH Photo                    ' Charge photoresistor's RC network Cap
  PAUSE 10                      ' Allow 10 milliseconds to charge fully
  RCTIME Photo, 1, PhotoVal     ' Measure discharge time thru photoresistor
RETURN
```

```
CalcError:                          ' Calculate error
  Error = PhotoVal - SetPoint
RETURN

CalcProp:                           ' Calculate proportional output
  Prop = Error
RETURN

CalcInteg:                          ' Calculate integral output
  IntegSample = Error * Samples     ' Sample is error x time (samples) Edt
  SignBit = IntegSample.BIT15       ' save sign for math
  IntegSample = ABS(IntegSample)/4          ' approx 1/4 sec per sample -
                                            ' convert to seconds

  IF SignBit = 1 THEN IntegSample = IntegSample * -1  ' re-apply sign

  Integ = Integ + IntegSample               ' Sum sample to current value
  SignBit = Integ.BIT15                     ' Save sign of integ, 1 = negative
  Integ = ABS(Integ) MAX 25000              ' Limit maximum to 25000
  IF SignBit = 1 THEN Integ = Integ * -1    ' Re-apply sign
RETURN

CalcDeriv:                                  ' Calculate derivative output
  DEBUG "!O txtLE=", SDEC LastError,CR      ' Update SP with last error
  ErrorChange = Error - LastError           ' Calculate dE
  SignBit = ErrorChange.BIT15               ' Save sign, 1 = negative
  Deriv = ABS(ErrorChange)/Samples * 4      ' Divide by dt
  IF SignBit = 1 THEN Deriv = Deriv * -1    ' Re-apply sign
  LastError = Error                         ' Save Last error for this eval
RETURN

CalcTotal:                                  ' Calculate total output
  Total = Prop + Integ + Deriv              ' Sum all for total
  SignBit = Total.BIT15                     ' Save sign of total, 1 = negative
  Total = ABS(Total) MAX 25000              ' Limit maximum to 30000
  IF SignBit = 1 THEN Total = Total * -1    ' Re-apply sign
RETURN

MarkProp:      ' Mark propotional evaluation as line from setpoint to actual
  DEBUG "!DMOD 9",CR,
        "!LINE (PTIME),(AINVAL0),(PTIME),(AINVAL1),(Green)",CR,
        "!DMOD 13",CR
RETURN

MarkInteg:     ' Mark integral evaluation as rectangles
  DEBUG "!DMOD 9",CR,
        "!FREC (DATAVAL1),(AINVAL0),(PTIME),(AINVAL1),(Yellow)",CR,
        "!RECT ,,,,(Green)",CR,
        "!DMOD 13",CR
RETURN
```

```
MarkDeriv:     ' Mark derivative evaluations as circles and lines
  DEBUG "!FCIR (PTIME),(AINVAL1),0.3A,(Blue)",CR,
        "!DMOD 9",CR,
        "^DWTH 3",CR,
        "!LINE (DATAVAL1),(DATAVAL4),(PTIME),(AINVAL1),(Green)",CR,
        "!SETD 4,(AINVAL1)",CR,
        "!DMOD 13",CR
RETURN

MarkTotal:     ' Plot outputs and update
DEBUG   "!O txtE=", SDEC Error,CR,
        "!O txtActual=", DEC PhotoVal,CR,
        "!O txtProp=", SDEC Prop,CR,
        "!O txtEdt=", SDEC IntegSample,CR,
        "!O txtsumEdT=", SDEC Integ,CR,
        "!O txtdE=", SDEC ErrorChange,CR,
        "!O txtdEdt=", SDEC Deriv,CR,
        "!O txtTotal=", SDEC Total,CR,
        "!O plot2.draw=line (DATAVAL1),0,(PTIME),0,(RED)",CR,
        "!O Plot2.Draw=LINE (DATAVAL1),(DATAVAL2),(DATAVAL1),",SDEC
Total,",(Black)",CR,
        "!O Plot2.Draw=LINE (DATAVAL1),",SDEC Total,",(PTIME),",SDEC
Total,",(Black)",CR,
        "!SETD 1,(PTIME)",CR,
        "!SETD 2,", SDEC Total,CR,
        "!O butLog.Run",CR
RETURN

PlotData:     ' Plot photoresistor value

  DEBUG "^AWTH 2",CR,
        "!ACHN 0,", DEC Setpoint,",(RED)",CR,
        "!ACHN 1,", DEC PhotoVal,",(BLUE)",CR,
        "!STAT Actual=", DEC PhotoVal,CR
RETURN
```

√ Run StampPlot macro sic_pc_pid_eval.spm.

Before connecting, we'll discuss some of the features of this StampPlot interface, shown in Figure 8-8. The uppermost plot shows the values read from the photoresistor, plotted with a blue line. The plot below shows the output drive, plotted in black.

The photoresistor plot has these characteristics:

- The default range is 0 to 2000.
- The Setpoint is plotted in red.

- Vertical green lines show the point at which data is evaluated for PID calculations.

The Output Drive Plot has these characteristics:

- The default range is from -2000 to 2000, making the scale half that of the upper plot.
- A red line shows the 0 value, not the setpoint.

√ Shade the sensor with your hand to a comfortable amount where you can quickly light or darken the sensor with hand movements.
√ If there is considerable difference between the setpoint (red line) and your value, disconnect on StampPlot, change the Setpoint value on the plot and reset your BASIC Stamp.

Testing Proportional Evaluation

√ In the "Evaluate and plot" section of the Main Routine, ensure that the integral and derivative subroutines are commented out as in the following:

```
' Evaluate and plot
  GOSUB CalcError
  GOSUB CalcProp
  GOSUB MarkProp
  ' GOSUB CalcInteg
  ' GOSUB MarkInteg
  ' GOSUB CalcDeriv
  ' GOSUB MarkDeriv
  GOSUB CalcTotal
  GOSUB MarkTotal
```

√ Ensure that "Evaluation Time" is selected to 4.
√ Press Reset on the Board of Education and reset the plot.
√ Lighten and darken the sensor and note how often the data is sampled for calculations (green vertical lines).
√ Note that the total drive is updated below.
√ Note that green vertical lines in the top plot denote when calculated samples are taken. At each sample, the output, due to proportional control only, is updated on the lower plot.
√ After about 40 seconds of data, change the sample time to 0.5 seconds and reset the BASIC Stamp (not the plot).

Figure 8-8 Proportional Only Evaluation of Light Level

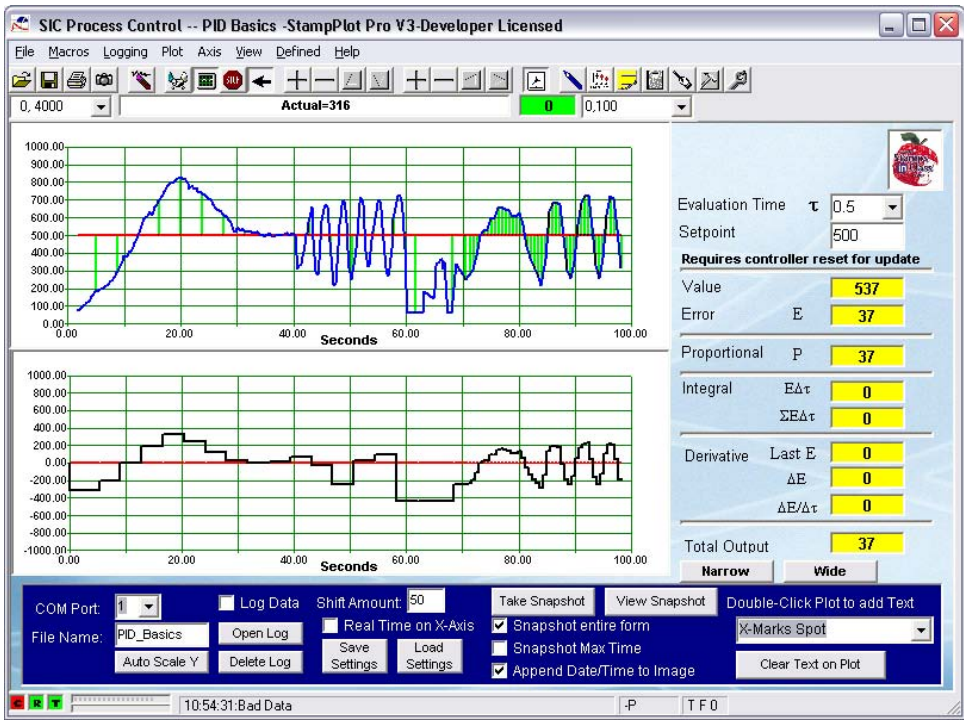


Figure 8-8 is a sample of the plot. Note that data is sampled and plotted much more often than error is calculated and total output is calculated and plotted. Time between samples is approximate. At each sample time, the error is calculated, and the proportional amount is plotted on the bottom plot. This program and macro have been designed so that the error is the actual value minus the setpoint. The greater the error, the greater the output value being plotted. In this case, output is proportional to the error.

Note that with slow sampling, the output is only a rough approximation of the actual curve. The rapid changes in signal are barely noticeable in the output. As the sampling rate increases, the output matches the error much closer.

The code reveals how the error is calculated and the proportional output calculated:

```
CalcError:
    Error = PhotoVal - SetPoint
RETURN

CalcProp:
  Prop = Error
RETURN
```

Error in Error? In the classical PID formula, as illustrated in Figure 8-1, error is calculated as:

Error = Setpoint – MeasuredValue ' Error per Figure 8-1

However, for this program we are using the expression

Error = Photoval – Setpoint ' Error used in PIDEval.bs2

In effect, errors that would be negative in the classical formula, are positive in our PIDEval program. This was done only for this activity, in order to show the output drive increasing as the photoresistor readings increase. That is, for photoresistor values greater than the setpoint, we would like to see a positive output drive, for which we need a positive error.

In the next activity, using the incubator model, we will again use the classical formula, which will produce a negative error when the temperature is greater than the setpoint.

The total output is the sum of the P, I, and D outputs, but in this case, Integral and Derivative will be zero since those calculations are not being performed. Since the limit of the BASIC Stamp calculations is +/− 32,000, code has been added to ensure the maximum does not exceed 25,000. The BASIC Stamp can't perform all math operations on negative numbers and provide results we would recognize. When working with division, MIN, MAX, and some other operations, they need to be performed on positive values only. To do this, the sign of the value is saved (Bit 15 = 1 if negative), the absolute value is used, and the sign re-applied.

```
CalcTotal:
  Total = Prop + Integ + Deriv
  SignBit = Total.BIT15
  Total = ABS(Total) MAX 25000
  IF signBit = 1 THEN Total = Total * -1
RETURN
```

Currently, the proportional control is simply equal to the error: P = E. In actuality, the error is multiplied by some constant, gain (K_P), to adjust how much drive is applied for a set amount of error: $P = K_P E$

- √ In the **`CalcProp`** subroutine, multiply the Error by 4 and retest.
- √ Compare the output plot to your previous for the same amounts of error. Notice the four-fold increase in output action taken for the same error as before.

In this activity, though there is an output based on error, there exists no feedback that would control the window blinds in an attempt to maintain the light level. But you can imagine that based on the amount of error, and therefore the proportional output, in an actual system, the higher the magnitude of the error, the more control action that would be taken.

Testing Integral Evaluation ΣEΔt

In this section we will test the integral control based on error over time. Since the BASIC Stamp does not have continuous internal timers, all times for samples and evaluations are approximations.

- √ In the code, comment out the proportional subroutines and uncomment out the integral ones:

```
' GOSUB CalcProp
' GOSUB MarkProp
GOSUB CalcInteg
GOSUB MarkInteg
```

- √ Download the code and close the Debug Terminal.
- √ Set the time between samples to 4 seconds.
- √ Set the control plot range to +/− 32,000 by clicking the "Widen" button several times.
- √ Increase the time of the plot to 200 seconds.
- √ Connect on StampPlot.

Set 1

- √ Reset the plot, reset the BASIC Stamp, and slowly darken and lighten the sensor over 30 seconds or so.
- √ Partially shade the sensor to a light level equal to the setpoint, and hold for 10 seconds.

Set 2

- √ Allow full light to fall on the sensor for 30 seconds.

Set 3

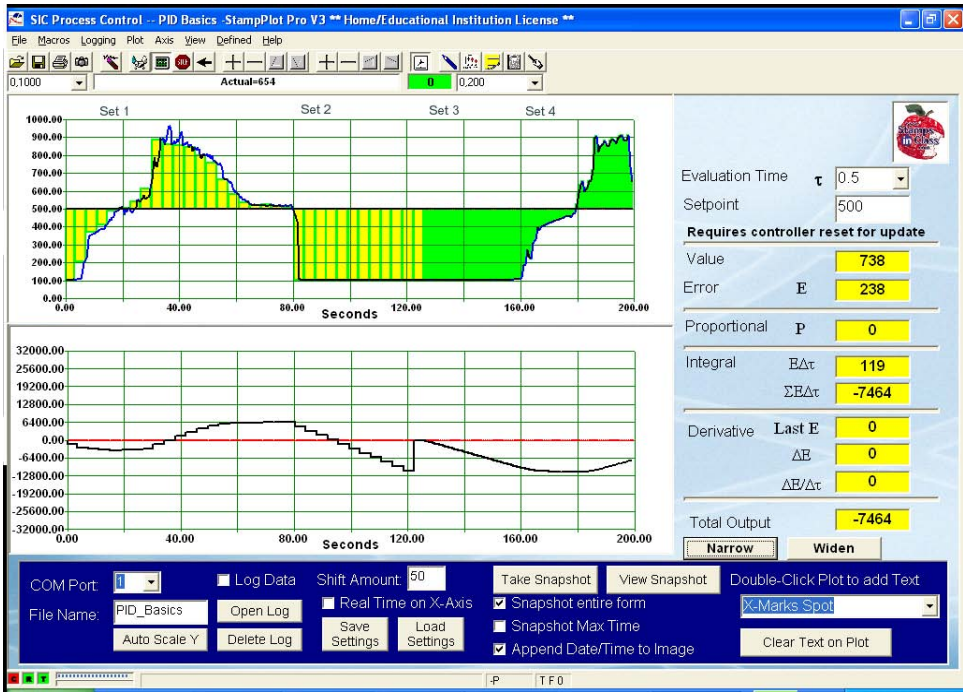
- √ Change the time between samples to 1 second and reset the BASIC Stamp. NOTE: A reset will cause the integral value to be reset to zero. This is expected.
- √ Continue to allow full light to fall on the sensor.

Set 4

- √ Gradually increase and decrease the light levels.
- √ Stop plotting (F6).

Figure 8-9 is a plot of our results for discussion.

Figure 8-9 Integral Only Evaluation of Light Level



Set 1
At reset, the amount of integral control is zero because nothing has been added yet. With each sample, the amount of error and change in time is calculated and added to the total. Large error with respect to time causes large incremental changes in output. As long as the error is negative, the output becomes increasingly negative. It is not until the error goes positive that the output will be reduced. With sufficient positive error, the output will become zero, and then become increasingly positive. Note what area is actually used at each sample. Sometimes it is very representative of the curve, and sometimes not, based on our sampling rate.

Set 2
With the value at the setpoint, the integral value remains the same since nothing is added or subtracted.

Sets 3 and 4
In Set 3, full light was allowed to fall for a while with 4-second control sampling. Note the slope of the integral control building. In Set 4, when the time was changed to 1 second, the slope of the output was relatively the same but the output change was smoother. Integral is $E\Delta t$. Over all, the error and the total time stayed the same, and the integral value grew at the same rate. The only difference was the total time was calculated and added much more frequently in Set 4.

For example, with 4-second evaluation, with an error of −400, each integral sample added −1600 or (−400)(4 seconds). Over 12 seconds, the total would be (3 samples)(−1600) = −4800. With 1-second evaluations, each sample would be (−400)(1 second) = −400. Over 12 seconds, the total or sum would be (−400)(12) = −4800.

Looking at the code for the integral control:

```
CalcInteg:                          ' Calculate integral output
  IntegSample = Error * Samples     ' Sample is error x time (samples) Edt
  SignBit = IntegSample.BIT15       ' save sign for math
  IntegSample = ABS(IntegSample)/4          ' approx 1/4 sec per sample -
                                            '  convert to seconds

  IF SignBit = 1 THEN IntegSample = IntegSample * -1  ' re-apply sign

  Integ = Integ + IntegSample       ' Sum sample to current value
  SignBit = Integ.BIT15             ' Save sign of integ, 1 = negative
  Integ = ABS(Integ) MAX 25000              ' Limit maximum to 25000
  IF SignBit = 1 THEN Integ = Integ * -1    ' Re-apply sign
RETURN
```

Since we want to see data between evaluations, 1 second's worth of data collection takes 4 samples (allowing us to see the data between evaluations). So that the integral value for the sample is 4 times too large and must be divided. Again, signed math is undesirable, so the sign is saved and re-applied. The sample value is then added to the integral total while ensuring it does not become too large.

Again, just as in proportional control, the output would be driving the system, but in this case, it is based on the magnitude and duration of the error. The longer that error exists, the greater the output will become. Large amounts of error for extended periods of time can cause integral values to reach very high values. This 'windup', problems associated with it, and how integral control can be used in the operation of the system will be explored more when the incubator is used.

As you can see, integral output control can build very quickly and easily masks other control action. The amount of integral output may be reduced by applying a very small gain (K_I) such as 0.1 to the calculation.

√ In the BASIC Stamp code, divide the Integral Sample by 10 for a gain of 0.1 and retest.

```
IntegSample = ABS(IntegSample)/4 / 10
```

Testing Derivative Evaluation: ΔE/Δt

In this section we will test the derivative evaluation based on change in error with respect to time.

√ In the code, comment out the Integral subroutines and uncomment the Derivative ones:

```
' GOSUB CalcInteg
' GOSUB MarkInteg
GOSUB CalcDeriv
GOSUB MarkDeriv
```

√ Download the code and close the Debug Terminal.
√ Set the time between samples to 2 seconds.
√ Set the control plot (lower) range to +/− 1000 through the use of the "Narrow" button.
√ Connect on StampPlot.

Set 1

- √ Reset the plot, reset the BASIC Stamp, and slowly darken and lighten the sensor over 30 seconds or so.

Set 2

- √ Change the light level fairly rapidly for about 10 seconds.

Set 3

- √ Hold the light level constant above the setpoint.

Set 4

- √ Change the evaluation sample time to 0.5 seconds and reset the BASIC Stamp (not the plot).
- √ Slowly darken and lighten the sensor over 20 seconds.

Set 5

- √ Quickly lighten and darken the sensor over about 10 seconds.
- √ Close the connection (F6).

Figure 8-10 is a plot of our results for discussion.

Set 1

As the signal gradually rises, the slope of the error (line connecting dots) is positive, and the output is positive. As the signal gradually falls, the slope of the error is negative and so is the output. It does not matter if the actual value is above or below the setpoint, it is simply a matter of the direction of change.

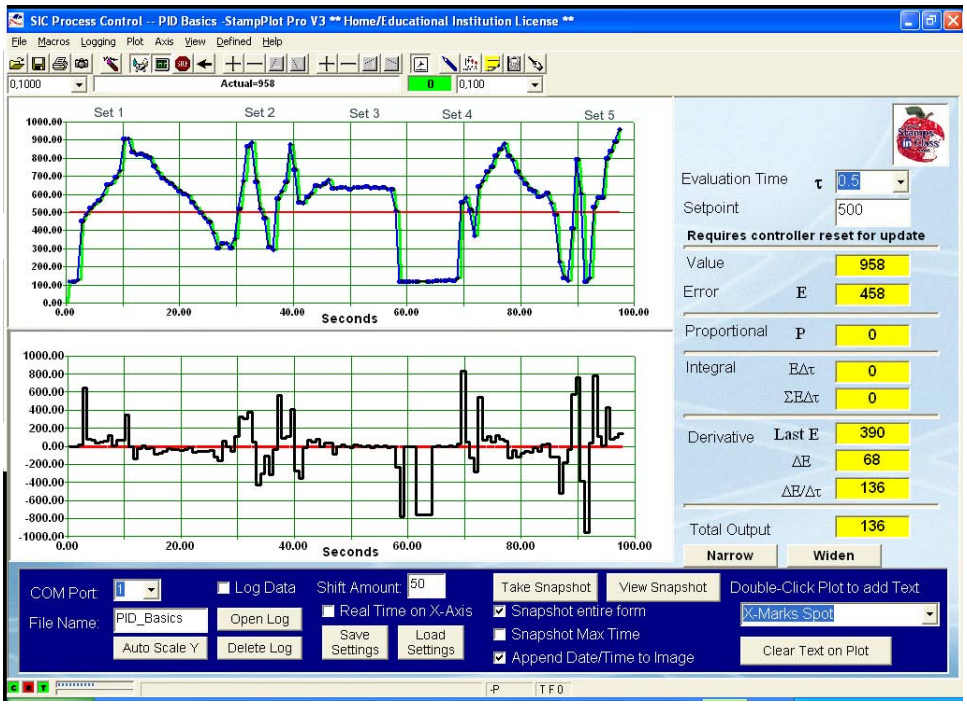
Set2

As the signal changes faster, the output increases in magnitude. The error changed more over the same sample time though the level of the signal was relatively the same.

Set 3

With the signal relatively constant, though well above the setpoint, the output is roughly zero (as well as we could hold our hand stable). Even though there existed an error, the error was not changing, therefore no output.

Figure 8-10 Derivative Evaluation Using Light



Set 4
With a decrease in time between evaluation sampling, roughly the same curves increased the output. With closer sampling, the maximum change of the signal is better evaluated. Note the negative output spike at 60 seconds. When the controller was reset, the value of the last error was reset to zero. When a reading was performed, it was calculated as a sudden change in error.

Set 5
With the shorter time and higher rates of error change, the output was significantly higher.

Note the measured readings: With a current error of 458, and a last error of 390, this produced a change in error (ΔE) of 68. With a change in time (Δt) of 0.5 seconds, this resulted in an output (ΔE/Δt) of 68/0.5 or 136.

In an actual system, just as with the automobile example, the derivative output changes in a direction and magnitude to counter the error. Acting on the window blinds, as the sensor suddenly lightened, the output would quickly move the motor with a motion relative to the change.

```
CalcDeriv:                                ' Calculate derivative output
  DEBUG "!O txtLE=", SDEC LastError,CR    ' Update SP with last error
  ErrorChange= Error - LastError          ' Calculate dE
  SignBit = ErrorChange.BIT15             ' Save sign, 1 = negative
  Deriv = ABS(ErrorChange)/Samples * 4    ' Divide by dt
  IF SignBit = 1 THEN Deriv = Deriv * -1 ' Re-apply sign
  LastError = Error                       ' Save Lat error for this eval
RETURN
```

Just as with proportional and integral, we can adjust the amount of force applied by derivative by setting the gain (K_D). Too much gain can cause erratic control of a system, though.

- √ Modify the program to multiply **`Deriv`** by a gain of 10, and set a sample time of 1.0 second.
- √ Try to hold your hand so that the output does not exceed a value of +/− 400 for 20 seconds.
- √ Even leaving the sensor totally uncovered, the measurement fluctuates due to noise. Note the amount of output change simply based on the noise.

Adding Bias

In the examples up until now, the output swung both positive and negative for control of the system. In most cases, the output will only be in one direction. Consider the car example. On a level highway, the pedal will be in a position to supply sufficient energy to keep the car at the setpoint speed. The pedal is adjusted up or down from this center position to control the speed. In the floating ball example, the electro-magnet will have a set amount of drive (preferable 50%) when the ball is correctly in the beam. As it rises, drive will cut back; as it falls, drive will increase.

Bias is used to provide a midpoint drive that, with good engineering, should maintain a stable system at the setpoint under normal conditions. We will test adding bias to our calculations and controlling the system with that as the midpoint.

√ Add a bias to your output by adding 500 to the total drive calculated in the **`CalcTotal`** subroutine.

```
Total = 500 + Prop + Integ + Deriv
```

Modify the code so that only proportional calculations are made and plotted.

√ Download the program, use 1.0 seconds for evaluations.
√ Hold the level at the setpoint.
√ Cycle the light level up and down several times while trying not to exceed the range of 0 to 1000 on the output. Note that when operating at the setpoint, the output is 500. Since there is no error, the only output is based on bias.
√ Have the BASIC Stamp calculate all 3 evaluations, but not plot mark any on the upper plot. Leave integral with a gain of 0.1 (/10), but remove the gains from proportional and derivative (no gain essentially means a unity gain, or gain of 1).

