

Experiments with Renewable Energy

Student Guide

VERSION 1.0a

PARALLAX 

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 2004 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, 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. StampPlot is a registered trademark of Selmaware Solutions. Other brand and product names are trademarks or registered trademarks of their respective holders.

ISBN 1-928982-22-0

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 Stamps](#) – 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 curriculum 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 curricula and to provide a forum for educators to develop and obtain Teacher's Guides.
- [Translators](#) – The purpose of this 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.
- [Robotics](#) – Designed exclusively 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.

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	iii
Audience	iv
Educator Resources	v
Foreign Translations	vi
Special Contributors	vi
Chapter 1: Setting Up for the Experiments	1
What is Energy?	1
About the Experiments in this Book	9
Activity #1: Building a Software Foundation and Superstructure	11
Activity #2: Programming the StampPlot Graphical Plotting Software Interface	18
Activity #3: Testing the StampPlot Graphical Plotting Software Interface	25
Activity #4: Build, Program and Test the 4-Channel A/D Converter	26
Chapter 2: Experiment #1, Programmable Battery Charger	43
How NiCad Rechargeable Batteries Work	44
Activity #1: Checking the Battery Voltage	48
Activity #2: Building the Programmable Battery Charger	50
Activity #3: Programming the Battery Charger	56
Activity #4: Discharging the Batteries	67
Activity #5: Charging the Batteries	71
Activity #6: Replayng the Charge Cycle	77
Activity #7: Detecting the End of Charge	80
Summary and Applications	80
Chapter 3: Experiment #2, Dueling Solar Cells	83
How Solar Cells Work	83
Activity #1: Building the Dueling Solar Cells	88
Activity #2: Programming the Dueling Solar Cells	91
Activity #3: How the Dueling Solar Cells Work	98
Summary and Applications	105
Chapter 4: Experiment #3, Solar Cell Sun Tracker	109
Real World Sun-Tracking Solar Arrays	109
Activity #1: Shielding the Solar Panels	116
Activity #2: Centering the Servo	119
Activity #3: Adding the Servo	122
Activity #4: Programming the Sun Tracker	126
Activity #5: How the Sun Tracker Works	130
Summary and Applications	134
Chapter 5: Experiment #4, Half and Full Wave Rectification	139

Activity #1: The Principles of Half Wave Rectification	141
Activity #2: Assembling the three-phase AC Alternator.....	142
Activity #3: Building the Half Wave Rectifier.....	148
Activity #4: Programming the Half Wave Rectifier.....	150
Activity #5: How the Half Wave Rectifier Program Works	152
Activity #6: Full Wave Rectifier Operation	162
Activity #7: Building the Full Wave Rectifier Circuit	164
Summary and Applications.....	170
Real World Power Supply Design	171
Power Electronic Systems.....	176
Chapter 6: Experiment #5, Three-Phase AC Alternator	179
Understanding Three-Phase Power	179
Activity #1: Building the Three-Phase Wye Circuit	190
Activity #2: Programming the Three-Phase AC Alternator	196
Activity #3: How the Three-Phase AC Alternator Program Works	201
Summary and Applications.....	221
Three-Phase Power in the Real World	222
Generating Electricity using Wind Power	224
Appendix A: LearnOnLine's REEL Power Project.....	235
Appendix B: Source Code Listing.....	239
Appendix C: Programming StampPlot Pro.....	251
Appendix D: Parts Listing	285
Appendix E: Resistor Color Code	289
Index.....	295

Preface

Energy is all around us. It heats our homes, powers our light bulbs and appliances, fuels our cars and provides for a variety of professional careers that deal with its many elements. It also comes in many forms such as heat, light, chemical, mechanical and electrical energy. And, according to physicists, energy can neither be created nor destroyed, only converted from one form to another. So why learn about energy? The answer is because energy, and the conversion of energy from one form to another, is fundamental to our modern living environment. By knowing the principles behind energy generation and conversion, you will come away with a knowledge base that can be applied to nearly every modern electrical, mechanical and chemical device that uses or produces power.

This *Experiments with Renewable Energy* course deals with one particular form of energy, electricity. In this course you will learn about the fundamentals of direct current (DC) and alternating current (AC) and how they both apply to our everyday use of electricity. By studying electrical energy as the primary theme, you will also learn simple, but elegant, programming techniques using PBASIC, the higher-level computer language of the BASIC Stamp microcontrollers. And to add a dramatic visual dimension to your learning activities, we have also integrated StampPlot software. StampPlot software provides you with the ability to graph the real-time voltage measurements that make up the basis for each of the five experiments that comprise this course.

This series of five experiments will introduce you to DC and AC electrical concepts using the Board of Education and the BS2 series of microcontrollers. To produce DC and AC power, these experiments use rechargeable batteries, miniature solar cells, and a three-phase wind-driven AC alternator that, taken together, will help you learn about two forms of renewable energy as well. The DC experiments start with a programmable battery charger and move into experiments with solar cells, including a servo-driven sun tracker. The AC experiments involve the use of a miniature wind-driven AC alternator to demonstrate two basic forms of rectification, half wave and full wave, and conclude with comparing single-phase and three-phase power generation and their applications to real-world devices.

The underlying goal of this course is to whet your intellectual appetite to learn more about the various forms of electrical energy and how best to apply them in a technical sense. To that end you will find detailed information about the theory and practice

behind each experiment, including links to appropriate web sites. One particularly noteworthy web site, LearnOnLine.com, hosts the REEL Power project. REEL Power, which stands for Renewable Energy Education Lab, is the first national renewable energy project where students can participate in experiments with miniature solar panels, wind turbines and fuel cells in the classroom and then seamlessly share their results with other participating REEL Power users on the World Wide Web. To learn more about REEL Power, point your web browser to www.learnonline.com. Also, check out the REEL Power section in the Appendix for an overview of this award-winning renewable energy project.

Another important goal of this course is to introduce students to careers in the energy and power electronics fields. Our country's dependence on electrical power and the electronic components that control and process this power still remains a sleeping giant. While computers and telecommunications were the technology bellwethers of the 1980s and 1990s, power technologies will dominate at least the first part of the 21st century electronics industry, largely due to our reliance on these computer-based technologies. This is why students should seriously consider careers in the energy and power electronics fields, because they will shortly represent the new-growth industries that will cry out for technical talent. Hopefully, this *Experiments with Renewable Energy* course will help to inspire you to participate in this new technology revolution.

This *Experiments with Renewable Energy* Student Guide will be periodically revised, expanded and updated based on feedback from students, educators and experimenters. We encourage all of you to submit any suggestions for additional experiments as well as author additions to this text. Please send them to stampsinclass@parallax.com. We will do our best to integrate your ideas and assist you with whatever technical support or sales support you need.

AUDIENCE

This *Experiments with Renewable Energy* Student Guide was created for ages 15 and above, ideally as a subsequent text to the *What's a Microcontroller?* and *Basic Analog and Digital* guides. Due to this, it is preferable that both students and teachers are already familiar with the PBASIC instructions and understand their functions in a normal programming environment. While some attention will be devoted to explanations of how the programs work, this is not a course in programming. However, all of the PBASIC source code will be available as a free download to educators, so the experiments may be accomplished without a prerequisite course by following the check-mark instructions. Students and teachers are referred to the *BASIC Stamp Programming Manual* and the

Help menu of the BASIC Stamp Editor for full details of PBASIC instructions and their operations.

Similarly, the experiments in *Experiments with Renewable Energy* use a multiplexed, 4-channel A/D converter system that takes in voltage values and outputs binary values relative to the converted analog voltages. The theory behind a single-channel A/D converter is covered in the *Basic Analog and Digital* guide that still applies to our 4-channel model. Rather than repeat what's been covered before, this text does not delve deep into this device. The *Experiments with Renewable Energy* series of experiments will focus on teaching new electrical and programming techniques that build on the experiments and theory presented in these preceding guides.

The target audience for these experiments are high school, community college and undergraduate college students who desire an introduction to understanding DC and AC electrical concepts in conjunction with solar and wind renewable energy technologies. Additionally, these experiments are also geared to hobbyists of all ages whose enthusiasm and feedback on the subjects presented can provide for continued improvements in this guide.

Finally, this text is designed for teachers who wish to augment their math and science instruction with direct, hands-on applications of the academic principles involved in these subjects. Teachers always hear the same "What good is studying math and science?" and "When will I ever get to use it?" lines from their students. This text gives teachers viable answers to these recurring questions, and also provides the incentive for students to excel in these subjects given their direct participation in the experiments.

EDUCATOR RESOURCES

The following Parallax Discussion Forums are available for those who would like support in using this text:

[Stamps In Class Group](#): Open to students, educators, and independent learners, this forum allows members to ask each other questions and share answers as they work through the activities, exercises and projects in this text.

[Parallax Educator's Group](#): Educators may join this forum after proof of status as an educator has been verified by Parallax. This moderated forum provides support for educators, provides a place for educators to post Teacher's Guides of their own

development, and welcomes feedback as we continue to develop our Stamps in Class curriculum. Parallax also posts question and answer keys for selected Stamps in Class titles on this forum.

These groups are accessible from www.parallax.com under Discussion Forums on the Support menu. If you are having difficulty subscribing to either of these groups, or have other questions about this text or Stamps in Class, contact the Parallax Stamps in Class Team directly at stampsinclass@parallax.com.

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 discussion group for Parallax translators which you may join. Please check the Website and Discussions Lists section after the title page.

SPECIAL CONTRIBUTORS

The author of this book is John Gavlik of LearnOnLine, Inc. Mr. Gavlik is an electronics engineer with a BSEE degree from the California Polytechnic State University (CalPoly) and is also president of LearnOnLine, Inc. LearnOnLine is involved in cutting edge Distance Learning projects that make use of the Internet and World Wide Web for both the content and delivery of teaching and learning materials.

One such project is REEL Power, which stands for Renewable Energy Education Lab, designed to create a “simulated” nationwide, student-built, Internet-connected power grid where each school becomes a “virtual power plant” on the grid. The material presented here is closely based on the REEL Power project. An overview of the REEL Power project can be found in the Appendix, plus you are invited to visit <http://www.learnonline.com> for complete information.

On February 9, 2003, LearnOnLine, Inc. was honored to receive the 2002 Power Sources Manufacturers Association (PSMA) Power Electronics Educational Award for its REEL Power Renewable Energy Education Lab project. According to the PSMA, this annual educational award is presented to the company or institution that has made the most significant effort to enhance awareness of Power Electronics to students in Kindergarten through undergraduate college. The award is intended to encourage companies and institutions to sponsor energy-related programs, which highlight the contributions,

challenges and opportunities available in the field of power electronics. Visit www.psma.com for complete information on the Power Sources Manufacturing Association.

Credit for the StampPlot software used in the experiments goes to Southern Illinois University professors Martin Hebel and Will Devenport for their work in creating the StampPlot graphic software. Martin also prepared the Appendix on StampPlot and worked closely with John Gavlik in the development of specialized interfaces for this text.

Credit for the three-phase AC alternator used in experiments 4 and 5 goes to Edwin Lenz, whose ingenuity and knowledge of wind-powered AC systems translated into a great demonstration model for this project.

Additional recognition goes to the Parallax technical support team who contributed valuable ideas and suggestions for this project. In particular these people include Andy Lindsay for support and evaluation of the materials and software presented here, Erik Wood for tips on marketing this course correctly, and Ken Gracey for selecting and encouraging the author to develop the *Experiments with Renewable Energy* project. Thanks to Ken Gracey, Aristides Alvarez, Rich Allred, Stephanie Lindsay, Kris Magri and John Barrowman for technical editing and review, to Rich Allred for technical graphics, and to Larissa Crittenden for cover design. Thanks again to customer Sid Weaver for his review in advance of publication.

Chapter 1: Setting Up for the Experiments

Welcome to this course on renewable energy. With it we hope that you will learn about the various forms of energy that you come in contact with every day such as light, heat, and especially electrical energy, which is the main focus of this course. Energy is involved in every action that occurs in the universe, including all physical actions like the sun shining, the winds produced by the weather, the actions of the tides and the growth and decay of plants and animals. All of these physical actions can be converted to electrical energy. The experiments in this course will allow you to harness and analyze energy produced by sunlight and by wind.

WHAT IS ENERGY?

Energy is neither created nor destroyed. This is called the principle of Conservation of Energy. In other words, the amount of energy in the universe always remains the same. And when we use energy, like burning wood to generate light and heat, we don't use it up; we simply transform it from one form of potential energy (fuel) into other forms of kinetic energy (heat and light).

Almost all energy transformations involve the production of heat, which is considered the lowest form of energy, because it quickly dissipates into the surroundings and is normally unavailable for further use. So, although the total amount of energy remains the same, the amount of "useable" energy constantly decreases. However, don't worry too much about the decrease of useable energy; our Sun is scheduled to produce solar energy for many years to come, so taking this course will not be a waste of your time (or energy).

The Definition of Energy

Before going much further you should first learn the fundamental definition of energy. For example, is light by itself energy? Is electrical energy measured in watts, or in calories for heat? Up until now you may have thought so, but let's get a strict definition of energy to start off with and then proceed with how to measure it.



Simply put, energy is the capacity to do work.

Very good! Now what do we mean by work? Is it a job you go to after school or on the weekend? Not hardly, at least not for a strict definition of energy.

Work is done when a force moves an object.

Two things need to be measured:



- 1 . The amount of the force used to move the object
- 2 . The distance that the object moves.

Multiplying these two quantities together (**Force × Distance**) gives the numerical size of the work done. And combining the units used to measure the force and the distance defines the units for work.

The common symbol for energy is the uppercase letter E and the standard energy unit is the joule, symbolized by J. Joules can be defined in terms of both mechanical energy familiar from the above definition of work, and electrical energy in circuits, which is a measure of power expended over time

The standard energy unit: the joule (J)



Mechanical Energy: One joule (1 J) is the energy resulting from the equivalent of one newton (1 N) of force acting over one meter (1 m) of displacement.

Electrical Energy: One joule (1 J) is equivalent to one watt (1 W) dissipated or radiated for one second (1 s).

A common unit of energy in electric utilities is the kilowatt-hour (kWh), which is the equivalent of one kilowatt (kW) dissipated or expended for one hour (1 h). Because

$$1 \text{ kW} = 1000 \text{ W}$$

$$1 \text{ h} = 3600 \text{ s}$$

$$1 \text{ kWh} = 3.6 \times 10^6 \text{ J}$$

We will be using other units of measure, including volts, amps, ohms, watts and hertz, as we conduct the experiments in the upcoming chapters.

Electrons are one of the three fundamental parts of an atom, along with protons and neutrons. One or more protons and neutrons stick together in the center of the atom to form the nucleus. The tiny electrons orbit around the nucleus. Electrons repel each other, and electrons and protons attract each other.

Charge: The tendency for an electron to repel from another electron and attract to a nearby proton is called negative charge. The tendency for a proton to repel from another proton and attract an electron is called positive charge. Negatively charged molecules have more electrons than protons; positively charged molecules have fewer electrons than protons. A molecule with an equal number of electrons and protons is neutrally charged.

The volt (V) is the unit of measurement for electrical pressure. Voltage can be thought of as electrical pressure caused by the attraction of electrons in a negatively charged molecule to the protons of a positively charged molecule.

The ampere, or amp, (A) is the unit of measurement for current. Current refers to the number of electrons per second passing through a circuit. Current is commonly denoted by the letter "I" in schematics and equations.

The ohm (Ω) is the basic unit of measurement for resistance. Resistance (R) is the ability of an element in a circuit to inhibit the flow of current.

Ohm's Law: The relationship among voltage, current, and resistance is defined as

$$V = I \times R$$

which states: "The voltage measured across a resistor's terminals (V) equals the current passing through the resistor (I) times the resistor's resistance (R)."

Potential and Kinetic Energy

Energy comes from many sources such as sunlight, wind, water, coal, oil, gas etc., and is of many types: thermal, electrical, chemical, nuclear etc. Whatever the source, there are two main forms of energy: potential energy and kinetic energy.

Potential energy, sometimes symbolized by an upper case 'U', is energy stored in a system. A stationary object in a gravitational field, or a stationary charged particle in an electric field, has potential energy.

Kinetic energy is observable as the motion of an object, particle, or set of particles. Examples include the falling of an object in a gravitational field (Newton's apple), the motion of a charged particle in an electric field, and the rapid motion of atoms or molecules when an object is at a temperature above zero Kelvin.

Despite these strict scientific definitions, common wood has potential energy. When exposed to enough heat and oxygen, its potential energy is converted to kinetic energy in the form of light and heat, commonly called fire. There are many other examples of potential and kinetic energy. To summarize, potential energy is energy *waiting* to do work while kinetic energy is energy actually *doing* work.

Energy Storage

Energy is stored in six basic forms:

Mechanical energy, the energy of motion (kinetic) and the energy of position (potential)

Chemical energy, the energy that bonds molecules together

Nuclear energy, the energy locked in the nuclei of atoms

Thermal energy, a kind of kinetic energy, the energy of moving and vibrating molecules

Radiant energy, energy that travels in waves like light, radio waves and x-rays

Electrical energy, a kind of kinetic energy; the energy of moving electrons

Energy Sources, Types and Issues

There are at least twelve major energy sources in use today: petroleum, natural gas, coal, uranium, propane, hydropower, geothermal, solar, wind, biomass, hydrogen and ocean wave. Our experiments in this course involve mostly solar and wind energy.

Energy sources are classified as renewable and nonrenewable. Renewable energy sources can be replenished in a short period of time, while nonrenewable sources take millions of years to form and their supplies are limited. Of those energy sources listed above, we can conveniently classify them into the following two categories:

- Renewable Energy - wind, solar, hydropower, geothermal, biomass, hydrogen and ocean wave
- Nonrenewable Energy - petroleum, natural gas, coal, uranium, propane

There are environmental, economic, and societal trade-offs in the use of all energy sources. Different people (and bureaucracies) place differing emphases on the relative importance of these factors. The availability and cost of energy are determining factors in the economic health and growth of societies. The sooner we can begin to use clean, non-polluting renewable energy as our everyday forms of energy, the sooner all of us will benefit from cleaner air and a stable climate.

Methods of Converting Energy to Electricity

There are basically six popular ways to convert energy to electricity:

- Mechanical – wind turning blades connected to a generator
- Chemical – batteries, fuel cells, anaerobic digesters (biomass)
- Combustion – burning fuel to generate steam (heat) to drive a generator
- Hydro – running water over the turbine blades connected to a generator
- Heat – creating steam which in turn operates a generator
- Photovoltaic – extracting electricity from the sun using semiconductor materials

As you may have already guessed, these methods overlap one another somewhat, but these are the common ways to generate electricity that routinely power our homes, cars and electronic appliances.

Methods of Generating Electricity

There are basically two ways to generate electricity – AC, alternating current and DC, direct current. Both have their particular advantages and applications to various electrical and electronic devices, as well as how each one can be distributed. Thomas Edison pioneered DC for lighting the streets of New York City but his hired assistant, Nikola Tesla, demonstrated that AC was a better choice for transmitting power over long distances. Both alternating and direct current had been used for arc lights, and both could be used for incandescent lamps. However, in the early 1880s, motors could function effectively only on DC. There was an expectation that electricity could be stored in batteries during off-peak hours, and this was possible only with DC. Finally, there was evidence that at the same voltages, AC was more dangerous than DC. All of this led Edison to prefer a DC system.

General Definitions of AC and DC

The term AC refers to “alternating current” while DC refers to “direct current”. So what’s, or watts, the difference (we’ll get to watts in a minute ;)

Direct current, or DC, is an electric current that flows “steadily” in one direction from positive to negative (current flow) or from negative to positive (electron flow). A common battery is an example of a device that produces direct current.

Alternating current, or AC, is an electric current that flows in a positive, then negative direction and repeats, usually at a fixed rate or frequency. Normal 110 VAC (volts alternating current), 60 Hz household current is an example of alternating current.

Watts and Hertz

No, not an act in Vegas or the name of a law firm, Watts and Hertz refer both to men of science and to electrical measurements named after them.

James Watt (1736-1819) was a Scottish engineer and inventor whose improvements in the steam engine led to its wide use in industry. The power output of the steam engine was thus measured in “watts.” and, when applied to electricity, 1 watt (W) corresponds to the power generated by 1 amp of current flowing through 1 ohm of resistance.

Gustav Hertz (1857-1894) was a German physicist credited with the discovery of creating electromagnetic waves artificially. In effect, Mr. Hertz was the father of all wireless communications in use today. While it used to be called “cycles per second” (CPS) or the number of times an AC wave went from positive to negative and back again, the term “hertz” and the symbol “Hz” was adopted to replace CPS in the late 1960’s in honor of Hertz’s contributions to measuring and understanding alternating waves. One Hz represents a frequency of one cycle of any AC wave in one second, regardless of the wave’s amplitude.



The **watt (W)** refers to the rate of dissipation of electrical energy. 1 watt corresponds to the power generated by 1 amp of current flowing through 1 ohm of resistance.

The **hertz (Hz)** is a unit of measurement for frequency of a repetitive wave form. A waveform with the frequency of 1 Hz would complete its cycle 1 time per second.

AC versus DC

An important advantage for AC became apparent with the invention of the transformer in 1883. This meant that the voltage from an AC generator could be efficiently increased for transmission and then decreased at the other end for use in the home or factory. From our previous explanation, electrical energy is proportional to voltage times current, so that boosting the voltage means that the same amount of energy can be transmitted with less current flow. Since heat produced in the line is a function of the current and the resistance, less current means fewer losses due to heat. For short distances of a mile or so, this made little difference. But for long distances, it would be critical.

The Westinghouse and Thomson-Houston companies preferred AC, and their faith was justified when Nikola Tesla invented a practical AC motor in 1888. Additional Tesla polyphase patents made AC systems more efficient, and these patents were used by Westinghouse at Niagara Falls in 1895. During the 1880s a sometimes fierce (and not always logical) battle was waged between proponents of AC and of DC. Edison himself became less involved as he devoted more time to his new laboratory at West Orange, New Jersey, after 1886, and as he became more involved with his iron-ore project. The Edison and Thomson-Houston companies merged in 1892 to form General Electric.

Benefits of AC

The primary benefit of alternating current, AC, is that it can be distributed over long distances with fewer losses as compared with direct current. Additionally, the wire sizes can be significantly reduced due to the reduced current load required to carry kiloamperes. Modern power plants and the towers that carry their power use a variety of voltages and sub-stations to route electricity to your home. As we mentioned before, electricity coming out of the typical wall sockets in your home is AC, measuring 110 V with a frequency of 60 Hz. This alternating current can directly power household appliances like toasters, irons, hair dryers and small electric hand tools.

Benefits of DC

Direct current has the advantage of a stable, constant voltage source, and the ability to be stored in batteries. Unlike alternating current that must constantly cycle from voltage peak to zero to negative voltage peak and back to zero again, direct current produces constant voltage levels that are advantageous for everything from starting your family car to powering your portable radio; not to mention your laptop computer or video game. And with today's microelectronic devices that use ever-lower voltages, down to 1.00 VDC (volts direct current) and going lower, DC is certainly the preferred choice.

Methods of Converting AC to DC and vice versa

Regardless of the individual benefits of each form of AC and DC electricity, the need to convert from AC to DC and vice versa literally defines an entire realm of the power electronics industry.

AC to DC – Power Supplies

Power supplies convert AC voltages to DC. For example, your television set uses both AC and DC voltages. AC voltage from the wall outlet is converted to DC by way of a

power supply inside the television. DC voltages then power the electronics that make the television work. The same is true with your desktop computer. It also has a power supply that takes in 110 VAC (volts alternating current) at 60 Hz and converts it to several levels of DC voltages with +12 VDC, +5 VDC and – 5 VDC being the most common. There are also two basic forms of power supply technologies – linear and switching – and both have their own particular advantages. Your desktop computer power supply is of the switching variety. In this course we study a very basic linear power supply, however most modern electronics are powered by switching power supplies that use embedded microcontrollers to convert AC to DC successfully.

DC to AC – Inverters

Somewhat less common in the average household is a device called an inverter. Inverters convert DC voltages to AC voltages; just the reverse of a power supply. But for what purpose you might ask? There are many applications for inverters with one application being powering your AC appliances from a battery. For example, if you do any camping and want some of the comforts of home in terms of hair dryers and coffee pots, an inverter will convert the 12 VDC from your car or SUV battery to 110 VAC, 60 Hz power to drive these devices. A more practical use of an inverter is a UPS (not the package delivery company), which stands for Uninterruptible Power Supply. By connecting a UPS between your AC wall plug and your desktop computer, your computer keeps working even when the AC power to your home fails. It does this by quickly switching from wall AC power to UPS generated AC power (from the DC battery), usually within less than one cycle of 60 Hz AC (or less than 16 milliseconds).

Methods of Storing Electricity

While DC electricity can be conveniently stored for later use, AC cannot, which is the primary shortcoming of AC electricity. There are many storage devices for DC electricity. The most popular DC storage device is the battery, which is a chemically based electric storage device. Batteries now come in all sorts of physical forms and chemical compounds. Some of the most common are lead acid (car battery), nickel cadmium (NiCad), nickel metal hydride, and lithium, to name some common types used in laptop computers, cell phones and video cameras. Each one has its particular advantages to the technology it powers.

ABOUT THE EXPERIMENTS IN THIS BOOK

Before launching into Experiment #1 in Chapter 2, it will be better if you first understand what is involved in the setup of all five experiments. After reading this section, you will come away with a much better appreciation of how we structured the code you will use to execute the experiments.

To establish a basis for our coding structure, we will treat the five interrelated experiments as one grand experiment, where each experiment shares some components of the whole. To do this effectively, we need to design a plan to successfully accomplish this task. For example, you wouldn't start building a house without a detailed plan to build the entire house. Imagine just building individual rooms that "might" come together later on with some luck, nails and duct tape! Then neither should you begin a series of interrelated experiments without a well thought-out plan. Scientists and engineers rarely just start experimenting without first considering the scope of the work to be done, as well as the procedures that will be carried out to correctly accomplish each experimental task. So, to stay in line with what the pros do, let's review what we want to accomplish and how each experiment will proceed.

First let's examine the name and intent of each of the five experiments:

- Chapter 2: Experiment #1, Programmable Battery Charger
- Chapter 3: Experiment #2, Dueling Solar Cells
- Chapter 4: Experiment #3, Solar Cell Sun Tracker
- Chapter 5: Experiment #4, Half and Full Wave Rectification
- Chapter 6: Experiment #5, Three-Phase AC Alternator

It may not be evident at this point but as we said earlier, each experiment draws on components of the whole. Therefore, it is vital for you to carefully follow directions and not deviate too much from the plans that are presented, including the order in which they are presented. There will certainly be enough opportunity to work "outside the box" in each experiment, but it is important to follow the detailed directions that serve as the foundation and structural framework of this entire experimental process.

To help visualize this process, think of each experiment as a separate floor in a five-story building. To make the building strong enough to support each floor, one needs to start with the correct foundation and then build the superstructure to hold each floor. In other words, you can't proceed to the next floor until the first floor is finished. Similarly, you can't jump to the Solar Cell Sun Tracker experiment unless the previous A/D Converter

and Dueling Solar Cells experiments are built, programmed and tested. And that's just what this chapter is about; building the foundation and superstructure for each experiment, in the correct order, so that when you're done your final experimental results will stand strong and tall!