```
GOSUB CalcProp
' GOSUB MarkProp
GOSUB CalcInteg
' GOSUB MarkInteg
GOSUB CalcDeriv
' GOSUB MarkDeriv
```

√ Experiment with light levels to try and keep the output at or near 500.
√ Identify the different types of control action illustrated in the output plot.

Challenge 8-1: Predict PID Outputs

Part A

Figure 8-11 on page 273 is a plot of light levels and 3 plots for the P, I and D outputs. With a bias of 0, predict the output for each evaluation individually. Assume gains of 1 for all, and a 5 second evaluation time. The first set is done for you.

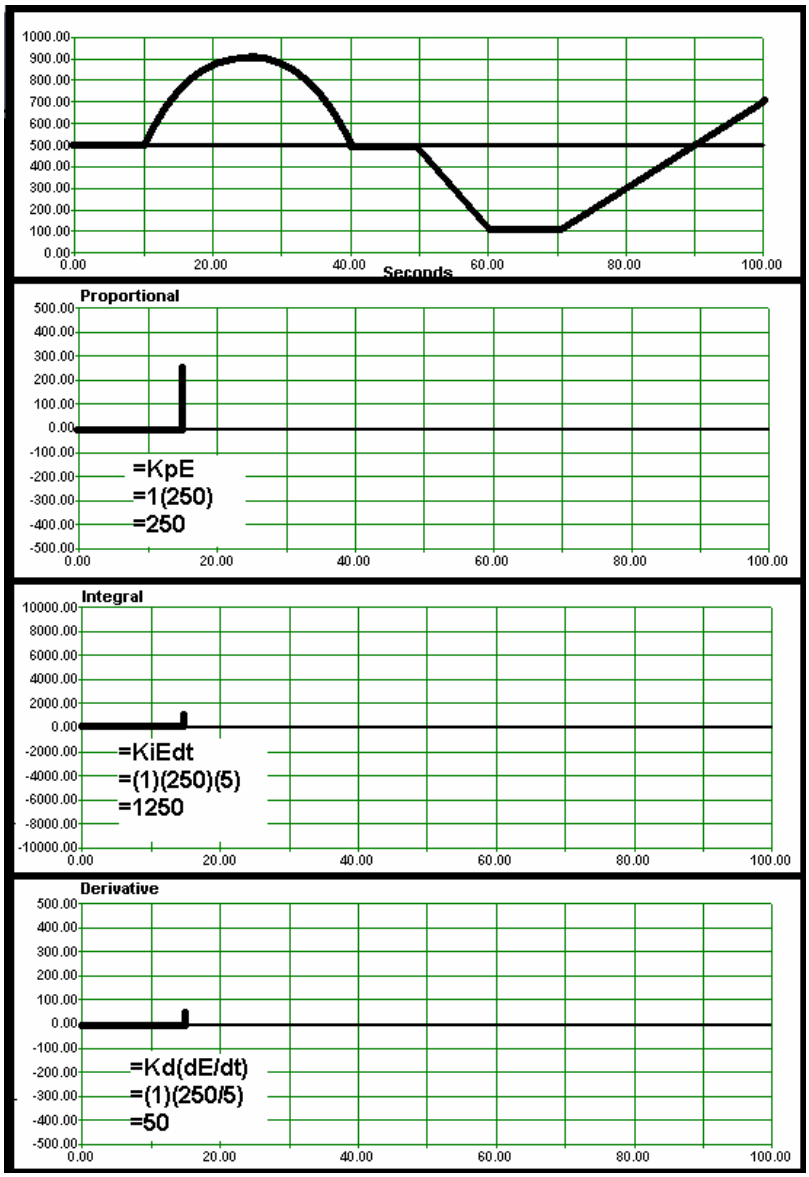
Part B

Below is data representing the systems conditions and data for PID evaluation. Calculate the output for each evaluation. The 1st 3 lines are done for you.

Table 8-1: PID Evaulation Activity
Given Kp = 2 Ki = 0.01 Kd = 4 Setpoint = 500

Actual	∆t	E	∆E	P(KpE)	E∆t	I(KiΣE∆t)	D(Kd∆E/∆t)
500	1.0	0	0	0	0	0	0
510	1.0	10	10	20	10	.1	40
490	0.5	-10	-20	-20	-5	0.05	-160
480	0.5						
470	2.0						
520	1.0						
500	0.5						
500	1.0						

Figure 8-11 PID Evaluation Activity



ACTIVITY #2: BIAS AND SYSTEM RESPONSE

Activity #1 was good to demonstrate the PID evaluations of a system, but it is not representative of the response of a closed-loop system. There was no electro-mechanical system in place to control the light level based on the PID evaluations. Consider though if we had a servo connected to blinds. As the light level increased, the servo would position the blinds in an attempt to maintain light level at the setpoint, perhaps in growing some rare plant that required a certain amount of light to flourish.

The standard hobby servo sold by Parallax requires values from the BS2 from 500 to 1000 for control. Sending the servo a pulse width value between 500 and 1000, the servo is controlled to a position between roughly 0 and 90 degrees. If the servo was coupled to a light shade, the shades could be adjusted to be fully open or fully closed. Biasing would be set so that the light level coming through the blinds was at a medium level under normal conditions (midday sun).

Now consider the value of light error in our experiments. An error of 200, 500 or a 1000 in some cases was very likely. If it got lighter outside (early morning sun shining through), the error and thus the control output would result in the servo adjusting the blinds to shut some. Conversely, darkening (cloudy day) would cause the blinds to open some.

Our output was hundreds or thousands, positive and negative. The servo requires a range of 500 to 1000 for full control. There is a disparity in the values we output and the values needed for actual control. Of course, with a little math we can span and offset the output value to match the controls input value. But it is much easier to discuss systems in terms of percentage so that no matter the ranges involved, the system is measuring, calculating, and driving in the common units of percent.

Under normal conditions, the servo will be controlled over a range of 0% for fully open to 100% fully closed. The light level will be from 0% for darkest to 100% for lightest. Under normal condition, the desired light level of 50% is achieved when the servo is driven at 50%. If the light level increases, say 10%, the servo will have its output increased by 10% to darken the area. If the light level reaches 100%, the servo will be at 100% to be fully closed.

We begin discussing the entire system in terms of percentages, and it makes it much easier to discuss in generalities and in actual system response. %Error, %Total Drive,

%Drive Bias, %Drive Proportional and so on. The new equation for the systems operation becomes:

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

As discussed, a system in equilibrium is where the energy gains in that system equals the energy losses. When a system is designed, the engineers will have anticipated typical losses on the system. The bias drive is the drive used to compensate for normal losses. In terms of control, you've already performed this in earlier chapters. The amount of PWM drive was adjusted until the incubator was at or near the setpoint of 101.5 °F. This would be the bias drive for the incubator.

Bias drive provides a best-case starting point for control of a system. When a system, such as an incubator, is engineered, it is possible to design it so that a 50% drive to the heating element will control it very near the setpoint under normal conditions. This design provides for up-to 50% of the drive to be removed or up-to an additional 50% to be added for control of the system. Consider an outdoor incubator. If 50% drive on the heaters is good for a nice spring day, an additional 50% drive can help warm it on very cold winter days, or cutting back the drive by 50% on very hot summer days will help keep it at the setpoint.

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 101.5 °F under our average laboratory conditions. Unfortunately, our system is not well engineered. Being a non-insulated polystyrene test tube with a small resistor for a heater, chances are that exactly 50% drive will not provide the exact amount of energy needed to meet our desired setpoint.

Note that this bias temperature may fluctuate due to room conditions, and that depending on conditions, your results may vary and 50% may be well above 101.5 °F.

$Co_{PID} = B$
$\%Drive_{TOTAL} = \%Drive_{BIAS}$

Parts required:

Same as Activity #1

√ Enter, save and run IncubatorPID-SP.bs2.

```
' -----[ Title ]-----------------------------------------------------------
' Process control - IncubatorPID-SP.bs2
' Controls an incubator using PID control.
' StampPlot is used to perform calculations
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------------

ADC_DataIn  VAR Byte       ' Analog to Digital Converter data
TempF       VAR Word

LED         PIN 0          ' LED output pin
ADC_CS      PIN 13         ' ADC Chip Select pin
ADC_Clk     PIN 14         ' ADC Clock pin
ADC_Dout    PIN 15         ' ADC Data output
ADC_Vminus  PIN 11         ' ADC Offset Value
ADC_Vref    PIN 10         ' ADC Span reference

Heater      PIN 5          ' Incubator Heater
Fan         PIN 0          ' Incubator Fan


DriveTime    VAR Byte      ' number of seconds (x4 for PWM) to drive
PWMVal       VAR Byte      ' PWM Value to drive heater
x            VAR Byte      ' Working variable
Offset       VAR Byte      ' ADC Offset value (tenths)
Span         VAR Byte      ' ADC SPan value (tenths)

' -----[ Initialization ]--------------------------------------------------
  LOW Fan
  PAUSE 1000               ' Connection Stabilization
  GOSUB ReadSP             ' Read values from StampPlot

' -----[ Main Routine ]-----------------------------------------------------
DO

  GOSUB ReadADC
  GOSUB UpdateSP
  GOSUB GetSPDrive
  GOSUB ControlIncubator
LOOP

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

' **** Set ADC Span and offset and read ADC value
READADC:
  PWM ADC_Vminus,Offset * 255/500,100
  PWM ADC_Vref,Span * 255/500,100
  LOW ADC_CS
  SHIFTIN ADC_Dout,ADC_Clk, MSBPOST,[ADC_DataIn\9]
```

```
  HIGH ADC_CS           '
RETURN

'  **** Read last total drive reading from StampPlot
'       and condition of cooling checkbox
'       Call macro routine to mark plot
GetSPDrive:
  DEBUG "!READ [[(txtDtot),*,255],/,100]",CR
  DEBUGIN DEC PWMVal
  PAUSE 50
  DEBUG "!READ (chkCool)",CR
  DEBUGIN DEC Fan
  PAUSE 50
  DEBUG "!MACR .PlotData",CR
RETURN

' **** Send new value to StampPlot.
UpdateSP:
  DEBUG CR,IBIN Fan,CR
  DEBUG DEC ADC_DataIn,CR
RETURN

' **** Drive PWM for amount of drive time in 250mSec intervals
'      Drive only if fan is off
ControlIncubator:
  IF FAN = 1 THEN PWMVal = 0
  FOR x = 1 TO DriveTime * 4
    PWM Heater,PWMVal,250
  NEXT
RETURN

' **** Read Configuration information from StampPlot
ReadSP:
  PAUSE 50
  DEBUG CR,"!Read (txtLower)",CR               ' Read Lower Temp
  DEBUGIN DEC Offset
  PAUSE 50
  DEBUG "!Read [(txtUpper),-,(txtLower)]",CR   'Read Span
  DEBUGIN DEC Span
  PAUSE 50
  DEBUG "!Read (txtTime)",CR                   ' time x 4 for PWM
  DEBUGIN DEC DriveTime
  PAUSE 50
RETURN
```

√ Open StampPlot macro sic_pc_pid_sp_calc.spm.

Before connecting, we will discuss some features of this interface, shown in Figure 8-12.

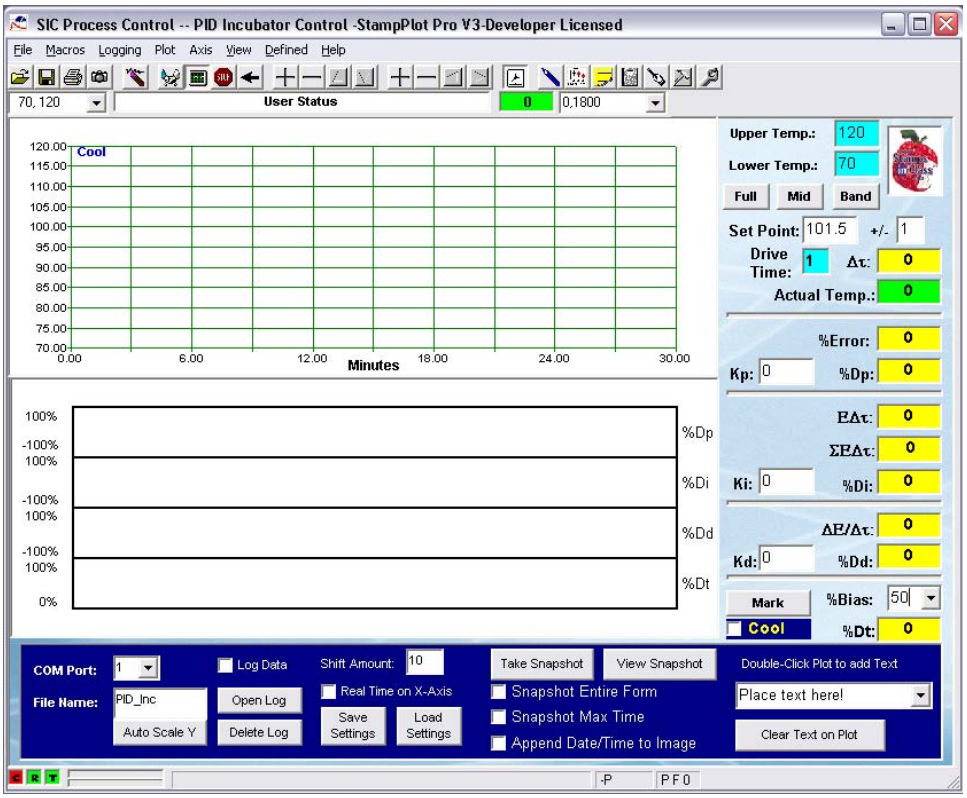
- Upper and Lower Temp.: Sets the upper and lower temperature for reading by the ADC. If this setting changed, you must reset your BASIC Stamp.
- Full, Mid, Band: Sets the span of the plot to the temperature limits, mid range for the band selected, or the control band only.
- Setpoint: Sets the temperature setpoint of the system.
- +/− : Sets the control band, such as a setpoint of 100 °F +/− 1 = 99 to 101.
- * Drive Time: Sets the seconds to apply heating drive continuously before updating.
- Δt: Actual time between samples.
- Actual Temp: Displays the current temperature of the system.
- %Error: Displays the %Error between the setpoint and actual temperatures. 200% is the maximum the StampPlot macro will calculate.
- Kp, Ki, Kd: Sets the gain constants for control.
- EΔt, ΣEΔt and ΔE/Δt: Displays values used for integral and derivative controls.
- %Dp, %Di, %Dd: Displays the %Drive from each of the control actions.
- %Bias: Sets the bias drive of the system.
- %Dt: Percent of total drive, 0 − 100.
- Cool: Energizes the fan, de-energizes the heater.
- Mark: Marks the current control settings on the plot.
- The lower graphs will plot the %Drives for P, I, D and total drive.

When plotting begins, the Message window should open, listing the current incubator settings and conditions. Data will also be logged to message and data files when logging is enabled. Let's try it now.

√ Connect on StampPlot and plot.
√ Verify that the Drive Control Setting match the following:

Upper Temp:	120
Lower Temp:	70
Setpoint:	101.5
Band (+/−):	1
Time:	1
Kp, Ki, Kd:	0
%Bias:	50

Figure 8-12 StampPlot PID Interface



- √ Allow the temperature to stabilize.
- √ Record your 50% bias temperature: _____
- √ Once the Temperature is stable, adjust the %Bias setting to bring the temperature closer to the setpoint.
- √ Click "Mark". This will mark the current drive settings in blue at the top of the plot and the current values of drive on the lower plots.
- √ Create a disturbance by checking "Cool". This will energize the fan and turn off the heater. For uniformity in disturbances, the cooling should be turned off once the temperature drops 5 °F.
- √ Allow the temperature to return to a stable temperature.

Analyzing our run in Figure 8-13, we see that it required about 5 minutes to stabilize at the 50% bias temperature of around 98 °F. Changing the %Bias to 60% and then to 55%, the temperature slowly drifted and stabilized closer to the setpoint.

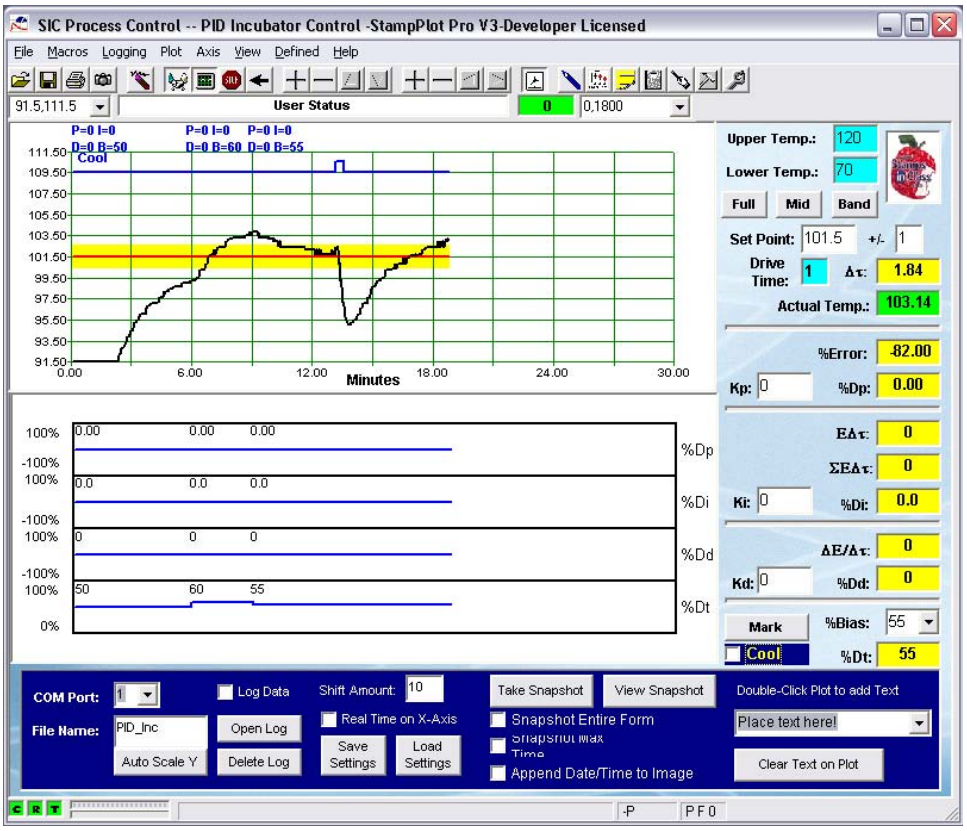
Is this system acting as a closed-loop? Not really. Just as in Chapter 6, we are manually manipulating the drive to control the system. As seen from the bottom graphs, the only time that the %Dt (Drive Total) changes is when we adjust the bias. Following a disturbance at 13 minutes, the system required approximately 7 minutes to recover and stabilize once again at the setpoint, though slightly higher - the room's winter heater may have raised temperature in the room, slightly changing conditions.

At 55% drive, the energy added by the heater equals the amount of energy lost at the setpoint to maintain the system stable. But, if the air around the incubator becomes cooler, greater energy escapes resulting in a lower stabilized temperature.

Program Discussion

The program IncubatorPID-SP.bs2 and macro sic_pc_pid_sp_calc.spm work together to control the drive to the heater. Upon BASIC Stamp reset, the temperaure limits for monitoring and the drive time are read. With each loop, the raw data is sent to StampPlot and the PID calculations and plotting are performed. The BASIC Stamp reads the total drive percent and drives the heater based upon that value. StampPlot can perform floating point math and handle very large numbers. All integral and derivative calculations are based on the real time that they arrived. One drawback is that it takes time to send data from the BASIC Stamp, allow StampPlot to perform calculations, and accept the returning data. With a drive time of 1 second, typical times of Δt were 1.75 to 2.10 seconds, and may be slower depending on the speed of your computer. Also, the incubator will not work unless StampPlot is running.

Figure 8-13 Adjusting Bias Drive



8

ACTIVITY #3: PROPORTIONAL CONTROL AT BIAS TEMPERATURE

Parts required:

Same as Activity #1

$$Co_{PID} = B + (K_P E)$$
$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%\ Drive_{PROP}$$

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 proportional drive. Error is the difference between the value we want the system to be and its measured value.

$$Error = Setpoint - Actual$$

Note that when the temperature is greater than the setpoint, it produces a negative error. This may seem arbitrary, but it's an important fact. As temperature increases, the drive must decrease to bring the system back to the setpoint. A temperature above the setpoint will produce a negative error. This is opposite of how the light sensor testing was performed. This is an example of the drive being inversely proportional. The output drive is proportional to the error, but in an opposite direction.

As mentioned, we will work in percentages, so let's review the math needed. First, we need to define a band of temperature control. How about 100 °F +/− 0.5 °F? This defines an allowable temperature band of 1.0 °F.

For a temperature error of +0.3 °F:

$$\%Error = E / 1.0\ °F \times 100\% = 0.3\ °F / 1\ °F \times 100\% = 30\%$$

For -0.8 °F?

$$\%Error = E / 1.0\ °F \times 100\% = -0.8\ °F / 1\ °F \times 100\% = -80\%$$

Quite simply, the %Drive due to proportional control is the proportional gain (K_P) times the %Error. A gain of 1 with a percent error of 30 yields a drive change of 30%!

$$\%Drive_{PROP} = K_P\%E = 1 \times 30\% = 30\%$$

With a gain of 3, and an error of 30%:

$$\%Drive_{PROP} = K_P\%E = 3 \times 30\% = 90\%$$

Note that the greater the error, the greater the drive due to proportional control. Consider what this means to our incubator. Let's assume a bias of 50% only got the temperature to 99.7F. This would produce an error of +30% for a total drive of 80% (with a K_P of 1).

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%\ Drive_{PROP} = 50\% + 30\% = 80\%$$

With a PWM duty cycle of 80%, what will the temperature in the incubator do? Increase. As the actual temperature approaches the setpoint, what happens to the proportional drive? It becomes less and less, backing off on total drive. Consider when temperature is at 99.9 °F. The error is +10%. What does this do for total drive?

$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%\ Drive_{PROP} = 50\% + 10\% = 60\%.$$

Notice that as the temperature approaches the setpoint, the drive is less. This may result in the temperature stabilizing below the setpoint, which is called steady-state error.

Proportional Band

Let's consider one more term: Proportional Band. Actually, we've already discussed all there is to know. It's simply another way of looking at the gain. Proportional band defines the percentage of the control band over which the system will take proportional control action.

Consider Figure 8-14a, which has a proportional band of 100%, and a K_P of 1. It shows that 100% of the drive is controlled over 100% of the control band. This is known as a 100% Proportional Band of Control. Gain is 1 because it is:

$$K_P = \Delta Output/\Delta Input = 100\%/100\% = 1$$

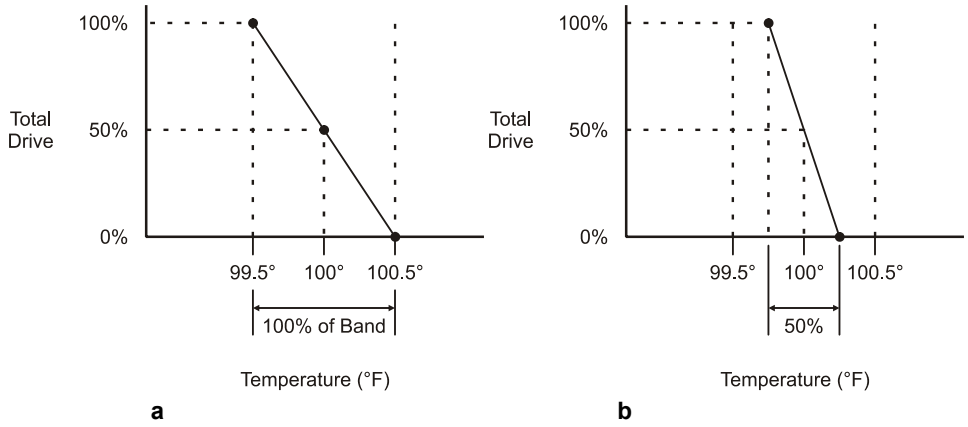


Figure 8-14: 100% Proportional Band (a) and 50% Proportional Band (b)

Now consider Figure 8-14b. 100% of the drive is controlled over only 50% of the control band. This is termed a 50% Proportional Band and has a gain of:

$$K_P = \Delta Output/\Delta Input = 100\%/50\% = 2$$

Gain and Proportional Band are actually inverses of one another. They are two different ways of discussing the same concept.

Applying Proportional Drive

It's time to apply the theory to practice and see the response of the system. For this run we want to be at the temperature that relates to a 50% bias drive.

√ Connect and plot with the following setting to find the 50% bias temperature:

Upper Temp:	120
Lower Temp:	70
Setpoint:	101.5
Band (+/−):	2
Time:	1
Kp, Ki, Kd:	0
%Bias:	50

√ Set the system setpoint to the current temperature, rounding to the nearest whole temperature (96.9 = 97).
√ Click the "Mid" button to scale to plot.

Run #1: Kp = 0.5

√ Set Kp = 0.5
√ NOTE: Text box values are not updated until you press "Enter", or move to another control.
√ Turn on fan and run until the temperature drops 5 °F, then turn off.
√ Monitor the control action until fairly stable.

Run #2: Kp = 1.0

√ Set Kp = 1.0
√ Enable cooling for a 5 °F disturbance.
√ Monitor the control action until fairly stable.

Run #3: Kp = 2.0

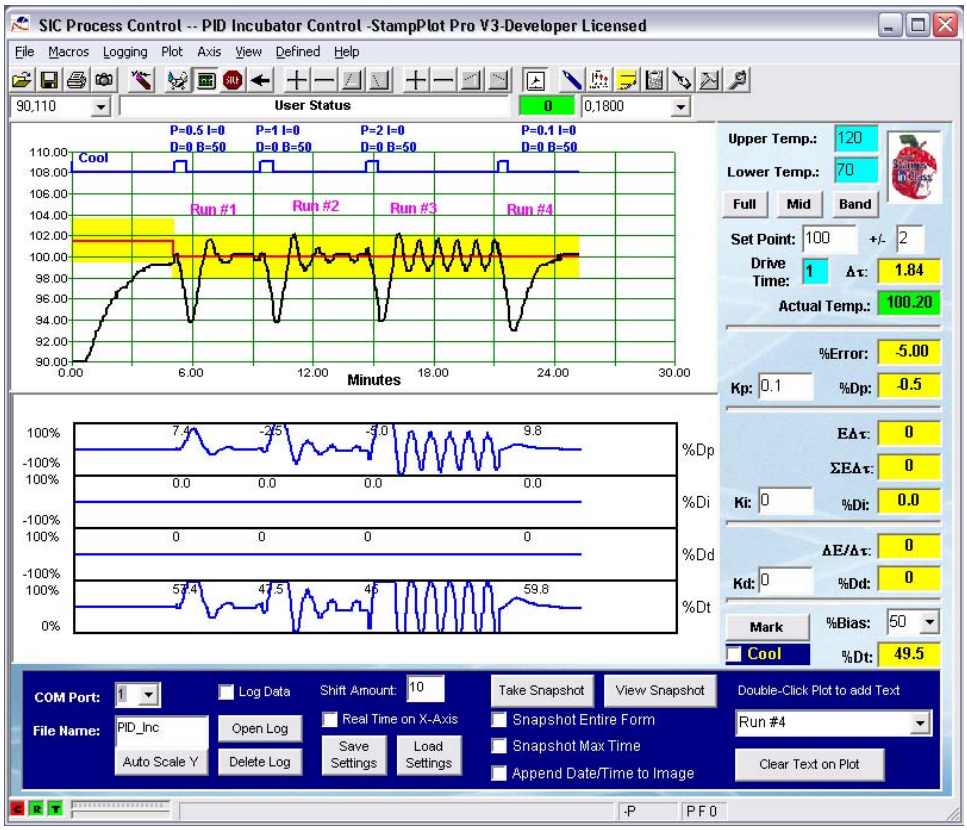
√ Set Kp = 2.0
√ Enable cooling for a 5 °F disturbance.
√ Monitor the control action until fairly stable.

Run #4: Kp = 0.1

√ Set Kp = 0.1
√ Set Band = 5
√ Enable cooling for a 5 °F disturbance.
√ Monitor the control action until fairly stable.

Figure 8-15 is a plot of our results.

Figure 8-15 Proportional Control Testing



Analysis

Prior to Run #1 we can see the temperature drifting slowly up due only to a 50% drive from bias.

Run #1: Kp = 0.5

Note that following the disturbance, the temperature rose quickly to the setpoint of 100 °F. Looking at %Dp and %Dt in the lower plots, we can see how the amount of

proportional drive was inversely proportional to the error and was added to total drive. The temperature oscillates briefly around the setpoint prior to stabilizing. This response is known as "Critically Damped" because the value quickly settles into the setpoint with few oscillations.

Run #2: Kp= 1.0
For a similar disturbance, the temperature rose and overshot by approximately the same amount as Run #1, but required more oscillations before setting into the setpoint. Compare the amount of %Dp for the same error as Run #1. With a higher gain, there is more forceful action. Due to the high number of oscillations, this system is "Under Damped". With a gain of 1, the system is operating with a 100% proportional band. That is, full control of the output occurs over full range of the control band. Note that at about time 11, the signal is at the top of the control band (100% of band). The output is at the bottom of the drive output (0% due to 50% - 50% proportional). Similarly, when temperature is at the bottom of the band (about time 10.5), the output is at maximum (100% due to 50% bias + 50% proportional).

Compare this to the drive in Run #1. The output was 100% when the temperature was at 96 °F, twice the width of the control band. Run #1 has a proportional band of 200%.

Run #3: Kp = 2
With a gain of 2.0, more forceful action is taken based on the same error. The temperature continually oscillates around the setpoint. This system is termed "Unstable". If gain is increased much more, the system would be operating in an on-off mode cycling between 0% and 100% as small errors drives the output one way and then the other.

Run #4: Kp=0.1
With a very low gain, the temperature slowly drifts up and settles into the setpoint because of the very low drive added due to error. This system is "Over Damped".

Challenge 8-3: Proportional Calculations

1. Draw a 200% proportional band curve for a system with a setpoint of 110 °F and a band of +/− 2 °F.
2. Calculate the proportional gain.
3. Based on the graph, what would the output be at 111 °F, 106 °F, and 115 °F?

ACTIVITY #4: PROPORTIONAL CONTROL NOT AT BIAS TEMPERATURE

Parts required:

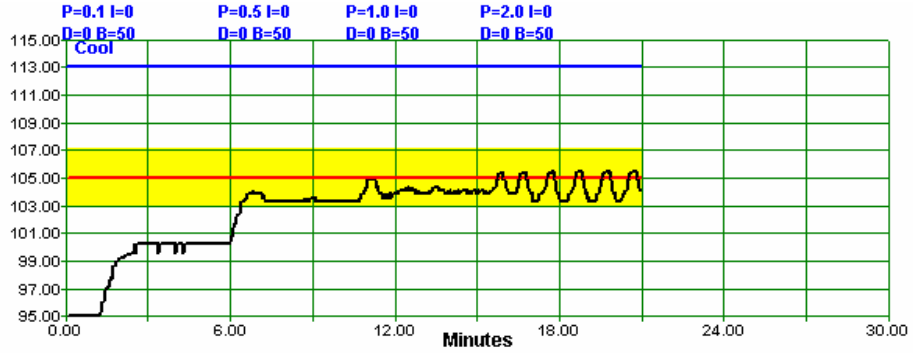
Same as Activity #1

While it seems pretty cut and dry that we can control at the setpoint using proportional control, what happens when not operating at the temperature defined by a 50% bias setting, or when a long-lasting disturbance affects the system equilibrium? In this activity we will explore controlling at many degrees away from where bias alone would be operating.

- √ Reset StampPlot.
- √ Connect on StampPlot.
- √ Set the Setpoint to a temperature distant from the temperature obtained with a 50% bias. In our case, we chose a temperature of 105 °F being 5 degrees away from the 50% operating temperature.
- √ Set a band of +/− 2.
- √ Set Kp to 0.1 and allow to stabilize.
- √ Repeat with gains of 0.5, 1.0 and 2.0.

Let's analyze the control response as seen in Figure 8-16.