Major Components

Before we start on our software foundation and superstructure, let's examine the building materials we have to work with.

The BASIC Stamp 2 Microcontroller Module

At the core of each experiment is the Board of Education® carrier board along with the BASIC Stamp® 2 microcontroller module (sold separately). The Board of Education carrier board has an ample solderless breadboard prototyping area for all the components that will be used in all five experiments.

To complete the experiments in this text, you will need to have your BASIC Stamp Editor v 2.0 or higher installed and running on your PC. Then, you will need to have your BASIC Stamp 2 module installed on your Board of Education carrier board and hooked up to your computer with a serial cable. You will need to supply power to this board with a fresh 9 volt battery or a 6-9 volt wall-mount power supply. If you need detailed directions, please see Chapter 1 of *What's a Microcontroller?* available on the Parallax CD, or free to view or download online at www.parallax.com. If you are using a public PC, remember to request authorization to the network administrator or your instructor before installing the software and connecting to your BASIC Stamp hardware.

A complete listing of the system, software, hardware and additional items required for all of the experiments can be found in Appendix D.

The Experiments with Renewable Energy Hardware

The solar cells are a major component of the Renewable Energy building materials. These are used for the direct current experiments. Each solar cell generates approximately 0.4 volts at 100 mA (milliamperes) under no load. Take a look at them now. There are eight of them and they fit into a convenient holder that allows you to hook them up in a number of series and parallel arrangements. Don't worry about what appear to be damaged cells with lines and streaks; this is how they come. Also look at the back of each solar cell to see the + (positive) and - (negative) screw terminals. When the cells are placed in the enclosure you can also view the + and - terminals through the holes

provided on the back (a very nice touch). In addition to the solar cells, a DC motor and a plastic propeller are provided to act as a load on the cells and the rechargeable batteries.

For the alternating current experiments, we will use the three-phase AC alternator along with the accompanying 3-blade wind turbine. With just a slight breeze the wind turbine will drive the AC alternator to produce upwards of 3 to 4 volts of rectified DC voltage. And that's what the first AC experiment is all about; that is, learning about half-wave and full-wave rectification. The final AC experiment will demonstrate the advantages of three-phase versus single-phase power.

To round out the building materials, you will find two AAA nickel cadmium batteries, a battery holder, a variety of resistors, capacitors and integrated circuits that act as the glue that holds everything together. Again, refer to Appendix D for the full parts list.

So now that you know what you have to work with, let's get started on building our foundation and superstructure; a software version, that is.

ACTIVITY #1: BUILDING A SOFTWARE FOUNDATION AND SUPERSTRUCTURE

Like any good building large or small, the key to its success and longevity lies in laying a solid foundation. Following that, building the superstructure that will hold the walls and floors is the next step. This analogy between building a physical structure and building a software structure is actually quite close. Just like throwing up a lean-to in a hurry for immediate shelter, you can also hack some code together with equal ease and less-than-permanent results. The term "hack" is computer lingo for doing something fast and quick in either hardware or software. As a matter of fact, some programmers, technicians and even degreed engineers never seem to get past the hacking stage. The end result of all their work, of course, is often replete with errors and inefficiencies. While it's okay to hack here and there, it is far better to learn the correct way of doing things in the first place. And this is exactly what you are about to learn; how to design a software program that will bear the load of what it supports. So let's get started!

Example Program: Template2.5.bs2

First we will take a look at the PBASIC code listing below. It will serve as both the foundation and superstructure for your code in the remaining part of this section and the experiments that follow.

- ✓ Enter the program into your BASIC Stamp Editor.
- ✓ Save the program under the name Template2.5.bs2. If you are working in a classroom environment, follow your instructor's directions for where to save your file.

```
' Experiments with Renewable Energy v1.0 - Template2.5.bs2

' Template for Experiments with Renewable Energy programs.
' {$STAMP BS2}
' {$PBASIC 2.5}

'-----
' Declarations
'-----

'-----
' Main Routine
'-----

Main:
DO
    GOSUB Exp_1                      ' Programmable Battery Charger
    GOSUB Exp_2                      ' Dueling Solar Cells
    GOSUB Exp_3                      ' Solar Cell Sun Tracker
    GOSUB Exp_4                      ' Half&Full Wave Rectification
    GOSUB Exp_5                      ' Three-Phase AC Alternator
    GOSUB Plot_It                     ' Plot Data w/ StampPlot Pro
LOOP

'-----
' Subroutines
'-----

'-----
' Experiment 1: Programmable Battery Charger
'-----

Exp_1:
Exp_1_End:
    RETURN

'-----
' Experiment 2: Dueling Solar Cells
'-----

Exp_2:
```

```
Exp_2_End:  
    RETURN  
  
'-----  
'   Experiment 3: Solar Cell Sun Tracker  
'-----  
  
Exp_3:  
  
Exp_3_End:  
    RETURN  
  
'-----  
'   Experiment 4: Half and Full Wave Rectification  
'-----  
  
Exp_4:  
  
Exp_4_End:  
    RETURN  
  
'-----  
'   Experiment 5: Three-Phase AC Alternator  
'-----  
  
Exp_5:  
  
Exp_5_End:  
    RETURN  
  
'-----  
'   A/D Converter Routine  
'-----  
  
A2D:  
  
A2D_End:  
    RETURN  
  
'-----  
'   Plot Acquired Data Using StampPlot Software  
'-----  
  
Plot_It:
```

```
Plot_It_End:  
RETURN
```

You may download the source code program files for the experiments in this text free from www.parallax.com. From the Parallax home page, choose Education, then Stamps in Class tutorials, then the title of this text. Available downloads will be linked on this products page.



Each Experiment adds to this template program. Each experiment will contain directions for adding code to the Declarations and Subroutines sections of the previous experiments program, filling in this template. Some experiments will require you to comment or uncomment some lines of code. You may follow the directions as shown, or download the program for each experiment. The final program that incorporates all of the experiments is included in Appendix B.

If you load this program into your BASIC Stamp module, it will actually run. It won't do much that you can see or measure, but it will run like any of the other programs that you will code in this course. Once again, take a look at the above listing that will serve as our software foundation and superstructure. You will add much more code to it as we go forward, but this is where everything has its start.

How Template2.5.bs2 Works

Beginning with the first few lines, let's examine each major part of this code just to see how close we came to building a solid software foundation and superstructure. The first two are shown below:

```
' Experiments with Renewable Energy v1.0 - Template2.5.bs2  
' Template for Experiments with Renewable Energy programs.
```

These are really nothing more than comments. Comments do not consist of executable code; they are there for reference. However, correct and appropriate comments are just as important as code in a well-structured program. Comments in PBASIC always begin with a single quote symbol (') and anything that follows the single quote to the end of the line is considered a comment. The first line simply indicates the title of this text (*Experiments with Renewable Energy*), the version number (1.0), and the file name as it is stored on disk. The file extension is a reminder that this program was written for the BASIC Stamp 2 microcontroller module. As this text is updated in the future the version number shall be updated also.

The next two lines look like comments but they are actually Complier Directives. These are commands for the PBASIC interpreter, the part of the BASIC Stamp module that takes in the English-like PBASIC commands and converts them into machine code that the microcontroller can understand.

```
' {$STAMP BS2}  
' {$PBASIC 2.5}
```

The first one is the \$STAMP directive, which informs the PBASIC interpreter that a BASIC Stamp 2 module is the target of this program. The second is the \$PBASIC directive, which allows the programmer to specify the syntax level for the compiler to use. Since compiler directives also start with a single quote, they are the exception to the rule for comments. Rather than typing them in, these compiler directives should be inserted into your program by using their toolbar icons. To learn more about complier directives, see the Help menu of your BASIC Stamp Editor.

The next three comment lines define a very important part of the coding process.

```
'-----  
' Declarations  
'-----
```

This process is called declaring your variables and constants for the underlying code. Modern coding techniques, especially those in other languages like C, Perl and even other varieties of BASIC, demand that you define your variables and constants BEFORE you use them in your code. This is a requirement for the PBASIC language, as well. You will see ample examples of these declarations very soon, so let's continue.

The Professional Use of Comments

Professional programmers often use comments liberally for several important reasons.

First, comments help both the programmer and users of the software to better understand the flow of the program and the functions of the code. Next, comments are absolutely necessary for jogging one's memory days, weeks or months after the code has been written. Unless you have an encyclopedic-type mind (and there are some of us that do, but I'm not one of them) you will naturally forget why you used a particular set of coding instructions to perform a particular task. The bottom line, then, is to use comments that add value to the accompanying code.



And one more thing about comments; keep them current with your code changes! Don't change your code and then forget to change your comments accordingly! The worst thing to do is to come back to your code weeks later to try and understand what you did. If your comments don't match the code, you will be immediately misled as to what is supposed to be happening with the code. Are the comments correct and the code wrong? Or is the code correct and the comments wrong? Worse yet, consider the innocent user of your program who will puzzle fruitlessly as to what the code is supposed to do. Comments are as important as the code itself. Make it a habit to comment your code correctly and you will never be sorry for it.

Now to the primary part of our software foundation, the Main Routine section (below).

```
' -----
' Main Routine
' -----
Main:
DO
    GOSUB Exp_1          ' Programmable Battery Charger
    GOSUB Exp_2          ' Dueling Solar Cells
    GOSUB Exp_3          ' Solar Cell Sun Tracker
    GOSUB Exp_4          ' Half&Full Wave Rectification
    GOSUB Exp_5          ' Three-Phase AC Alternator
    GOSUB Plot_It        ' Plot Data w/ StampPlot Pro
LOOP
```

While sparse in terms of actual code, the Main Routine section forms a very powerful and flexible foundation for the superstructure of our program. The superstructure, then, consists of three comment lines to identify this section of the program, followed by the **Main** label and six subroutine calls nested in a **DO...LOOP** code block. At each **GOSUB** command, the program will branch to the subroutine with the corresponding label, execute the instructions within it, and then return to the next statement in the **Main** routine.

There are six subroutines, one each for each of the five experiments, and a final subroutine that plots the voltage values gathered from the experiments. After the last subroutine, the **LOOP** command returns the program back to **DO**, and the subroutines are called in sequence all over again, repeating the experiments and updating the plot with each pass through the **Main** routine until the program is interrupted by the user (you).

Just as you sometimes want to go visit a building without spending time on every floor, you may want to run an experiment without running the experiments that come before or after. You can conveniently skip over an experiment by placing a single quote (') in front of its **GOSUB** command, thereby turning the command into a comment. In our series of experiments we will do this from time to time, to make more efficient use of breadboard and variable space. Refer to the BASIC Stamp Editor Help menu for a complete explanation of the **GOSUB** and **DO...LOOP** commands and all the other PBASIC instructions.

The next three lines are comments indicating that the remaining code consists of subroutines.

```
' -----
' Subroutines
' -----
```

Now look at the subroutine we currently have for Experiment 1:

```
' -----
' Experiment 1: Programmable Battery Charger
' -----
```

```
Exp_1:
Exp_1_End:
RETURN
```

The first three comment lines identify the subroutine's sequential order and function. The next line is the initial label for the subroutine, **Exp_1:**. Notice that labels always end with a colon (:) at the beginning of the associated subroutine. At this point, there is nothing between the initial label and the final label, **Exp_1_End:**, but you will fill this in with code once you get to Experiment 1. The very last instruction in each routine is the **RETURN** command. This causes the program sequence to branch back to the next

instruction, **GOSUB Exp_2**, following the calling of the **GOSUB Exp_1** instruction in the **Main** routine.

While seemingly simple in design, each of the five experiment subroutines and the **Plot_It** subroutine are built on the same coding framework. These simple, but elegant, software subroutines serve as placeholders for the code that will be inserted into them. And to expand on our software foundation/superstructure analogy, think of the first label **Exp_1**: as the ceiling and the last label **Exp_1_End**: as the floor of the first floor of our software building. What comes next is filling the floors with coding materials to make our experiments work.

ACTIVITY #2: PROGRAMMING THE STAMPPLOT GRAPHICAL PLOTTING SOFTWARE INTERFACE

After you have built the framework for your program, it's time to add the code for the **Plot_It** subroutine. In the upcoming experiments, you will be harnessing and analyzing energy. The results of your experiments will be analyzed by graphing the generated voltages on your computer screen. The **Plot_It** subroutine allows the StampPlot software to access these voltage values and graph them as they change over time. Once the **Plot_It** subroutine is in place we will install and test StampPlot Pro.

Example Program: PlotIt.bs2

- ✓ In your BASIC Stamp Editor, open Template2.5.bs2, if it is not already open.
- ✓ Select File, then Save As. Rename the file PlotIt.bs2.
- ✓ Update the comments at the beginning of the program to read:

```
' Experiments with Renewable Energy v1.0 - PlotIt.bs2
' Transmits acquired data to StampPlot software for plotting.
```

- ✓ Next, enter the variables that will be needed by the **Plot_It** subroutine into the Declarations section so it reads like this:

```
' -----
' Declarations
' -----
'
' ----- For Plot_It Subroutine -----
ch0      VAR     Byte          'Voltage reading from A/D CH0
ch1      VAR     Byte          'Voltage reading from A/D CH1
ch2      VAR     Byte          'Voltage reading from A/D CH2
ch3      VAR     Byte          'Voltage reading from A/D CH3
checkSum  VAR    Byte          'Sum of all of above readings
```

The `ch0`, `ch1`, `ch2` and `ch3` variables (which stand for Channel 0 through Channel 3) will be storing 8-bit A/D converted voltage values gathered from the experiments. That's why we specified a byte (8 bits) for variable storage. As a double check against the other four values, the StampPlot software uses the `checkSum` variable, which is also a byte.