Figure 8-16 Proportional Response not at 50% Bias Temperature



With a K_P of 0.1, temperature stabilized several degrees away from the setpoint. With bias at 50%, the proportional drive had to be positive to drive temperature up. But the

closer the actual temperature got to the setpoint, the less drive that proportional supplied. If the temperature actually reached the setpoint, the total drive would be 50 %, all due to bias. However, from previous activities, we found that 50% drive would result in a temperature of 98 °F.

In order for proportional drive to have control, there must be error. With a K_P of 0.1, the amount of error allowed a stable temperature around 100 °F. This error is called "Steady-State Error."

When K_P was increased to 0.5, this increased the proportional drive for the same error resulting in a temperature even closer to the setpoint. At 1.0, the steady-state temperature was even closer, but oscillations are beginning. Finally, with a K_P of 2.0, it drove even closer but had heavy oscillations, and as can be seen, the midpoint of the oscillations are still below the setpoint.

If bias drive is not correct for the setpoint, proportional drive alone cannot maintain temperature at the setpoint. Proportional drive is good for driving the system back towards the setpoint, but there needs to exist error for proportional drive to have effect.

Of course, the logical thing to do would be to adjust the bias to meet the conditions. In this case changing bias to 70% may provide sufficient energy. But what happens when the conditions change again? We are back to manual control at the setpoint.

Challenge 8-4

An oil flow system is designed and it is found that 80% drive is needed to drive the pump for the required flow rate.

1. If bias were set to 50%, would the flow stabilize above or below the setpoint?
2. How would increasing the proportional gain affect the steady state flow rate?
3. How would decreasing the proportional gain affect the steady state flow rate?

ACTIVITY #5: PROPORTIONAL-INTEGRAL CONTROL

Parts required:

Same as Activity #1

$$Co_{PID} = B + (K_P E) + (K_I \Sigma E \Delta t)$$

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

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, we saw 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 any error remaining due to long lasting disturbances or imbalances in the system. These errors 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 room temperature were 10 degrees cooler? 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 with a steady-state error remaining. The system will stabilize at a temperature below the desired setpoint. Over time, integral control 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!

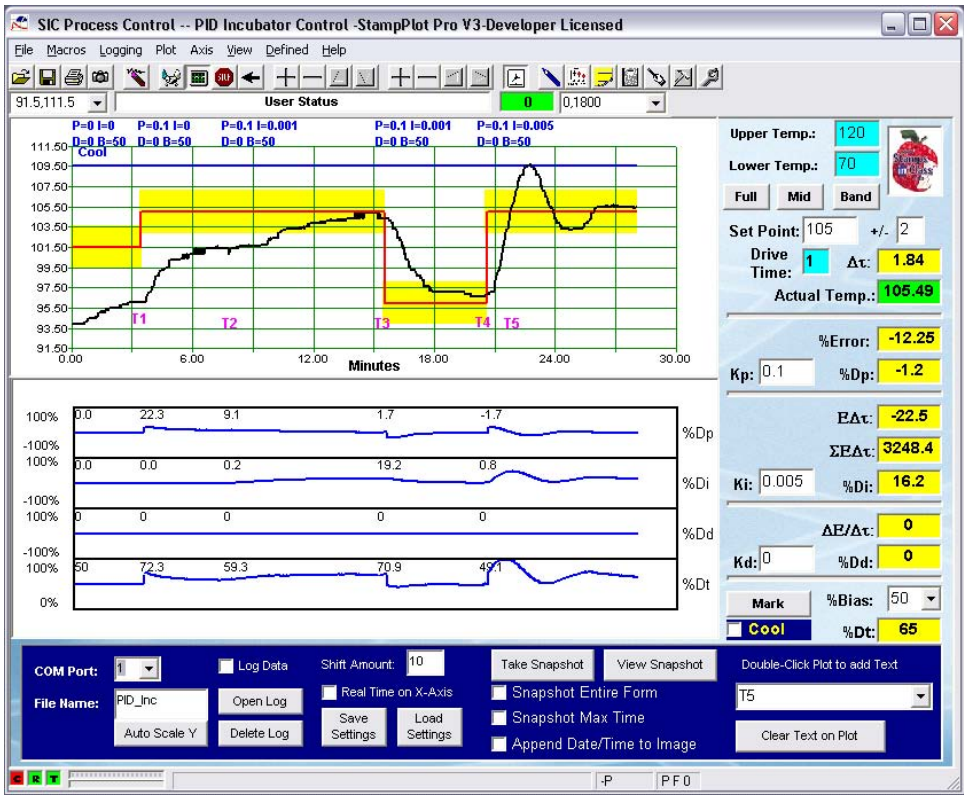
An industrial example may be that of adding pigment to paint for a desired color. Electronic circuitry monitors paint color and gradually adds pigment until the desired color is reached. In integral drive the amount of error is integrated over time. The larger the error and the longer it lasts, the greater the integral drive will be.

- √ Connect and plot.
- √ Update the settings with the following:

Upper Temp:	120
Lower Temp:	70
Setpoint:	101.5
Band (+/−):	2
Time:	1
Kp:	0
Ki:	0.000
Kd	0
%Bias:	50

- √ Allow the temperature to stabilize. Record this temperature here:_____.
- √ Set the setpoint approximately 10 degrees above this temperature.
- √ Change Kp to 0.1 and allow temperature to stabilize.
- √ Change Ki to 0.001 and allow temperature to stabilize.
- √ Change the setpoint to the temperature recorded above, and allow it to stabilize.
- √ Change the setpoint back to the +10 value.
- √ Change Ki to 0.005 and monitor.

Figure 8-17 Proportional - Integral System Response



Looking at our results in Figure 8-17, at Time 1 (T1 marked on plot), the system is fairly stable at 96 °F on only 50% bias. At this point, the setpoint is changed to 105 °F, K_P is set to 0.1, and the error causes temperature to rise.

At T2, the system has stabilized with proportional control at a temperature of 101.5 °F (pure coincidence). From the lower plot, we can see that the steady-state error is 9.1% for proportional drive. Integral control is introduced with a K_I of 0.001.

At T3, the system has almost reached the setpoint. Note that proportional drive is nearly 0 (1.7%) and integral drive is 19.2, for a total drive of 70.9. Because integral plus bias

are adding sufficient drive to be at setpoint, the error is nearly 0, therefore proportional drive is nearly 0. Also at T3, the setpoint is changed to the 50% drive temperature, which in our case was 96 °F. The drive built up by integral drive must again be removed based on the error and time.

At T4, the temperature is nearing the setpoint as integral is reduced. Integral drive is nearly 0 (0.8%), and total drive is 49.1%, with a small error providing –1.7% for proportional. The setpoint is returned to 105 °F with a K_I of 0.005, five times larger than previously. The integral drive builds quickly to cause an overshoot. Integral must again be reduced by error to bring temperature back down and causing undershoot. Hunting can occur with integral control, and the durations will be much longer based on the slow integration times.

At T5, note that the temperature is crossing the setpoint. What happens with the integral drive at this time? It is at this point that it turns and begins to be reduced. Depending on how long, and what magnitude of error, integral drive values can become very large.

Integral drive can be thought of as a dynamic bias. Based on error, integral drive will adjust to get the steady-state controlled variable to the setpoint. When integral drive has very low gain, the change in $\%Drive_{INT}$ will be very small, allowing proportional drive to respond to correct temperature. Large values of K_I are usually undesirable as they can cause the controlled output to be driven faster than the system can respond, leading to high overshoot and oscillations.

It is also important to limit the maximum value that $\Sigma E\Delta t$ reaches. A long lasting disturbance, such as a leaving the incubator door open overnight, could cause the value to reach extremely high values. Once the door is closed, the drive will be excessive, based on the high integral drive, and will require a very long time for the positive error to be driven away. This will lead to very high temperatures for a very long time. This is effect is called "Integral Windup." Feel free to try this test by removing the cover from the incubator, and allowing it run for 5 minutes, then replace the cover.

Challenge 8-5: Integral Control Testing

Can a system be controlled using integral control alone? Set %Bias to 50% and Kp to 0 and test the control action, and then discuss the results.

ACTIVITY #6: PROPORTIONAL-DERIVATIVE CONTROL

Parts required:

Same as Activity #1

$$Co_{PID} = B + (K_P E) + (\Delta E/\Delta t)$$
$$\%Drive_{TOTAL} = \%Drive_{BIAS} + \%Drive_{PROP} + \%Drive_{DERIV}$$

Derivative control responds to a CHANGE in the error. The fundamental premise determining derivative drive assumes that 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. It can also help damp the system to limit oscillations, and drive to limit the effects of short disturbances.

To achieve the best plot response, we will work around the stable temperature for 50% bias drive. Due to dealing with error over time, the smaller the resolution of our temperature, the better, so the upper and lower temperatures will be set accordingly.

- √ Connect and plot. Adjust your settings to the following:

Upper Temp:	50% Bias Temperature + 10 (e.g. 100+ 10 = 110 °F)
Lower Temp:	50% Bias Temperature − 10 (e.g. 100 − 10 = 90 °F)
Setpoint:	50% Bias Temperature (e.g. 100 °F)
Band (+/−):	2
Time:	1
Kp:	1
Ki:	0.000
Kd	0
%Bias:	50

- √ Allow temperature to stabilize.

Run #1

- √ Introduce a disturbance and monitor the resulting action.

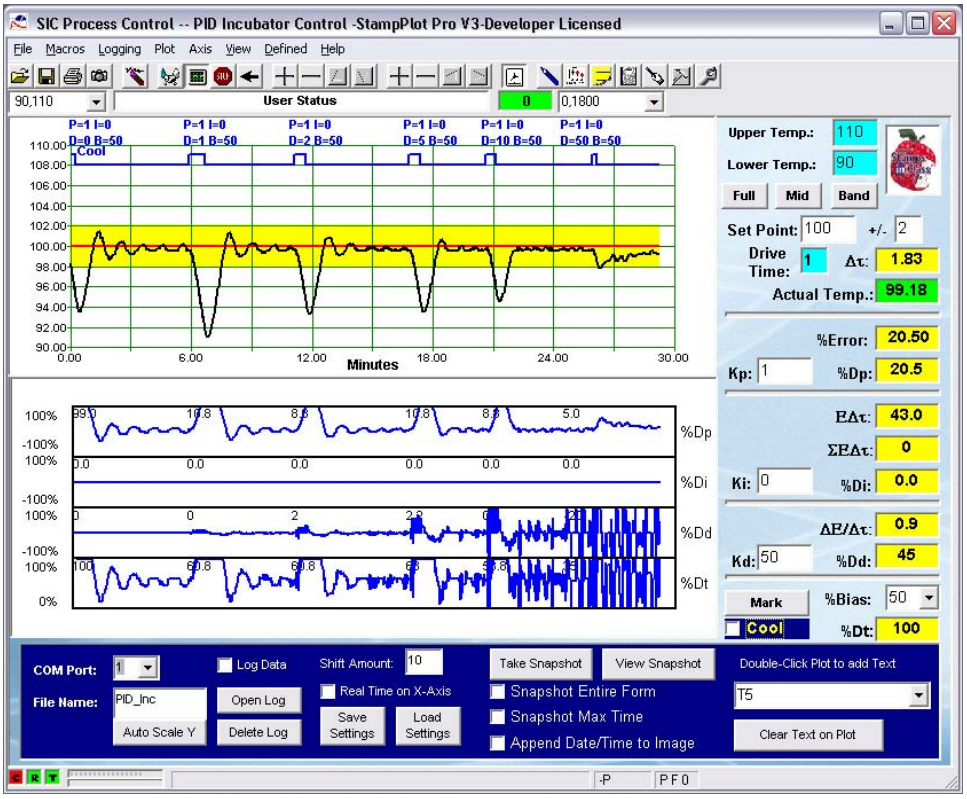
Run #2

- √ Set Kd to 1.0, or another under-damped response that provides multiple oscillations before stabilizing.

- √ Introduce a disturbance and monitor the resulting action.
- √ Starting with 1, use progressively higher gains for derivative (Kd) until the system returns as quickly as possible with little or no overshoot.
- √ Try a very large gain (50) for Kd and monitor the response to a small disturbance.

How did derivative control affect the response of the system? As shown in Figure 8-18, with only proportional control, there were heavy oscillations around the setpoint. As the derivative gain was progressively increased at times 6, 12, 18 and 21, the overshoot was progressively reduced and the system came under control quicker.

Figure 8-18 Proportional-Derivative System Response



Note the response of %Dd:

- Temperature increasing → %Dd is negative.
- Temperature decreasing → %Dd is positive.
- Temperature constant → %Dd is 0.

Derivative control drives to oppose the rising or falling temperature. When K_P is changed to 10, a small change in error produces very large changes in drive. The response of the system is such that there were almost no oscillations. With Proportional-Derivative control, a high proportional gain will assist control of the system by having a fast response, and results in very little proportional error. With derivative control, the heavy oscillations associated with a high proportional gain can be damped, resulting in a very fast, stable response. But too much derivative gain can lead to very unstable control of the system. Around time 24, the gain was increased to 50. With just a little disturbance, the erratic behavior of the system can be seen. As proportional drive tried to raise the temperature to the setpoint, derivate control fought any upwards motion. The total drive cycled continually between 0% and 100% as the error went one way and then the other. If this were an oil-flow system, consider what the pump must sound like during this control!

Derivative can be very good for limiting the effect of an error. Imagine operating at the setpoint, and quick, short disturbance causes the temperature to drop suddenly. Derivative drive will rapidly increase the output to limit the amount of the temperature drop. We were not able to show this response well with our slow-responding system. Can you?

Challenge 8-6: Derivative Control Testing

Can a system be controlled on derivative drive alone?

- √ After establishing operation near the setpoint, set Kp, Ki to 0 and %Bias to 0.
- √ Set Kd to a value of your choosing.
- √ Provide a cooling disturbance and test control of the system.
- √ Allow operation for several minutes.
- √ Discuss the results of your testing.

Final Challenge: Incubator 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. Increase the integral term until the system reaches the point of instability, and then back it off slightly.

As you gain experience in embedded control, you will see that the characteristics of the process will determine how you should react to error.

Part A:

- √ Load and run IncubatorPID-SP.bs2.
- √ Use StampPlot macro sic_pc_pid_sp_calc.spm.
- √ Tune the PID settings for the fastest stabilization for the incubator temperature of 101.5 °F +/− 1 °F with a bias of 50% and drive time of 2 seconds. Start from a stable temperature at the setpoint, and provide standard disturbance.
- √ Discuss your results and capture a screenshot of control action.

Part B:

A new species of eggs have arrived!

- √ Re-tune your system to control the temperature at 110 °F, +/− 2 °F with a 50% bias and drive time of 1 second. Start from a stable temperature at the setpoint, and provide standard disturbance.
- √ Discuss your results and capture a screenshot of control action following a disturbance.

Stand-Alone Control with the BASIC Stamp

Also included in the file distributions, but not used in the activities, are the following:

- BASIC Stamp 2 program IncubatorPID-BS2.bs2
- StampPlot macro sic_pc_pid_b2_calc.spm

This macro and program pair work together for an interface, but the BASIC Stamp performs the math. This allows the routines that read StampPlot to be commented-out, and stand alone operation to be performed. Also, with StampPlot interaction, the speed is somewhat greater. But in order for the BASIC Stamp to perform the necessary calculations, some limits were imposed:

- Values for gains are based on a scalar used (to get a K_I of 0.001) for example, and only limited values for gains are possible. Drop-down boxed on the interface are updated with allowable values when the connection opens.
- Maximum errors of 200% and drives of 200% in most cases.
- Maximum integral sum of 30000.
- An "Update" checkbox on the interface must be checked to have the BASIC Stamp read any updated values (including temperature settings and drive time).

```
' -----[ Title ]-------------------------------------------------------------
' IncubatorPID-BS2.bs2
' Controls an incubator using PID control with StampPlot Interface
' BASIC Stamp performs the calculations
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------

ADC_DataIn  VAR Byte       ' Analog to Digital Converter data
TempF       VAR Word       ' Calculated temperature in tenths

LED         PIN 0          ' LED output pin
ADC_CS      PIN 13         ' ADC Chip Select pin
ADC_Clk     PIN 14         ' ADC Clock pin
ADC_Dout    PIN 15         ' ADC Data output
ADC_Vminus  PIN 11         ' Control ADC offset
ADC_Vref    PIN 10         ' Control ADC Span

Heater      PIN 5          ' Incubator Heater
Fan         PIN 0          ' Incubator Fan

DriveTime   VAR Byte       ' Amount of time to drive heater in seconds
Err         VAR Word       ' Calculated error in tenths
TempDrive   VAR Word       ' Working variable
```

```
I_Edt       VAR Word       ' Integral variables

I_Sign      VAR Bit


DriveTotal  VAR Word       ' Calculated total drive (tenths)
LastErr     VAR Word       ' Last error amount
Bias        VAR Word       ' Amount of Bias drive
Band        VAR Byte       ' Size of temperature control band

Span        VAR Byte       ' ADC Span (tenths)
Offset       VAR Byte      ' ADC Offset (tenths)

SetP         VAR Word      ' Setpoint Value (tenths)
PWMVal       VAR Byte      ' Calculated PWM value (0-255)

x            VAR Byte      ' Working variable
SignBit      VAR Bit       ' Holds calculation sign (+/-)

' The actual gain is a function of K/Scalar
' Ex: Gain Prop = Kp/Kp_Scalar = 0 to 15 divided by 5 = 0.0 to 3.0
'     in increments of 0.2 (1/5)
Kp           VAR Nib       ' Proportional Gain Constant (scaled 0-15)
Kp_scalar    CON 5         ' Scalar for Proportional Gain
Ki           VAR Nib       ' Integral Gain Constant  (scaled 0-15)
Ki_scalar    CON 1000      ' Scalar for Integral Gain
Kd           VAR Nib       ' Derivative Gain Constant  (scaled 0-15)
Kd_scalar    CON 1         ' Scalar for Derivative Gain

' -----[ Initialization ]----------------------------------------------------
  ' Set initial values
  LOW Fan        ' Cooling off
  SetP = 1015    ' Temp in tenths
  Band = 20      ' Temperature band in tenths
  Kp = 0         ' 0-15, actual kp = kp/scalar)
  Ki = 0         ' 0-15, actual ki = ki/scalar)
  Kd = 0         ' 0-15, actual kd = kd/scalar)
  Offset = 80    ' Lowest temp
  Span = 40      ' Highest - lowest
  Bias = 50      ' Bias Drive
  DriveTime = 1  ' Drive time

  PAUSE 1000         ' Allow connection to stabilize
  GOSUB ConfigSP     ' Configure StampPlot Controls
  GOSUB ReadSP       ' Read SP updates, comment out to disable
                     ' StampPlot interactivity