- ✓ Now add the `Plot_It` subroutine code under the appropriate heading of the Subroutines section, so it reads as follows:

```
'-----  
' Plot Acquired Data Using StampPlot Software  
'-----  
  
Plot_It:  
    checkSum = ch0 + ch1 + ch2 + ch3          ' Compute checksum for StampPlot  
    DEBUG ch0, ch1, ch2, ch3, checkSum        ' Transmit data to StampPlot  
    PAUSE 250                                ' Give StampPlot time to plot data  
  
    ch0 = 0                                    ' Reset voltages and checkSum  
    ch1 = 0  
    ch2 = 0  
    ch3 = 0  
    checkSum = 0  
  
Plot_It_End:  
    RETURN
```

- ✓ From the menu bar, choose File and then Save to preserve your work.
- ✓ Load the file into the BASIC Stamp module by clicking on the Run icon in the BASIC Stamp Editor, or by pressing Ctrl-R.

Notice that the Debug Terminal pops up, and that its green RX LED flashes four times a second, or once every 250 ms, as programmed. The flashing RX LED shows you that the program is loaded and that your BASIC Stamp module is communicating with your PC.

How PlotIt.bs2 Works

The first instruction in the `Plot_It` routine computes the checksum of the four A/D voltage values. The second instruction uses the `DEBUG` command to transmit the four voltage values followed by `checkSum` to the PC via the COM port. The third instruction pauses our program for 250 ms (one fourth of a second) to allow StampPlot to plot the data. Finally, the program clears all these variables as a matter of proper cleanup, by setting `ch0` through `ch3` and `checkSum` equal to zero before returning to the `Main` calling

routine (`Plot_It_End`). Notice the use of comments to the right of some of the instructions. These comments explain what the code is doing and, also, act as memory joggers for future reference.

Now here's a little more explanation about computing checksums. Since each `ch0` through `ch3` variable can hold a value between 0 and 255, adding all four values together may cause the `checksum` variable to overflow, leaving only a partial result. This is perfectly okay since the StampPlot software will also compute its own checksum in the same way. If our checksum matches with StampPlot's computed checksum, StampPlot will graph the `ch0` through `ch3` values. Otherwise, it will flag an error and not plot anything.

- ✓ Now make sure you have saved your program, then close the BASIC Stamp Editor.

StampPlot Software

Before going too much further it would be best to concentrate on the StampPlot Pro graphical plotting software that will add a tremendous visual aid to the experiments. We will be using the StampPlot Pro version with the Free Home/Educational Standard License.

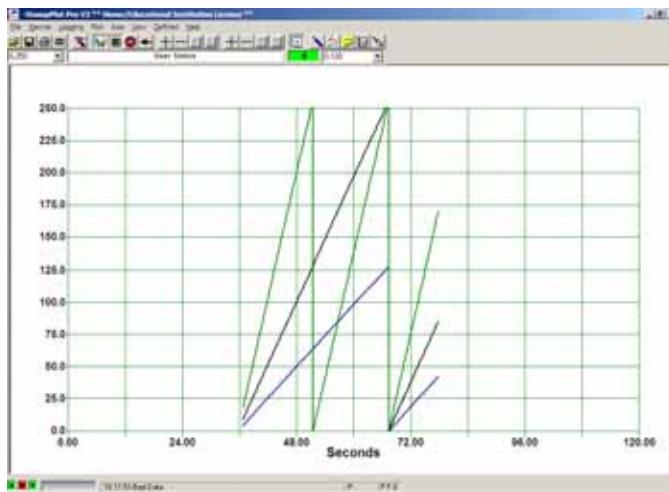


Figure 1-1
StampPlot Pro Example
Graph

Developed by two professors, Martin Hebel and Will Devenport at Southern Illinois University, the StampPlot graphic plotting software brings the voltages produced in the experiments alive! Figure 1-1 shows just one example of a single channel voltage plot. StampPlot is also fully programmable using the macros that we have provided for you.

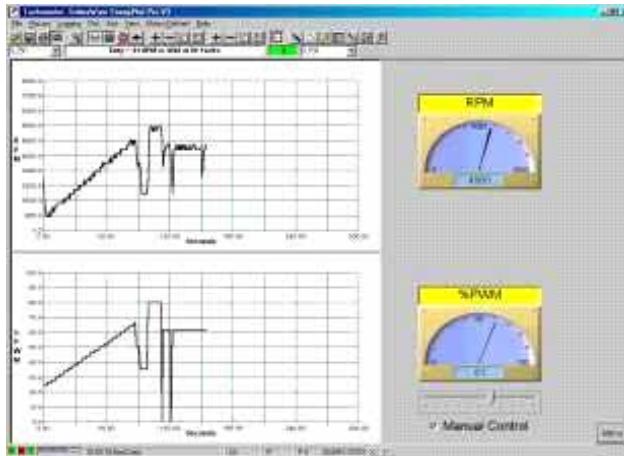


Figure 1-2
StampPlot Features

Besides the standard graphic plots of up to ten analog channels of data, you can also program virtual gauges that react to voltage readings just like actual meters.

Figure 1-2 show just one of the major features of the StampPlot graphing software that we will use in our upcoming experiments. With StampPlot Pro you can:

- Run the provided macros that control the look and feel of your plots
- Send commands via the BASIC Stamp module to control plotting
- Draw gauges and meters on your plots
- Sound alarms when voltage readings fall or rise above preset limits
- Play WAV files at preset voltage limits

The Free Home/Educational Standard License version of the StampPlot Pro software and the macros needed for this text are available on the Parallax CD that comes with the Experiments with Renewable Energy kit. WinZip is also included on the Parallax CD. (If you are in a classroom environment, check to see if you already have StampPlot Pro installed on your computer; follow the directions of your instructor.)

- ✓ Load the Parallax CD
- ✓ From the Welcome menu, choose Software.
- ✓ Click the plus sign next to BASIC Stamp.

- ✓ Click on StampPlot Pro.
- ✓ Click on Install, and follow the directions.
- ✓ Repeat the procedure to install the StampPlot Pro Energy Manual Macros

You may also download the StampPlot Pro software and the macros for these experiments free from the Parallax website. If you need WinZip to open these files; a free demo version can be obtained from www.winzip.com.

- ✓ Go to www.parallax.com.
- ✓ From the Education menu, choose Stamps in Class Tutorials.
- ✓ Click on Experiments with Renewable Energy.
- ✓ Scroll to the bottom of the page.
- ✓ Click on the link to install StampPlot Pro, and follow the installation instructions.
- ✓ Repeat the procedure to download the StampPlot Pro Energy Manual Macros.

Once your software is installed, you will need to register it.

- ✓ Open StampPlot
- ✓ Select Register from the menu bar
- ✓ Select Free Home/Educ Standard License
- ✓ A box will appear informing you that this is authorized ONLY for home and educational use. Click Yes.
- ✓ Another box will appear thanking you for registering. Click OK.

The Home and Educational version of StampPlot Pro is free. Its main limitation is that it does not allow you to create custom macros (macros are like mini programs). You can still run macros with this version, and all of the macros needed to complete the experiments have been included on the Parallax CD and website.

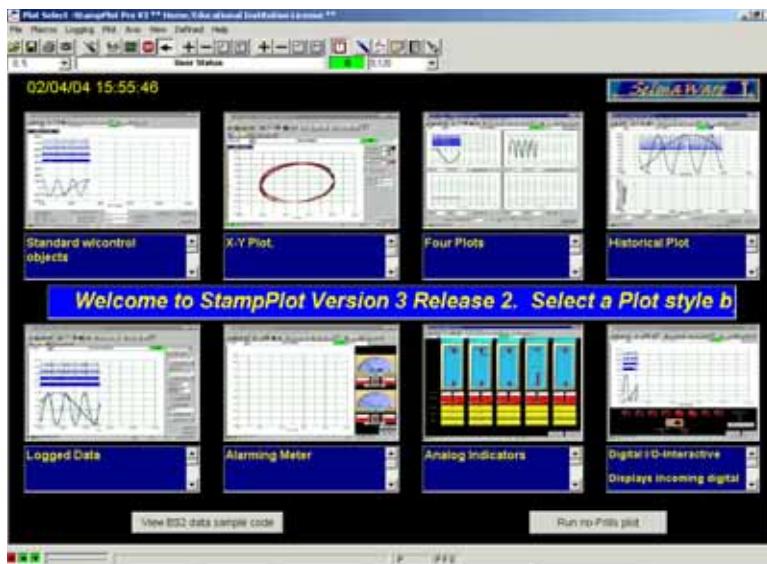


Figure 1-3
StampPlot Pro
Welcome Screen

After you have successfully installed the StampPlot software and the Experiments with Renewable Energy macros, it's time to utilize the first macro. The macros customize the graphics in the plotting area to best display the results of each upcoming experiment. There are two different ways to access these macros. One way is to open up StampPlot with a specific macro through the Start menu.

- ✓ From your computer's Desktop, click the Start button
- ✓ Choose Programs.
- ✓ Choose Parallax Inc.
- ✓ Choose StampPlot.
- ✓ Choose Experiments with Renewable Energy.
- ✓ Click on sic_ewre_basic.spm.

The StampPlot program should open automatically and switch to the screen shown in Figure 1-4. If instead you get an error message asking which program to use to open the macro, try this:

- ✓ Cancel out of the error message window.

- ✓ Open StampPlot from the desktop icon, or by clicking on Start → Programs → StampPlot.
- ✓ From the toolbar, choose File.
- ✓ Choose Update Associations for .spm, .plt.
- ✓ Close StampPlot, and try reopening from the Start button through the macros again.

Or, if you StampPlot is already open and you want to change from one macro to another:

- ✓ On the top menu bar of the StampPlot program, click on Macros.
- ✓ From the drop-down menu, click on Select Start-Up macro.
- ✓ In the Open window, double-click the folder named SIC-Energy.
- ✓ Click on the file sic_ewre_basic.spm, then click on the Open button.
- ✓ Click the Yes button when asked “Start a new plot using this macro?”

Either way you chose, your screen should resemble the one shown below in Figure 1-4.

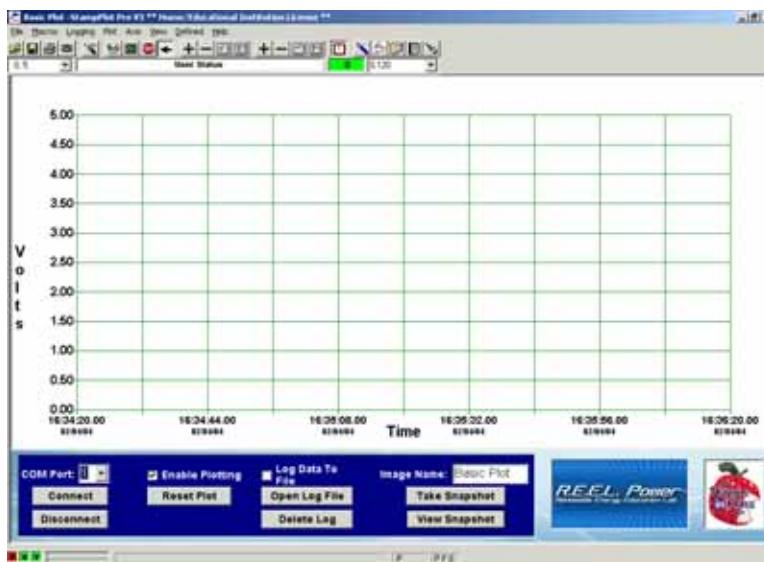


Figure 1-4
The StampPlot
Pro Start-Up
Screen for the
Experiments with
Renewable
Energy Text.

ACTIVITY #3: TESTING THE STAMPPLOT GRAPHICAL PLOTTING SOFTWARE INTERFACE

In this activity, we will use the PlotIt.bs2 program to create a graph in StampPlot. This will allow us to test both our program and our connection to StampPlot.

- ✓ Open PlotIt.bs2 in the BASIC Stamp Editor.
- ✓ In the `plot_it` subroutine, change the values of `ch0` through `ch3` as shown below.

<code>ch0 = 0</code>	changes to	<code>ch0 = 25</code>
<code>ch1 = 0</code>		<code>ch1 = 50</code>
<code>ch2 = 0</code>		<code>ch2 = 75</code>
<code>ch3 = 0</code>		<code>ch3 = 100</code>
<code>checkSum = 0</code>		<code>checkSum = 0</code>

- ✓ Note that the `checkSum = 0` statement remains unchanged.
- ✓ Download the modified code to the BS2 by clicking on the Run icon or by typing Ctrl-R. You should see the Debug Terminal pop up with the RX LED flashing and some gibberish scrolling across the screen. These are actually the ASCII symbols for the numbers that you just changed above.
- ✓ Check the top of the Debug Terminal to note which COM Port is being used.
- ✓ Now close the Debug Terminal (but not the entire BASIC Stamp Editor). This frees up the COM port for StampPlot.
- ✓ Open StampPlot again, if it isn't already open.
- ✓ In the lower left corner, set the COM Port to the same one that was being used by the BASIC Stamp Editor's Debug Terminal.
- ✓ Click on the Plot icon at the top of the screen (it looks like a small sine wave graph on the tool bar). The Enable Plotting box at the bottom of the screen should be checked.
- ✓ Click on the Connect button in the lower left corner (or the handshake icon on the toolbar).

You should see four straight lines in different colors being plotted across the screen at 0.5., 1.00, 1.50, and 2.00 volts, as shown in Figure 1-5. If you do, then you have reached a major milestone in your programming efforts. From here on you will be able to use StampPlot for all the other experiments as well as for testing the A/D converter software that comes next.