' -----[ Main Routine ]------------------------------------------------------
DO
  GOSUB ReadADC
  GOSUB CalcError
```

```
  GOSUB AddBias
  GOSUB CalcP_Drive
  GOSUB CalcI_Drive
  GOSUB CalcD_Drive
  GOSUB Control_Incubator
  GOSUB ReadSP              ' Comment out to disable StampPlot interactivity
LOOP

' -----[ Subroutines ]------------------------------------------------------
' **** Sets ADC Span & Offset, Reads ADC Value, Calculates Temp F,
'      Updates StampPlot for temperature
READADC:
  PWM ADC_Vminus,Offset * 255/500,100
  PWM ADC_Vref,Span * 255/500,100
  LOW ADC_CS
  SHIFTIN ADC_Dout,ADC_Clk, MSBPOST,[ADC_DataIn\9]
  HIGH ADC_CS
  TempF = ADC_Datain * Span /26 + (Offset*10)
  DEBUG CR,DEC SetP,",",          ' Send setpoint
        DEC Band, ",",            ' Send Band
        DEC TempF,","             ' Send Actual Temp
RETURN

' **** Calculates the amount of Error based on setpoint and temperature
'      Based on band calculates %Error, maximum 200% in tenths
'      Signbit saved and recalled to perform math on positive integers.
CalcError:
  Err =  SetP - TempF
  SignBit = Err.BIT15
  Err = ABS(Err) * (1000/Band) MAX 2000
  IF Signbit = 1 THEN Err = Err * -1
  DEBUG SDEC Err,","            ' Send %Error
RETURN

' **** Adds bias to total drive
AddBias:
  DriveTotal = DriveTotal + (Bias * 10)
RETURN

' **** Calculates %proportional drive and adds to total drive
CalcP_Drive:
  signBit = err.BIT15
  TempDrive = ABS(Err)* Kp / Kp_Scalar MAX 2000
  IF signbit = 1 THEN TempDrive = TempDrive * -1
  DEBUG SDEC TempDrive,","       ' Send %Drive-P
  DriveTotal = DriveTotal + TempDrive
RETURN

' **** calculates Integral Drive.
'      Calculates integral sample based on error * time
'      Integrated error is accumulated in I_Et
```

```
CalcI_Drive:
  IF (Ki <> 0) THEN
    SignBit =Err.BIT15
    ' Scale to hold very high readings for SumEdt
    TempDrive = ABS(Err)/20
    IF SignBit = 1 THEN TempDrive = TempDrive * -1
    I_Edt = I_Edt + (TempDrive * DriveTime)
    SignBit = I_Edt.BIT15
    I_Edt = ABS(I_Edt) MAX 31000
    IF SignBit = 1 THEN I_Edt = I_Edt * -1
    TempDrive = ABS(I_Edt)/10
'     %Integral Drive based on total integrated error and gain
    TempDrive = TempDrive * Ki / (Ki_Scalar/200) MAX 2000 '(scalar /100) * 2
    IF SignBit = 1 THEN TempDrive = TempDrive * -1
  ELSE
'     Ki = 0 then reset total integrated error
    I_Edt = 0
    TempDrive = 0
  ENDIF
'     Update StampPlot and add to total drive
    DEBUG SDEC I_Edt,","                  ' Send SumEdt
    DEBUG SDEC TempDrive,","              ' Send Drive_I
    DriveTotal = DriveTotal + TempDrive
RETURN

' **** Calculate %Derivative based on change in error over change in time
'      Added to total drive
CalcD_Drive:
  TempDrive = Err - LastErr
  signBit = TempDrive.BIT15
  TempDrive = ABS(TempDrive)/ DriveTime
  IF SignBit = 1 THEN TempDrive=TempDrive * -1
  IF Kd = 0 THEN TempDrive = 0
  DEBUG SDEC TempDrive,","                ' Send dE/dt
  TempDrive = ABS(TempDrive) * Kd / Kd_Scalar MAX 2000
  IF signBit = 1 THEN TempDrive=TempDrive * -1
  DEBUG SDEC TempDrive,","                ' Send Drive-I
  DriveTotal = DriveTotal + TempDrive
  LastErr=Err
RETURN

' **** Drive incubator with PWM based on Drive total
Control_Incubator:
'      Ensure => 0 and < 1000 (100%)
  IF DriveTotal.BIT15 = 1 THEN DriveTotal = 0
  IF DriveTotal > 1000 THEN DriveTotal = 1000
  DEBUG SDEC DriveTotal,CR              ' Send Drive-T
  DEBUG IBIN Fan,CR                     '
'      Convert to 0-255 PWM value
  PWMVal = DriveTotal/10 * 255/100
'      If cooling, do not heat
```

```
  IF Fan = 1 THEN PWMVal = 0
'      Drive heater for drive time at 250mSec each
  FOR x = 1 TO DriveTime * 4
    PWM Heater,PWMVal,250
  NEXT
'     Reset Total Drive
  DriveTotal = 0
RETURN

' **** Read StampPlot for updates
ReadSP:
  DEBUG CR,"!READ (chkSet)",CR
  DEBUGIN  DEC signbit
'      If 'UPDATE' checked, read all other values
  IF signbit = 1 THEN
    PAUSE 50
    DEBUG "!Read (txtLower)",CR
    DEBUGIN DEC Offset
    PAUSE 50
    DEBUG "!Read [(txtUpper),-,(txtLower)]",CR
    DEBUGIN DEC Span
    PAUSE 50
    DEBUG "!READ [(txtSetP),*,10)]",CR
    DEBUGIN DEC SetP
    PAUSE 50
    DEBUG "!READ [(txtBand),*,20)]",CR
    DEBUGIN DEC Band
    PAUSE 50
    DEBUG "!READ (txtTime)",CR
    DEBUGIN DEC DriveTime
    PAUSE 50
    DEBUG "!READ (drpBias)",CR
    DEBUGIN DEC Bias
    PAUSE 50
    DEBUG "!READ [(txtKp),*,", DEC Kp_Scalar,"]",CR
    DEBUGIN DEC Kp
    PAUSE 50
    DEBUG "!READ [(txtKi),*,", DEC Ki_Scalar,"]",CR
    DEBUGIN DEC Ki
    PAUSE 50
    DEBUG "!READ [(txtKd),*,", DEC Kd_Scalar,"]",CR
    DEBUGIN DEC Kd
    PAUSE 50
    DEBUG "!READ (chkCool)",CR
    DEBUGIN DEC Fan
    PAUSE 50
    DEBUG "!O ChkSet=0(CR)!BELL",CR
  ENDIF
RETURN

'**** Configure controls on StampPlot
```

```
ConfigSP:
'     Clear gain drop-downs
  DEBUG CR,"!O txtKp.clear",CR,
        "!O txtKi.clear",CR,
        "!O txtKd.clear",CR

'     Populate each for allowable values based on the Scalar
  FOR x = 15 TO 0
    DEBUG  "!O txtKp.add = [[", DEC x,",/,", DEC Kp_Scalar,",],FORMAT,0.0]",CR
    DEBUG  "!O txtKi.add = [[", DEC x,",/,", DEC Ki_Scalar,",],FORMAT,0.000]",CR
    DEBUG  "!O txtKd.add = [[", DEC x,",/,", DEC Kd_Scalar,",],FORMAT,0]",CR
  NEXT