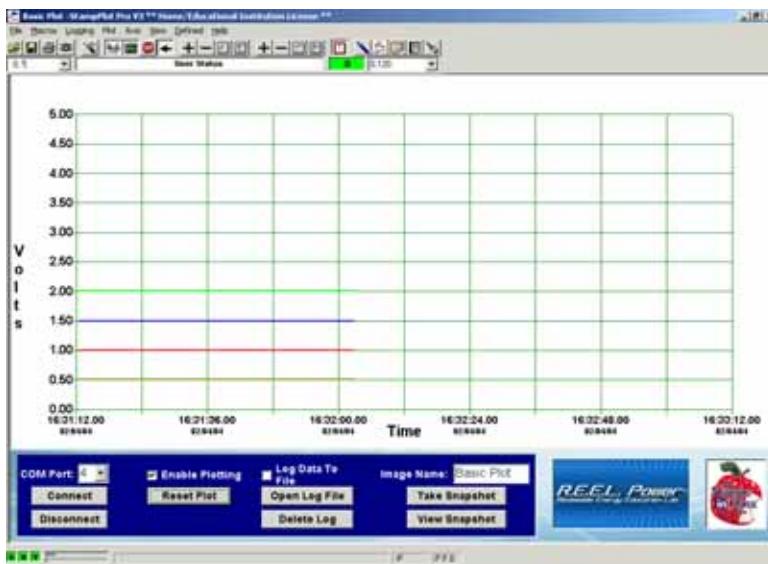


Figure 1-5
Test Plot of All
Four Channels

- ✓ Return to the BASIC Stamp Editor and close the PlotIt.bs2 program without saving the changes. When you use the program again, the value of the `ch0` through `ch3` variables will need to be zero, as they were originally.
- ✓ Congratulate yourself for a job well done up to now!

ACTIVITY #4: BUILD, PROGRAM AND TEST THE 4-CHANNEL A/D CONVERTER

The next task prior to getting into our energy experiments involves building, programming and testing the 4-channel A/D converter circuit. This is where the `ch0` through `ch3` values from the `Plot_it` subroutine come in. They represent the four analog voltages that the A/D converter converts into binary values. The 4-channel A/D converter selected for our experiments is the National Semiconductor ADC0834.

The ADC0834 - A 4-Channel Analog to Digital Converter

The ADC0834 is an 8-bit analog to digital converter (A/D converter) with 4-channel synchronous serial output. The information box below will explain what each of these terms mean.

Analog to Digital Converter (A/D converter or ADC for short) A device that measures an analog voltage sample and returns a binary number that describes the sample.

8-bit: The number of binary digits the ADC0834 uses to describe the analog voltage it samples. 8-bit is also the resolution of the A/D converter. You can count from 0 to 255 (decimal) using an 8-bit binary number. This means that the ADC0834 can approximate the voltage it measures as one of 256 levels. A higher resolution converter, such as 12-bit, would break the same voltage range into 4096 levels because you can count from 0 to 4095 with 12 binary bits.

Synchronous serial means that data bits are sent one after another along a single data line, in a time-coordinated manner (in sync). The ADC0834 depends on a clock signal sent by the BASIC Stamp to time the sending of each serial output bit.

4-Channel means that four different analog voltages can be input to this A/D converter, one to each channel. Any channel can be converted, but only one channel can be converted at a time. The BASIC Stamp module sends signals to the ADC0834 to let the ADC0834 know which channel to convert.

Take a peek ahead at the schematic in Figure 1-6 on page 30. The BASIC Stamp module must be wired to the ADC0834, and then programmed to send binary control signals to it. The PBASIC program will make the ADC0834 do its job, and will allow the BASIC Stamp module to read and store the 8-serial-bits transmitted by the ADC0834. The notation for the ADC0834's inputs and outputs that we will be using are shown in Table 1-1 below.

Table 1-1: Measured voltages during charge cycle

CH0	Channel 0 Analog Input
CH1	Channel 1 Analog Input
CH2	Channel 2 Analog Input.
CH3	Channel 3 Analog Input
V+	Voltage supply
VREF	Reference Voltage
AGND	Analog Ground
DGND	Data Ground
DO	Data Out (the serial data output)
DI	Data In (the serial data input)
CS	Chip Select (active low)



Why are we using V+ instead of Vcc? If you're wondering why the ADC0834 A/D Converter chip's Vcc pin (pin 14) is not used, then please refer to page 18 of the manufacturer's data sheet for this device, which can be downloaded from National Semiconductor's web site www.national.com. Here you will see that an internal diode goes from V+ (pin 1) to Vcc along with a zener diode going to ground, which provides for a better degree of voltage regulation to the internal circuitry. By supplying voltage to the V+ pin in this way, the Vcc voltage level will always be guaranteed to be at the correct voltage. That said, this selection is based on your author's personal preference, and you are free to wire Vdd directly to Vcc (pin 14) if you so choose.

Understanding Vref

The voltage applied to the reference input, Vref, defines the voltage span of the analog input, over which the 256 possible output values apply. When $V_{ref} = V+ = +5V$, the A/D converter's maximum voltage is set to +5V. The minimum value is 0V or ground.

Therefore the voltage spans 5V over a range of 256 possible values. Each of the 256 possible values, or levels, thus represents:

$$5 \text{ V} / 256 \text{ levels} = 0.0195 \text{ volts per level}$$

which, for practical purposes, we will round off to 0.02 volts.

For example, if the A/D converter returned a reading of 52, the voltage represented is:

$$52 * 0.02 = 1.04 \text{ V}$$

Your Turn

Calculate the voltages represented by the following A/D converter readings:

87
10
221
100

Controlling the ADC0834 from the BASIC Stamp

To prime the ADC0834 for taking a measurement, the CS pin has to receive a signal from the BASIC Stamp module that starts high, then goes low. This signal has to stay low for the duration of the conversion. Next, the BASIC Stamp module must tell the ADC0834

which channel to convert. This is done by sending a serial data code via the DI line. Then the CLK input must receive a single clock pulse to signify that the conversion should start at the next clock pulse. For this device, a clock pulse starts low, goes high, then goes low again. Once this happens, the reading will be sent via the DO (Data Out) line. It takes 8 more clock pulses to complete the conversion. Each time a clock pulse is received by the CLK input, another of the serial bits is sent by the DO output. When all 8 bits have been received by the BASIC Stamp module, it must set the CS pin high, signifying the end of the data transmission.

Electronics designers use data sheets to find the kind of information just discussed. Each IC manufacturer publishes data sheets for the integrated circuits they make. The information just covered on the pin map and control signals was condensed from a data sheet published by National Semiconductor, the maker of the ADC0834. Datasheets are available on the manufacturer's web sites.

Parts Required

- (1) A/D converter – ADC0834 DIP
- (8) Jumper wires

Building the A/D Converter



Schematics and wiring diagrams are provided for the majority of the circuits. You will be using the A/D converter circuits with others from each chapter. Please follow the parts placement EXACTLY!

- ✓ Disconnect power to your BASIC Stamp module; always do this before building or modifying circuits.
- ✓ Install the 14-pin ADC0834 DIP (Dual Inline Package) chip as show in Figure 1-7. Note that the DIP is oriented with the notch pointing down. This is done intentionally for reasons that will become clear shortly.
- ✓ Build the remainder of the circuit with the jumper wires.



On a DIP chip, the pins are numbered counterclockwise from the reference notch.

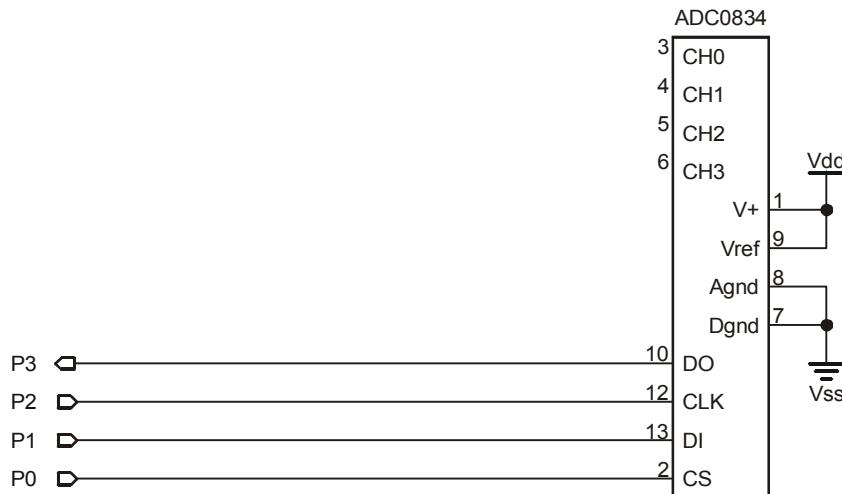


Figure 1-6
A/D
Converter
Schematic

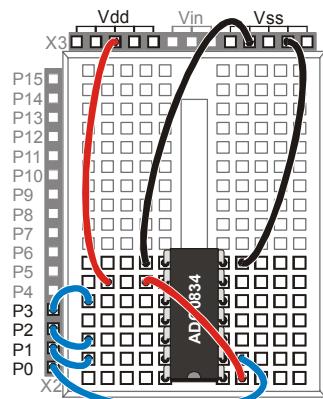


Figure 1-7
A/D Converter Wiring
Diagram



Double-check your wiring! Make sure that P3 is connected to pin 10 and not pin 11 of the ADC0834 chip! An accidental connection to ADC0834 pin 11 would cause false readings for output voltages in future experiments.

Choosing Correct Wire Colors

The Stamps in Class - *Experiments with Renewable Energy* kit comes with an assortment of colored wires, which you will use to hook up the various components for each experiment. The wire colors have a certain value in and of themselves that should not be ignored. Just as the colored bands on a resistor indicate the resistance and tolerance of the resistor, the wire colors can also communicate a particular function of the circuit that you create. For example, if you are able to use the following wire colors consistently for the functions listed below, you will be able to see at a glance where voltages, signals and grounds are located. This could avoid confusion and would make your experiments more interesting and rewarding:



Red – Vdd

Black –Vss

Blue – Port connections or connections between components

Yellow – Port connections or connections between components

These are not intended to be hard and fast rules for you to follow, but are presented as a suggestion for consistent wiring techniques. If you choose to follow the color code above, your circuits will be more attractive to the eye and will be much easier to understand in terms of what is connected to what.

Example Program: ADCtest.bs2

As before, in preparation for programming the A/D converter code you must first declare the variables this routine will use.

- ✓ Open and resave PlotIt.bs2 as ADCtest.bs2.
- ✓ Double-check the `Plot_It` subroutine to make sure the `ch0` through `ch3` values are set equal to zero; correct if necessary.
- ✓ Add the variables for the A/D Converter Routine to the Declarations section, which should read as shown below when you are finished. Note that the `Plot_It` subroutine variables remain in place.

```
' -----
' Declarations
' -----
'
' ----- For Plot_It Subroutine -----
ch0      VAR     Byte          'Voltage reading from A/D CH0
ch1      VAR     Byte          'Voltage reading from A/D CH1
ch2      VAR     Byte          'Voltage reading from A/D CH2
ch3      VAR     Byte          'Voltage reading from A/D CH3
checkSum  VAR     Byte          'Sum of all of above readings
```

```

' ----- For ADC0834 4-Channel Multiplexed A/D Converter -----
A2dChipSel PIN      0          ' A/D Converter Chip Select(P0)
A2dDataIn  PIN      1          ' A/D Converter Data Input(P1)
A2dClk    PIN      2          ' A/D Converter Clock(P2)
A2dDataOut PIN      3          ' A/D Converter Data Output(P3)

A2dMuxId0 CON      %1100      ' A/D MUX IDs for Ch 0..3
A2dMuxId1 CON      %1110      ' Bit 3 = 1, start bit
A2dMuxId2 CON      %1101      ' Bit 2 = 1, single (1)/diff (0)
A2dMuxId3 CON      %1111      ' Bit 1 odd / sign, Bit 0 select

a2dMuxId  VAR      Nib        ' A/D Channel MUX ID to shift out
a2dResult VAR      Byte       ' 8-bit result of A/D conversion

```

- ✓ In the Subroutines section, insert the following A/D Converter Routine code underneath its heading, like this:

How the A/D Converter Routine Works

This routine references names (aliases) defined in the Declarations section. The first four **PIN** directives give names to the BASIC Stamp I/O pins that identify which pin of the ADC0834 they are connected to. These aliases will be used in commands that enable communication between the two devices.

A2dChipSel	PIN	0
A2dDataIn	PIN	1
A2dClk	PIN	2
A2dDataOut	PIN	3

The four **CON** directives that follow are specially coded to let the ADC0834 know which channel, 0 through 3, that we want to convert. One of the first things the BASIC Stamp must do is send the appropriate MuxId to the ADC0834.

A2dMuxId0	CON	%1100
A2dMuxId1	CON	%1110
A2dMuxId2	CON	%1101
A2dMuxId3	CON	%1111

To accomplish this, the nibble sized variable **a2dMuxId** will be set to either **A2dMuxId0**, 1, 2, or 3, depending on which channel is to be converted. When the conversion is finished, the result will be stored in **a2dResult**.

a2dMuxId	VAR	Nib
a2dResult	VAR	Byte

The subroutine **A2D** begins by initializing the ADC0834 control signals. To start a conversion, the **A2dChipSel** (CS) pin must transition from high to low, so here, **A2dChipSel** is initially set high. **A2dDataIn** and **A2dClk** are then set low, to initialize the Data In and Clock pins to be low. Finally, **a2dResult** is initialized to 0.

```
A2D:
HIGH A2dChipSel
LOW A2dDataIn
LOW A2dClk
a2dResult = 0
```

At the label **A2D_Start_Conversion**, the command **LOW A2dChipSel** sends a low signal to the ADC0834's CS pin, thus telling the ADC0834 to begin a conversion. The **A2dChipSel** pin needs to stay low for the duration of the conversion.

```
A2D_Start_Conversion:  
    LOW A2dChipSel
```

Next, the BASIC Stamp module must tell the ADC0834 which channel to convert, done at the **A2D_Shift_Out_Channel_ID** label. The **SHIFTOUT** instruction sends the proper channel ID to the ADC0834.

```
A2D_Shift_Out_Channel_ID:  
    SHIFTOUT A2dDataIn,A2dClk,MSBFIRST, [a2dMuxId\4]
```

At the **A2D_Shift_In_Result** label, the next command, **PULSOUT A2dClk,10** will send a clock pulse that has the right shape, low-high-low.

```
A2D_Shift_In_Result:  
    PULSOUT A2dClk,10
```

PULSOUT A2dClk,10 sends a clock pulse to the ADC0834's CLK input. This is the first clock pulse, and all it does is tell the ADC0834 to start converting on the next clock pulse. Because of this, we don't need to check for input from DO after this first clock pulse.

Since we set the clock low initially, **PULSOUT** sends the desired low-high-low signal. The duration of the high segment is twice the number specified in the **PULSOUT** command, in microseconds (μ s). $1 \mu\text{s} = 1/1,000,000$ of a second. Therefore the duration of this high segment is $2 \mu\text{s} \times 10 = 20 \mu\text{s}$.

The following **SHIFTIN** command is a powerful instruction that takes care of all the synchronous serial communication.

```
SHIFTIN A2dDataOut,A2dClk,MSBPRE, [a2dResult\8]
```

In effect, this command sends clock pulses to the ADC0834's CLK input and reads output bits from ADC0834's DO output. This command also loads each of the ADC0831's output bits into the variable **a2dResult** byte. The **SHIFTIN** command is discussed in more detail in the BASIC Stamp Manual and the BASIC Stamp Editor's Help menu, but the general format for the command is:

SHIFTIN data_pin, clock_pin, mode, [variablebits]

In our case, the data pin is **A2dDataOut**, and the clock pin is **A2dC1k**.

The mode in this case is **MSBPRE**, and it's one of four transmission modes that can be used in this command. It indicates that the ADC0834's output bits are ready before the clock pulse's negative edge, the transition from high to low. It also indicates that the bits are transmitted in descending order, starting with the MSB (most significant bit). **[a2dResult \8]** means the data is shifted into the **a2dResult** variable, and 8-bits are expected.

Finally, the command

```
HIGH A2dChipSel
```

ends the data exchange with the ADC0834. The last label **A2D_End** is followed by the **RETURN** command that sends the program back into the **Main** routine.

These additions to the larger program will remain in place for all of the experiments. Before going directly to Experiment 1, it would be prudent and proper to test the A/D converter to see if it is performing correctly. The A/D converter is used in each one of the five experiments, so validating its performance here and now would be wise. Then you can be confident of proper converted voltage readings when you do get to the experiments (which will be soon if you've done everything right up until now).

Testing the A/D Converter

In order to begin testing the A/D converter you'll need to make a temporary modification to the Main Routine section.

- ✓ Modify the Main Routine section by adding three lines to the **DO...LOOP** code block, before the **GOSUB** commands, so it reads like this:

```
' -----
' Main Routine
'-----

Main:
DO
    a2dMuxId = A2dMuxId0          ' Set A/D to convert Ch 0
    GOSUB A2D                      ' Do the conversion
    ch0 = a2dResult                ' 8-bit result into ch0

    GOSUB Exp_1                    ' Programmable Battery Charger
    GOSUB Exp_2                    ' Dueling Solar Cells
    GOSUB Exp_3                    ' Solar Cell Sun Tracker
    GOSUB Exp_4                    ' Half&Full Wave Rectification
    GOSUB Exp_5                    ' Three-Phase AC Alternator
    GOSUB Plot_It                  ' Plot Data w/ StampPlot Pro
LOOP
```

This is what we will call a programming “hack”, since it violates our structured coding rules. However, it is an acceptable hack since it will help test the A/D converter. After we run our tests and know that it’s working, we’ll erase these three lines.

- ✓ Restore power to your BASIC Stamp module.
- ✓ Save and run the “hacked” code.

If successful, the Debug Terminal will pop up, showing the RX LED flashing and, possibly, some gibberish scrolling across the screen.

Additional Part Required

- (1) Resistor – 1 kΩ (brown-black-red)

Testing the A/D Converter’s Upper Limit – Channel 0

- ✓ Disconnect power to your BASIC Stamp module.
- ✓ Next, add a 1 kΩ resistor between pin 3 (CH0) and pin 1 (Vdd) of the ADC0834 as shown in Figure 1-8 and Figure 1-9. This will apply +5 VDC to the CH0 input channel.

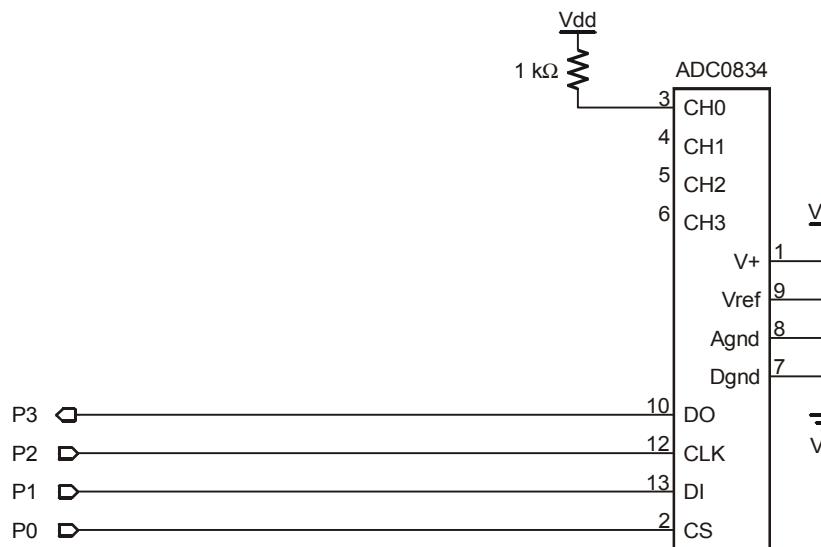


Figure 1-8
Testing The
A/D
Converter's
Upper Limit
Schematic

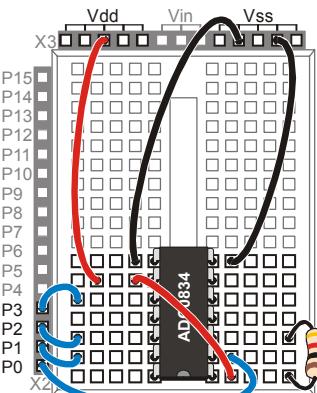


Figure 1-9
Testing The A/D
Converter's Upper Limit
Wiring Diagram

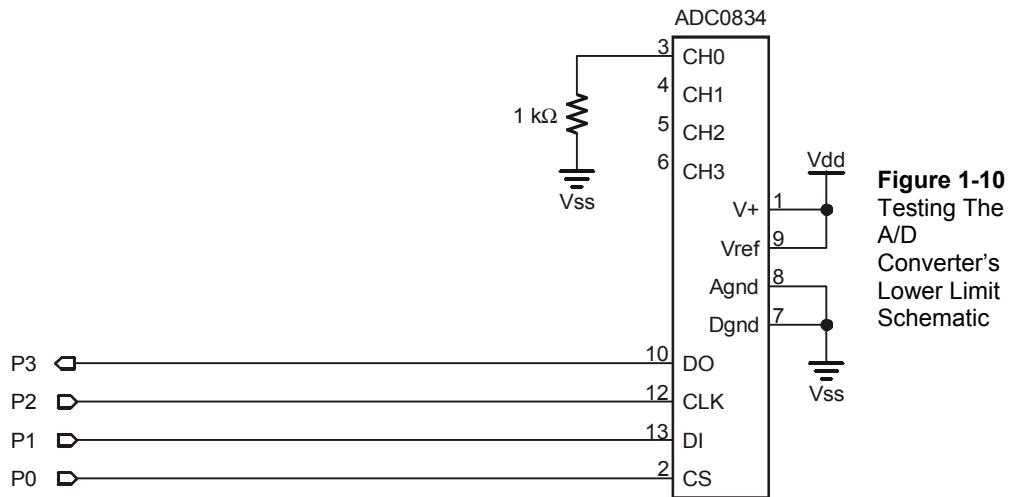
- ✓ Restore power to your BASIC Stamp module. (From here on we will assume you are in the habit of disconnecting power before altering circuits, and of course reconnecting power before using them!)
- ✓ Run the program ADCtest.bs2.

- ✓ Close the BASIC Stamp Editor's Debug Terminal so StampPlot can use the COM port.
- ✓ Bring up the StampPlot screen.
- ✓ Click on the Connect icon making sure that the Plot icon to the right is also selected.

You should now see the plot line marching across the screen at the 5-VDC level.

Testing the A/D Converter's Lower Limit – Channel 0

- ✓ Modify the circuit by moving the 1 kΩ resistor so it connects pin 3 (CH0) and pin 7 (Vss) of the ADC0834. This is shown in the schematic in Figure 1-10 and the wiring diagram in Figure 1-11.



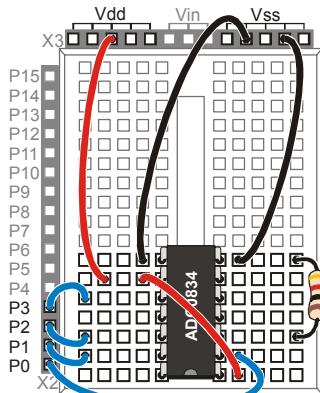


Figure 1-11
Testing The A/D
Converter's Lower Limit
Wiring Diagram

StampPlot should now display a marching line at the 0 VDC level. If this is what you see, then you have just tested Channel 0 of the A/D converter.



Each channel will display in a different color on the StampPlot graphs.

Channel 0 = Olive Channel 1 = Red

Channel 2 = Blue Channel 3 = Green

Your Turn – Testing the A/D Converter's Remaining Channels

Now you can perform these same two tests for the remaining three channels. For each channel, you will have to update the 3 lines of “hacked” code we inserted into the **Main** routine’s **DO...LOOP**, and you will have to reposition the 1 k Ω resistor for each test. As you run the modified code and plot the output of each circuit, you will be able to visually verify that the upper and lower voltage limits are the same for each channel.

Testing Channel 1

- ✓ Click on the Disconnect button in StampPlot.
 - ✓ In the BASIC Stamp Editor, update the three lines of code in the **Main** routine to test channel 1, as shown below.

```
a2dMuxId = A2dMuxId1          ' Set A/D to convert ch 1  
GOSUB A2D                      ' Do the conversion  
ch1 = a2dResult                 ' 8-bit result into ch1
```

- ✓ To test the upper limit, place the 1 kΩ resistor between pin 4 (CH1) and pin 1 (Vdd) of the ADC0834.
- ✓ Download the modified code, and then close the Debug Terminal.
- ✓ Bring up StampPlot, and click on the Connect button to verify the voltage on the graph, then click on Disconnect again.
- ✓ To test the lower limit, modify the circuit by moving the 1 kΩ resistor so it connects pin 4 (CH1) and pin 7 (Vss) of the ADC0834.
- ✓ Re-run the modified code, and then close the Debug Terminal.
- ✓ Bring up StampPlot, click on the Connect button to view the voltage graph again, and then click on Disconnect.

Testing Channel 2

- ✓ In the BASIC Stamp Editor, update the three lines of code in the **Main** routine to test channel 2, as shown below.

```
a2dMuxId = A2dMuxId2          ' Set A/D to convert ch 2  
GOSUB A2D                      ' Do the conversion  
ch2 = a2dResult                 ' 8-bit result into ch2
```

- ✓ To test the upper limit, place the 1 kΩ resistor between pin 5 (CH2) and pin 1 (Vdd) of the ADC0834.
- ✓ Download the modified code, and then close the Debug Terminal.
- ✓ Bring up StampPlot, and click on the Connect button to verify the voltage on the graph, then click on Disconnect again.
- ✓ To test the lower limit, modify the circuit by moving the 1 kΩ resistor so it connects pin 5 (CH2) and pin 7 (Vss) of the ADC0834.
- ✓ Re-run the modified code, and then close the Debug Terminal.
- ✓ Bring up StampPlot, click on the Connect button to view the voltage graph again, and then click on Disconnect.

Testing Channel 3

- ✓ In the BASIC Stamp Editor, update the three lines of code in the **Main** routine to test channel 3, as shown below.

```
a2dMuxId = A2dMuxId3          ' Set A/D to convert ch 3  
GOSUB A2D                      ' Do the conversion  
ch3 = a2dResult                 ' 8-bit result into ch3
```

- ✓ To test the upper limit, place the 1 kΩ resistor between pin 6 (CH3) and pin 1 (Vdd) of the ADC0834.
- ✓ Download the modified code, and then close the Debug Terminal.
- ✓ Bring up StampPlot, and click on the Connect button to verify the voltage on the graph, then click on Disconnect again.
- ✓ To test the lower limit, modify the circuit by moving the 1 kΩ resistor so it connects pin 6 (CH3) and pin 7 (Vss) of the ADC0834.
- ✓ Re-run the modified code, and then close the Debug Terminal.
- ✓ Bring up StampPlot, click on the Connect button to view the voltage graph again, and then click on Disconnect.

Once your channel testing is completed:

- ✓ Erase the 3 lines of testing code from the `Main` routine, and resave the program.
- ✓ Remove the 1 kΩ resistor from the circuit, since we won't need it any longer in its current position.