'     Update current values for settings
  DEBUG "!O txtKp=[[", DEC Kp,",/,",DEC Kp_Scalar,"],Format, 0.0]",CR,
        "!O txtKi=[[", DEC Ki,",/,",DEC Ki_Scalar,"],Format, 0.000]",CR,
        "!O txtKd=[[", DEC Kd,",/,",DEC Kd_Scalar,"],Format, 0]",CR,
        "!O drpBias=", DEC Bias,CR,
        "!O txtBand=[", DEC Band,",/,20]",CR,
        "!O txtSetp=[", DEC SetP,",/,10]",CR,
        "!O txtTime=", DEC DriveTime, CR,
        "!O txtLower=", DEC Offset,CR,
        "!O txtUpper=", DEC Offset + Span,CR,
        "!O btnFull.Run",CR,
        "!BELL",CR
RETURN
```


CONCLUSION

With PID control, the actual temperature is used as feedback and three separate drive evaluations are performed to calculate the final drive output to the control element. Bias drive is used to estimate the drive needed to sustain a setpoint value 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 is used to drive away steady-state 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. Volumes have been written on the subject of PID control and tuning. Not all systems require all three evaluations and drive. The floating ball, discussed earlier, operates fine without integral, but derivative is a must because of the fast disturbance reaction needed. A car's cruise control doesn't need derivative, because in general fast disturbances are unlikely and the control action would be probably be uncomfortable to feel.

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 you 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 your incubator?

SOLUTIONS TO CHAPTER 8 CHALLENGES

Challenge 8-1 Solution

Part A:

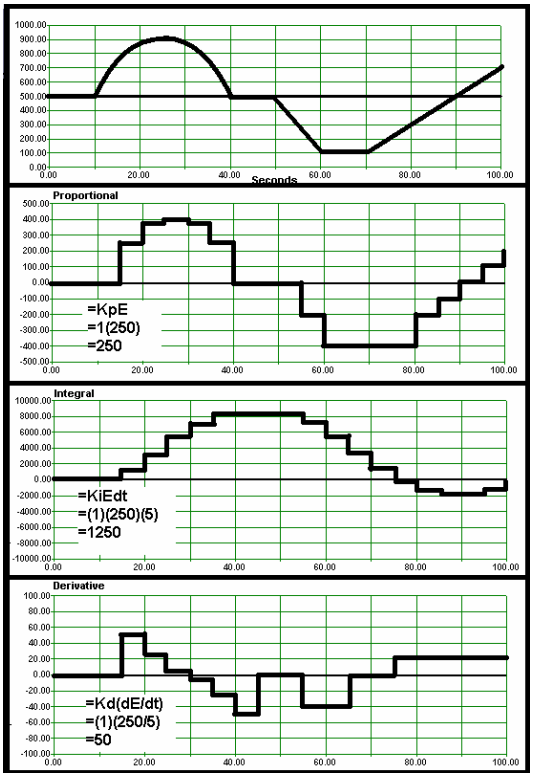


Figure 8-19
Challenge 8-1 Part A Solution

Part B:
Below is data representing the systems conditions and data for PID evaluations. Calculate the output for each evaluation. Kp = 2 Ki = 0.01 Kd = 4 Setpoint = 500

Table 8-2: PID Evaulation Activity Solutions
Given Kp = 2 Ki = 0.01 Kd = 4 Setpoint = 500

Actual	∆t	E	∆E	P(KpE)	E∆t	I(KiΣE∆t)	D(Kd∆E/∆t)
500	1.0	0	0	0	0	0	0
510	1.0	10	10	20	10	.1	40
490	0.5	-10	-20	-20	-5	0.05	-160
480	0.5	-20	-10	-40	-10	-0.05	-80
470	2.0	-30	-20	-60	-60	-0.65	-40
520	1.0	20	50	40	20	-0.45	200
500	0.5	0	-20	0	0	-0.45	-160
500	1.0	0	0	0	0	-0.45	0

Challenge 8-3 Solution

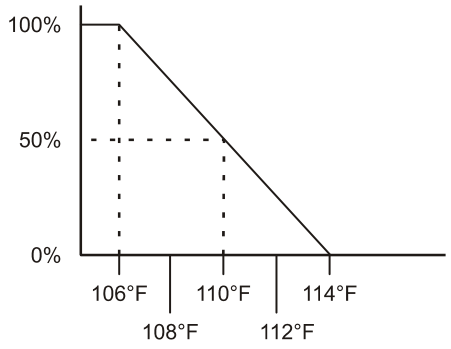


Figure 8-20

1. 200% proportional band curve for a system with a setpoint of 110 °F and a band of +/- 2 °F

2. Proportional Gain = 1/200% = 1/2 = 0.5
3. 111 °F = 37.5%
   106 °F = 100%
   115 °F = 0%

Challenge 8-4 Solution

1. A positive error would remain and flow would stabilize to less than the setpoint.
2. Increasing gain would result in a smaller error, but would result in more oscillations.
3. Decreasing gain would result in a larger error with fewer oscillations.

Challenge 8-5 Solution

Yes, a system can be controlled using integral alone, though response will be very slow with low settings of K_I, and larger values can lead to excessive oscillations. It would be appropriate in systems where disturbances are relatively small but long lasting.

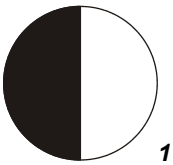
Challenge 8-6 Solution

No, a system cannot be controlled on derivative drive alone over the long haul. Temperature may stay fairly stable for a couple minutes as derivative bucks the change, but the temperature will drift lower, eventually to room temperature. Derivative responds to the rate at which error changes only, and is not based on the actual error.

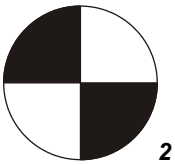
Final Challenge Solution

Solutions will vary, as they will be unique for each system.

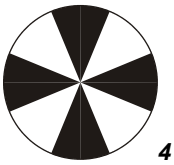
Appendix A: Cut-Outs



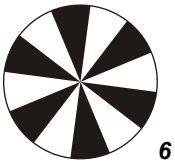
1 Cycle/revolution



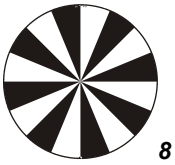
2 Cycles/revolution



4 Cycles/revolution



6 Cycles/revolution



8 Cycles/revolution

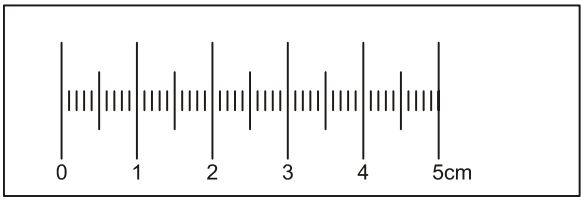
Retro-Reflective Encoder Wheels

Cut-out Duplicate of Figure 4-17 from page 116.

NOTE: Test to see if the optic sensor can detect the black area on these encoders before cutting them out.

If the optic sensor cannot see these encoders, make a copy of this page using a laser or ink copier. Poor-quality pale images or images made with a photo printer (which uses metallic ink) will not properly absorb infrared light, and therefore will not work.

A pdf of these encoders can be found in the free download of this book on the Process Control product page at www.parallax.com.



Reflection Ruler

Cut-out Duplicate of Figure 4-4 on page 88.

B

Appendix B: Parts Listing

To perform the experiments in this text, you will need the following:

Computer System Requirements:

- PC running Windows 2000/XP
- An available serial port or USB port. If you need a USB to Serial Adapter, we recommend Parallax part #800-00030.
- CD-ROM or Internet access

Software Requirements (Available free from the Parallax CD or as a download from the Process Control product page):

- BASIC Stamp Editor for Windows v2.0 or higher
- StampPlot Pro version 3, release 6, under the free Home/Educ. License
- StampPlot Macros for Process Control. (The .bs2 code is a free download only.)

Hardware Requirements

- Board of Education Full Kit Serial (#28102) or USB (#28802)*
- One of the following power supplies:
  - 7.5 V unregulated
  - 9 V regulated
  - 9 V 300 mA unregulated
- Process Control Parts Kit (#130-28176)
- 9 V battery (or 7.5 V unregulated or 9 V regulated power supply) for Chapter 6, Activity #3: Manual Control of Incubator

Household Items:

- Scissors
- Tape or rubber cement
- Lighter or matches
- Ice or other cold object, such as a chilled soda can

***About BASIC Stamp HomeWork Board Compatibility**

All of the activities in this text are safe with the HomeWork Board, except as noted on page 120. However, the HomeWork Board's Basic Stamp I/O pins are protected with 220 Ω series resistors, which will need to be accounted for in activities using RC networks and amplifier circuits. Also, the incubator activities drain 9 V batteries very quickly; so a fresh battery may be needed with each experiment.

Process Control Parts & Text Kit #28176 **Process Control Parts Kit #130-28176** Parts and quantities subject to change without notice		
Parallax Part #	**Description**	**Quantity**
122-28176	*Process Control* text (Included in #28176 only)	1
150-01011	Resistor, 5%, 1/4 W, 100 Ω	1
150-01012	Resistor, 5%, 1/2 W, 100 Ω	1
150-01020	Resistor, 5%, 1/4 W, 1 kΩ	3
150-01030	Resistor, 5%, 1/4 W, 10 kΩ	3
150-01040	Resistor, 5%, 1/4 W, 100 kΩ	1
150-01051	Resistor, 5%, 1/4 W, 1 MΩ	1
150-02210	Resistor, 5%, 1/4 W, 220 Ω	6
150-04702	Resistor, 5%, 1/4 W, 47 kΩ	1
150-04720	Resistor, 5%, 1/4 W, 4.7 kΩ	1
152-01031	Potentiometer - 10 kΩ	1
153-00002	Transistor, BS170 MOSFET 60 V N-channel	1
200-01040	Capacitor, 0.1 µF, 100 V	1
200-06840	Capacitor, Metal Film, 0.68 µF	2
201-01062	Capacitor, electrolytic, 10 µF, 25 V	1
350-00001	LED - Green - T1 3/4	1
350-00006	LED - Red - T1 3/4	1
350-00007	LED - Yellow - T1 3/4	1
350-00009	Photoresistor	1
350-00021	Optoreflective switch QRB1114	1
400-00002	Pushbutton – Normally Open	2
500-00001	Transistor, NPN BJT – 2N3904	2
602-00015	Dual Op-Amp, LM358 8-pin DIP chip	1
604-00011	LM34 Temperature Sensor	1
700-00040	Fan, Brushless, 12 V DC	1
700-00079	Test tube, polystyrene 12 x 55 mm	1
800-00016	3” Jumper Wires – Bag of 10	2
900-00001	Piezo Speaker	1
ADC0831	8-bit A/D Converter– 8-pin DIP chip	1

Index

E

F

G

H

I

K

R

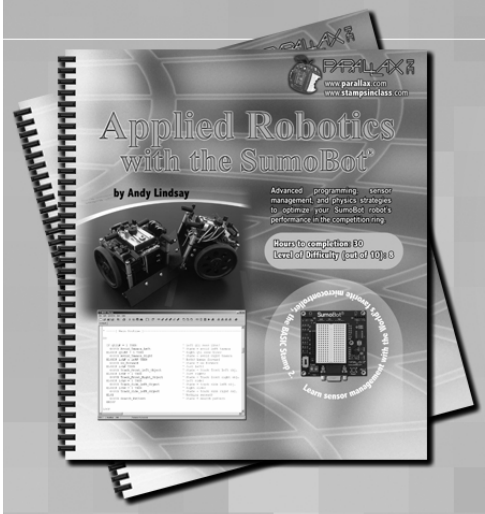
S

T

U

V

W



KEEP ON EXPERIMENTING!

If you enjoyed this set of experiments, consider these other tutorials from the Parallax Stamps In Class program. All of our educational texts are available as free downloads online in *.PDF format. Visit www.parallax.com/sic for details.

- What's a Microcontroller? (#28123)
- Applied Sensors (#28127)
- Process Control (#28156)
- Understanding Signals (#28119)
- Robotics with the Boe-Bot (#28125)
- IR Remote for the Boe-Bot (#70016)
- Applied Robotics with the SumoBot (#27403)
- Advanced Robotics with the Toddler (#122-00001)
- Basic Analog and Digital (#28129)
- Elements of Digital Logic (#70008)

We are always adding new tutorials to our current offering. Check our web site for the latest Stamps In Class news.

StampWorks

Includes 35 experiments based on the BS2-IC module and the Professional Development Board (included). StampWorks gives you the hardware, the electrical components and, most importantly, the know-how to become a confident embedded programmer.Working your way through StampWorks you will learn about efficient embedded design, connecting circuits and "smart" sensors to the BASIC Stamp, adding computer control to your projects, and "Power PBASIC" programming techniques.

StampWorks Experiment Kit; #27297

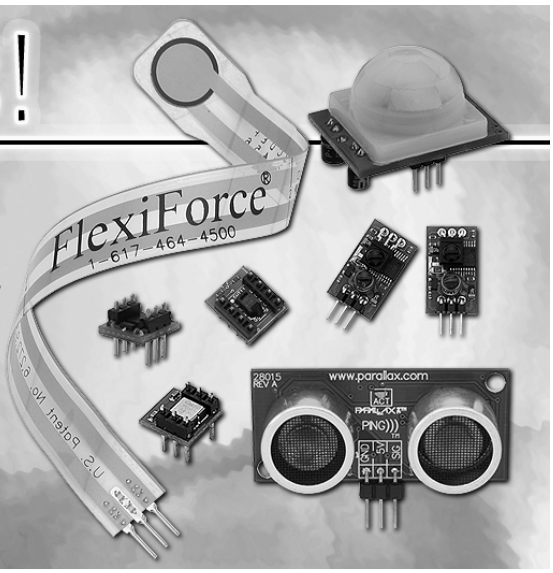
PARALLAX

www.parallax.com

sensors!

Parallax offers a wide array of sensors suitable for your next robotic, industrial or home automation project. We have sensors to detect touch, light, color, temperature, humidity, motion, distance, vibration, pressure, direction, tilt and acceleration.

Check out our Sensor selection in the Product/Accessory section of our web site at www.parallax.com.



PARALLAX

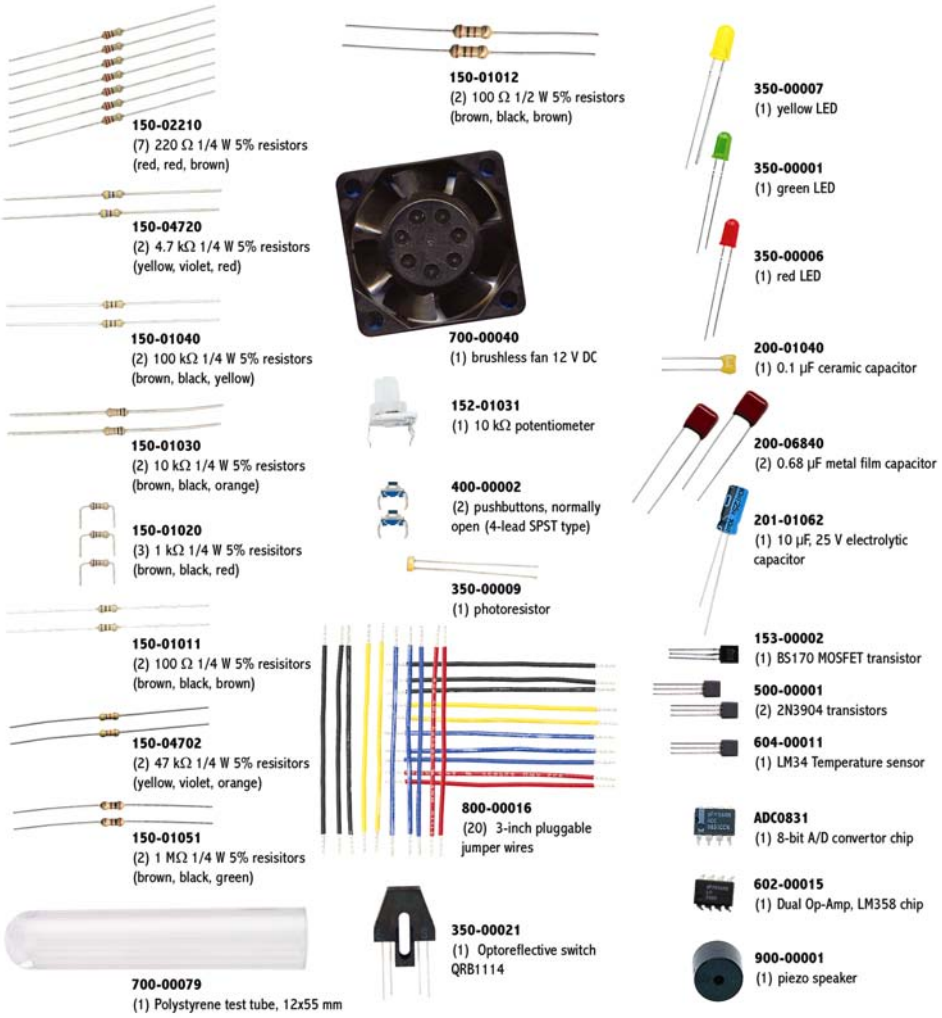
SERIAL LCDS BY PARALLAX

Parallax Serial LCDs are very functional, low-cost LCDs that can be easily controlled by a BASIC Stamp. Our LCDs have basic text wrapping so that your text looks legible on the display. In addition, the Serial LCD provides you with full control over all of its advanced LCD features, allowing you to move the cursor anywhere on the display with a single instruction and turn the display on and off in any configuration. It supports the same visible characters as the BASIC Stamp Editor's Debug Terminal.

- **2x16 Serial LCD (#27976)**
- **2x16 Serial LCD - Backlit (#27977)**
- **4x20 Serial LCD - Backlit (27979)**

PARALLAX

WWW.PARALLAX.COM



Parts and Quatities in the Process Control Parts Kit are subject to change without notice. Parts may differ from what is shown in this picture. Please contact stampsinclass@parallax.com if you have any questions about your kit.