I'm sure by now you can see that the StampPlot graphical plotting software adds considerably to your understanding of how voltages can be viewed in real-time. It's almost like having an oscilloscope to monitor the voltage readings. The difference, however, is that the StampPlot software cannot react anywhere near as fast as a conventional scope display. Nevertheless, the StampPlot software is more than adequate for our upcoming experiments. So now that we have the StampPlot software functioning along with the A/D converter, let's proceed with Experiment 1.

But before you do, just remember ...

"Try not to have a good time. This is supposed to be educational." -- Charles Schulz.

Now forget it...you will have fun...we promise!

More Information

Find more information about energy, DC and AC at www.learnonline.com/ewre.html.

Chapter 2: Experiment #1, Programmable Battery Charger

Experiment #1 begins this *Experiments with Renewable Energy* course with an interesting and practical project; that is, building a programmable battery charger.

Battery chargers have been around ever since rechargeable batteries were invented. So what's so interesting about charging batteries? Actually quite a bit, since battery charging is really a combination of half art and half science, and there seems to be no end to the variety and complexity of battery charging circuits and techniques. The battery charger circuit that you will build in this experiment is no exception. It is yet one more way to charge two standard 1.2-volt AAA-style nickel cadmium (NiCad) batteries. However, the unique feature about this particular battery charger design lies in the fact that the Board of Education circuitry and software constantly monitors the battery's charge and discharge profiles using all four channels (`ch0` through `ch3`) of the A/D converter. We'll get into how this happens in Activity #3 - Programming the Battery Charger.

In the real world, rechargeable batteries can be either a convenience or a necessity. For devices like portable radios and flashlights, you have the choice of either rechargeable or non-rechargeable batteries. For these and other similar devices it may be easier to use non-rechargeable batteries because they typically cost less and last longer as compared with their rechargeable equivalents. The choice is yours. However, you do need a rechargeable battery to start the family car day after day. This is a necessity, since running to the battery store every day or two would become very inconvenient and expensive. Your cell phone and battery-operated electric drill also demand rechargeable batteries for the same reasons. Modern buildings, especially hospitals, use rechargeable batteries for immediate backup lighting if the normal AC power fails (because most doctors have trouble operating in the dark...). And if you're wise you will have a UPS, or Uninterruptible Power Supply, to back up your desktop computer in case of a blackout or brownout. Even a temporary loss of 110 VAC power (about 50 milliseconds, or less) can bring your computer to its knees. A UPS is a good idea to keep your computer up and running, since it has a rechargeable battery that is continuously kept at full charge in case this happens.

HOW NICAD RECHARGEABLE BATTERIES WORK

To begin to understand how NiCad rechargeable batteries work, one might appropriately ask the question, "How does an electrochemical cell work?"

A cell is a single electrochemical device with a single anode and a single cathode, while a battery is a collection of cells usually connected in series to obtain a higher terminal voltage. Batteries, whether they are primary (use once) or secondary (rechargeable) are devices that convert chemical energy into electrical energy. In the latter case they can also take in electrical energy and store it as chemical energy for later use. The key to understanding electrochemistry are the processes of oxidation and reduction, sometimes commonly referred to by the memory-aid phrase "LEO (the lion) goes GER (grr)"

Some Rechargeable Battery Vocabulary

LEO = Lose Electrons Oxidation

GER = Gain Electrons Reduction

Anode: the electrode where oxidation takes place in an electrochemical cell.

Cathode: the electrode where reduction takes place in an electrochemical cell.



Electrode: a conductor that forms an electrical contact between metallic and non-metallic components in a circuit.

Electrolyte: a liquid solution that can conduct ions.

Ion: a charged atom or radical of a molecule that is capable of transferring electrical charge.

Oxidation: the loss of electrons from an atom, ion, or molecule.

Reduction: the gain of electrons to an atom, ion, or molecule.

When a material oxidizes it gives up electrons and becomes more positively charged, or enters a higher oxidation state. Likewise, a material is reduced when electrons are added to it, either making it negatively charged or reducing its oxidation state. For example, assume we start with two materials A and B. Now let's further assume that material A is easily oxidized (it likes to lose electrons) and material B is easily reduced. Now immerse materials A and B into an electrolyte solution, which can conduct ions. By doing this, a circuit is completed from A to B where A is oxidized and electrons are released to flow through the circuit to B where it is reduced. This, of course, is an oversimplified view because only certain combinations of materials (A and B) and electrolytes provide useful and practical batteries.

The cathode of a cell is where reduction takes place (gains electrons), and the anode is where oxidation takes place (loses electrons). So in a battery, which is producing current, the positive terminal is the cathode and the negative terminal is the anode. This is counterintuitive based on an understanding of diodes, where the cathode is negative with respect to the anode, but that's how a cell's anode and cathode function, nevertheless. When a NiCad cell is fully charged, the cathode is composed of Nickelic Hydroxide. Nickel is one of those elements that have multiple oxidation states; in other words, it can lose a different number of electrons per atom, depending on how hard it is coerced. Nickel is usually found with oxidation states of 0 (free metal), +2, +3 and +4. The +2 state is referred with an “-ous” suffix, while the +3 and +4 states are referred with a “-ic” suffix. So, nickelic hydroxide is really NiOOH (the nickel has a charge of +3) or Ni(OH) (the nickel has a charge of +4).

The anode is composed of free cadmium metal (zero oxidation). The electrolyte is usually a solution of potassium hydroxide (KOH). As explained earlier, when a load is connected to the cell the anode becomes oxidized and the cathode is reduced. Electrons leave the anode where the cadmium is oxidized and forms: $\text{Cd}(\text{OH})_2$, plus 2 free electrons. These two electrons go to the cathode where they reduce the nickelIC hydroxide to form nickelOUS hydroxide or Ni(OH) (where the nickel has a charge of +2). This reaction can take place until the materials are exhausted. In theory, cells are manufactured so that both anode and cathode are spent at roughly equal rates.

Charging versus Discharging

The previous discussion was about what happens when a NiCad cell or battery is discharging. Well, what happens when a cell or battery charges? In effect, charging is the reverse of discharging. During the charging cycle current is being forced back into the cell (opposite of discharge current). Here, electrons are being taken out of the positive terminal and forced into the negative terminal. This means that the material at the positive terminal is being oxidized (hence it is now the anode -- confusing, eh?) and the material at the negative terminal is being reduced (now the cathode). In the NiCad system the cadmium hydroxide is being re-converted into cadmium, and the nickelous hydroxide is being re-converted to nickelic hydroxide.

The easy part of charging is reconverting the spent material on the plates to the charged condition. The hard part comes in knowing when to stop. Let us take a moment to think about what happens when we overcharge the battery. Once all the nickelous hydroxide is converted into nickelic hydroxide, and in theory all the cadmium hydroxide is converted into cadmium, the charging current has to go somewhere. As the energy of the charging

current cannot go into more chemical energy, it goes into splitting water molecules (water is still the major constituent of the electrolyte). Just like the age old chemistry experiment of splitting water into hydrogen and oxygen, a fully charged NiCad cell does the same thing. You are forcing oxidation at the positive terminal and reduction at the negative. When one oxidizes water (actually the OH- ion), one produces oxygen. Likewise, at the negative terminal (now the cathode), one produces hydrogen.

This of course is bad because oxygen + hydrogen = BOOM!!!

Cell manufacturers attempt to prevent this from happening. During manufacture, they deliberately oversize the negative plate, and then partially discharge it. That is, they put a fully charged positive plate, but put a slightly discharged, but bigger plate of cadmium in. The amount of free cadmium in the oversized plate is matched to discharge in step with the amount of nickel hydroxide provided in the positive plate.

Now consider what happens as full charge is achieved. Oxidation of water starts at the anode, but since the cathode is oversized, and has excess hydroxide, the current continues to produce cadmium metal instead of hydrogen. At the same time, the separator (the material used to prevent the plates from shorting) is designed to allow oxygen gas to diffuse through, from the positive to the negative plate. The free oxygen then oxidizes the cadmium metal to form more cadmium hydroxide to prevent hydrogen from being formed. The outcome is a safe battery.

The Danger of Overcharging NiCad Batteries

The charging technique just described will work only as long as the overcharging current is limited to a value such that the rate of oxygen liberation at the anode is less than or equal to the rate of diffusion across the separator. If the overcharging current is too high, excess oxygen is produced at the anode. And since not enough oxygen can diffuse across to make up for the reduction at the cathode, the excess cadmium hydroxide is used up. Hydrogen is then formed. This leads to a dangerous situation, due to both fire and overpressure. Cells are designed to vent when this condition occurs, releasing the excess hydrogen and oxygen to the air before bad things happen. While this may keep cells from blowing up, it does damage them, since the cell is losing material. Further, it upsets the chemical balance inside the cell; in other words, if the cells lose enough water, they stop working. Another problem is the heat that is produced by the process of generating oxygen and recombining it at the cathode. With a moderate amount of current, the cell temperature can rise considerably to 50 or 60 degrees Celsius (122 to 140 degrees Fahrenheit). If, after charging, the batteries are hot, then you have overcharged them.

Just remember, charge control is the key to proper battery management. More batteries are destroyed or damaged by bad charging techniques than all other causes combined. Once a battery reaches full charge, the charging current has to go somewhere - most often generating heat and gases. Both are bad for batteries.

Heat Is The Enemy of Batteries

A NiCad battery stored, used, or charged under high temperature conditions will die an early death. Heat causes the separator to weaken, the seals to weaken, and greatly accelerates changes in the plate material, some of which cause the dreaded memory effect (explained below). Even though the cells may not vent, the heat by-product is wearing down the cells. Specifically, hydrolysis or degradation of the separator material, usually polyamide, is greatly accelerated at high temperatures. This leads to premature cell failure.

Rechargeable Battery Care

DON'T deliberately discharge the batteries to zero volts.

DON'T leave the cells on trickle charge for long periods of time.

DON'T overcharge the cells.

DO let the cells discharge to 1.0V/cell on occasion through normal use.

DO protect the cells from high temperature both in charging and storage.

The Dreaded Memory Effect

What is the dreaded memory effect? Simply put, it's a condition that is particular to NiCad batteries where the battery can never be recharged to its maximum usable voltage again. Here is what usually causes the dreaded memory effect:

When a NiCad battery has been fully charged and then begins to discharge due to normal use, a strange thing happens (chemically, inside the battery) if the battery is repeatedly recharged *before* it is fully *discharged*. This strange thing is called the "dreaded memory effect", since the battery will "remember" the voltage it was at when the recharge begins to take place and will never go much higher than that voltage ever again.

For example, if you took a normal 9 V NiCad battery, fully charged it to 9 volts, then put it in a circuit, like the Board of Education, and allowed it to discharge to, say 6 volts, then

took it out and began to recharge it again, and did this repeatedly, the battery would eventually never charge up over 6 volts again. Other practical examples of the NiCad battery memory effect occur in household appliances like a mini hand vacuum or portable hand drill. If these appliances are repeatedly recharged without draining their batteries completely, they may never work as they did when they were new. How many "dead" battery-operated appliances do you have laying around the house that will never see another moment of useful application? Their useless working condition is probably due to the memory effect we're discussing here. In other words, their NiCad batteries were recharged too often and before they were fully discharged.

In order to prevent the memory effect, a NiCad battery needs to be both fully charged *and* then nearly fully discharged to get the maximum life out of it. Otherwise, the battery will revert to a lesser maximum voltage on subsequent charges. There is much controversy regarding the memory effects of NiCad batteries; some experts say it exists while others say it doesn't, so it all depends on what battery expert you speak with or read about. Nevertheless, your author has experienced this memory effect for years, now, so it does appear to be real. That said, with newer battery technologies coming on the market coupled with newer designs of NiCad batteries themselves, this once daunting "dreaded memory effect" will go away and hopefully be "forgotten forever".

Credit

Credit for these materials and insights go to Ken A. Nishimura whose NiCad battery ramblings provided much of the content for this section.

ACTIVITY #1: CHECKING THE BATTERY VOLTAGE

In this chapter we will be charging two (2) AAA 1.2 volt NiCad batteries. In this activity we'll determine the state of our rechargeable batteries. We'll use our A/D converter circuit to measure the voltage of the set of batteries, and from that information, we'll know whether the batteries are charged or discharged.

Parts Required

- (1) A/D Converter circuit from Chapter 1, from Figure 1-6 and Figure 1-7
- (1) 1 k Ω resistor (brown black red)
- (1) AAA Battery Holder (red wire is +, black wire is -)
- (2) 1.2-volt AAA NiCad rechargeable batteries (included)

- ✓ Disconnect power to your BASIC Stamp.
- ✓ Insert the AAA batteries into the battery holder.
- ✓ Add the $1\text{ k}\Omega$ resistor and battery holder to the circuit shown in Figure 2-1 and Figure 2-2.

This is the same basic circuit we used in the last chapter; here, we'll use it to read the voltage produced by the AAA batteries. The battery voltage is connected to ch0 of the A/D converter via a $1\text{ k}\Omega$ resistor.

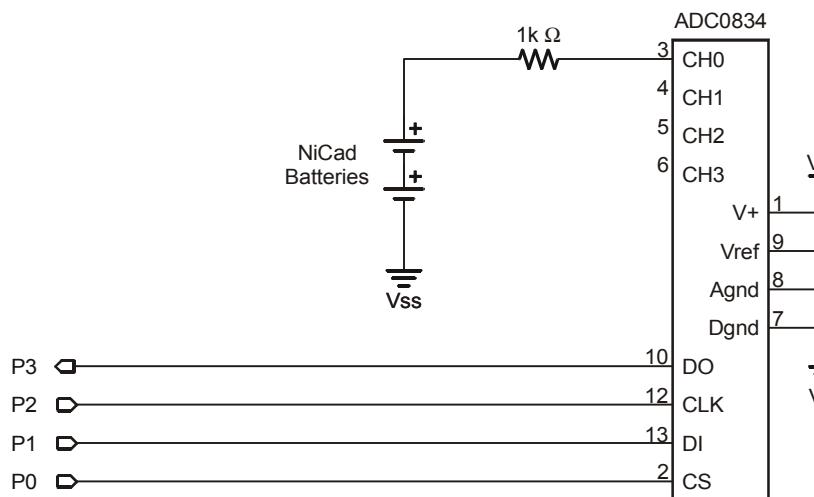


Figure 2-1
Voltage
Tester
Schematic

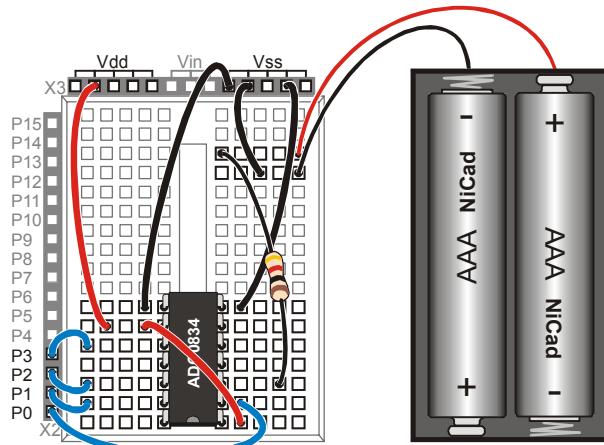


Figure 2-2
Voltage Tester Wiring Diagram

- ✓ Run the program ADCtest.bs2.
- ✓ Note the COM port being used, then close the Debug Terminal.
- ✓ Bring up the StampPlot screen using the sic_ewre_basic macro, and set the COM port to the one you just noted.
- ✓ Click on the Connect button, making sure that the Enable Plotting box to the right is also selected.

You should see a line plotted across the screen, representing the voltage of the AAA batteries.

- ✓ Read the value from the graph, and write it in the space below:

AAA Batteries Initial Voltage: _____

In the next experiment, you will build a programmable battery charger. If your batteries measure less than 2.0 V, then they will take a charge. However, if the batteries measure 2.0 V or greater, they don't need charging. In this case, they'll be discharged before they can be charged.

ACTIVITY #2: BUILDING THE PROGRAMMABLE BATTERY CHARGER

Now it's time to build the NiCad battery charging circuit! Start by gathering the parts listed below.

Parts Required

- (2) 2N3904 transistors
- (1) Red LED
- (1) Yellow LED
- (1) Green LED
- (2) 10 Ω resistors
- (2) 220 Ω resistors
- (3) 470 Ω resistors
- (1) 1N4148 diode
- (1) AAA Battery Holder (red wire is +, black wire is -)
- (2) 1.2-volt AAA NiCad rechargeable batteries (included)
- (1) DC fan assembly (from the Solar Kit)
 - 1 DC motor
 - 1 stand base
 - 1 stand post
 - 1 stand loop
 - 1 screw
 - 1 hex nut
 - 1 2-wire lead with connectors
- Electrical tape (not included)

- ✓ Assemble the DC fan motor, propeller, and stand, following the direction in the Solar Kit.
- ✓ Connect the wire leads to the back of the fan motor with the attached slotted clips.
- ✓ Bend one bare end of each jumper wire into a tight hook.
- ✓ Slip one hook through each eyelet on the other end of the fan motor leads.
- ✓ Pinch the hooks closed to make sure the bare metal of both parts stays in contact.
- ✓ Cover the hook and eye wire joint with a small fold of electrical tape.
- ✓ Make sure power is disconnected from your board.
- ✓ Build the circuit shown in the schematic (Figure 2-3) and wiring diagram (Figure 2-4) being very careful to place the parts in exactly the positions shown.
- ✓ Be sure that both transistors are positioned with their flat sides facing towards the edge of the board.

- ✓ Be careful that the leads of the LEDs do not touch each other, or the leads to the resistors. If, after you have programmed your battery charger, you see some strange LED blinking behavior that is not described in the text, check your LED leads to make sure they are not forming unwanted connections!

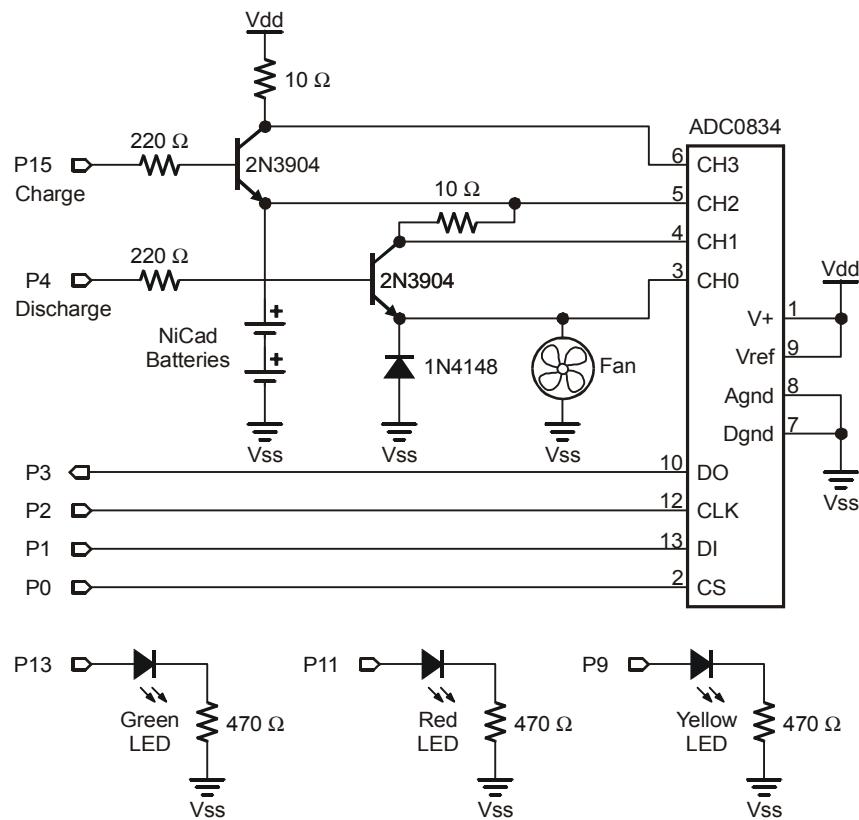


Figure 2-3: Programmable Battery Charger Schematic

NOTE: The ADC channels are not depicted in the same order in this schematic as they are in other schematics throughout the text. Remember that the schematic depicts only connections and not actual parts placement.

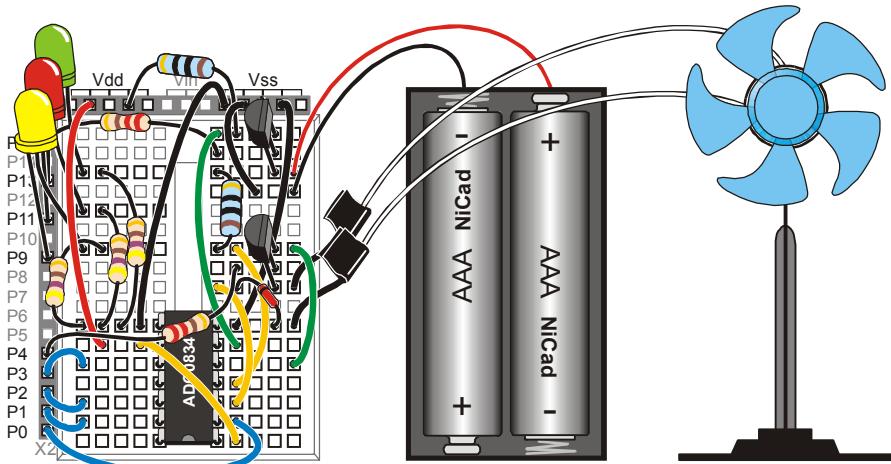


Figure 2-4

Programmable Battery Charger Wiring Diagram

- ✓ Connect the DC fan motor to the battery charger circuit as shown in the schematic (Figure 2-3) and wiring diagram (Figure 2-4).
 - ✓ If you are continuing on now, check your wiring carefully then reconnect power to your board.



WARNING: Do not leave your Programmable Battery Charger Unattended! Commercial programmable battery chargers have many built-in sensors and safety features that are beyond the scope of this text. Please follow a few precautionary guidelines:

DO double-check your wiring before programming your battery charger

DON'T leave your battery charger circuit unattended while the board's power supply or the batteries are connected.

DON'T leave your batteries in the battery holder if the holder is not connected to the breadboard. The unsecured leads could touch each other and cause a short circuit.

If your plot varies greatly from what is expected, disconnect your batteries and board power supply, and reexamine your circuit and code before proceeding.

If your batteries feel hot to the touch while charging, disconnect the power supply from your board. However, some slight warmth is normal.

How the Battery Charger Circuit Works

The battery charging circuit consists of two 2N3904 NPN transistors, the NiCad rechargeable batteries themselves, the A/D converter, a small-signal diode and four

resistors. In addition, three LED's are used as indicators, one for each program state: Charging (Green), Discharging (Red), and Replay of the charge cycle (Yellow).

Now take a closer look at the battery charger schematic in Figure 2-3.

Simply put, the Charge transistor on P15 is used to charge the batteries, and the Discharge transistor on P4 is used to discharge them. For their individual circuits, both transistors use a 220Ω resistor connected to the transistor's base and a $10\ \Omega$ resistor connected to the collector. The other ends of the $220\ \Omega$ resistors are connected to the Board of Education via two pins, P15 and P4, which control the charge and discharge cycles, respectively. The two $10\ \Omega$ resistors act to limit the current for the battery charge and discharge cycles.

If you're not familiar with transistor operation, the following is a brief, but adequate, explanation for their use in the battery charger circuit.

Common transistors, like the 2N3904 used in this experiment, are designed to amplify a small input signal (on the base) to a correspondingly larger output signal (on the collector). For our purposes, however, we are using both transistors as low-current, solid-state switches and not as amplifiers.

With either P15 or P4 at a logic low voltage, at or near zero volts, the respective transistors are said to be in a "cutoff" state. That is, no current can flow from the collector to the emitter. Therefore, if we wish to disable either the charge or discharge features, we need only set P15 or P4 to a logic 0 with the `LOW` instruction. In effect, this turns our solid-state transistor switch OFF.

However, in order to enable the charge or discharge functions, P15 or P4 needs to be at a logic 1 state, or around 3.5 volts. Programming either pin with the `HIGH` instruction does this. When this happens the voltage at the base is high enough to put the transistor into a "saturation" state. With the transistor in saturation, the maximum amount of current can flow from the collector to the emitter, with the current limited only by the voltage drop across the $10\ \Omega$ resistor plus the small internal resistance of the transistor itself. Putting the transistor into saturation turns our solid-state transistor switch ON.

Once again, the above should not be considered a rigorous explanation of transistor operation. It does, however, point out the manner in which the transistors are used in our battery charging circuit. Therefore, if you're so inclined, and we hope you are, please

refer to the web for more material on transistor theory of operation. Transistors are fundamental to nearly all modern electronics, so learning more about them will be a rewarding investment of your time. Plus, you will be able to better understand how other circuits work, even those that use integrated circuits, which are all built around transistors.

Now back to how the battery charger circuit works...

The battery charger program code makes sure that only one of the two transistors is ON at any given time. What this means is that the batteries are either being charged or discharged, but never at the same time.

During the charge cycle, the Charge transistor is turned ON and the Discharge transistor is turned OFF. Current is allowed to flow into the batteries from Vdd, the +5 volt regulated supply, through the $10\ \Omega$ resistor, and the small internal resistance of transistor itself, thus limiting the amount of current flow to the sum of these two resistances.

During the discharge cycle, the Charge transistor is turned OFF and the Discharge transistor is turned ON. Now current is allowed to flow into the load, the motor-fan, which is again limited by a second $10\ \Omega$ resistor and the small internal resistance of the transistor.

That takes care of the actual charging and discharging functions. Now to the rest of the circuit...

The three LEDs show which state the battery charger circuit is in at the time:

- Flashing green for the charging state
- Steady green when batteries have finished charging
- Flashing red for the discharge state
- Steady red when batteries have finished discharging
- Flashing yellow for the replay state where the charge cycle's voltage and current are displayed in a fast-forward manner

The diode in this circuit is functioning as a “flyback diode”, and its purpose is to protect the transistor from a wash of back-EMF when the fan motor stops or reverses direction during our discharge cycle.

Finally, the A/D converter is used to measure four important parameters in the charge and discharge cycles:

- BCI - Battery charge current (A/D CH3)
- BV – Battery voltage (A/D CH2)
- BLI – Battery load current (A/D CH1)
- BLV- Battery load voltage (A/D CH0)

The A/D converter constantly monitors these parameters and provides feedback to the BASIC Stamp module about the charge and discharge cycles. We will be using StampPlot Pro to view plotting examples of the charge-discharge cycles after you program this experiment, next.

ACTIVITY #3: PROGRAMMING THE BATTERY CHARGER

With the circuit built it's now time to program the BASIC Stamp module for this application. As before, let's start with updating the program name and declaring the variables.



You may enter the program by typing in the code shown below, or you may download it from the Experiments with Renewable Energy product page at www.parallax.com.

- ✓ In the BASIC Stamp Editor, open ADCTest.bs2
- ✓ Select File, then Save as; rename the program BatteryCharger.bs2
- ✓ Update the title to read as follows:

```
' Experiments with Renewable Energy v1.0 - BatteryCharger.bs2
' Programmable Battery Charger which charges, drains, and keeps a record
' of the voltages and currents, to be replayed via StampPlot.
```

Add the following code to the Declarations section, below the other declared variables for the `Plot_It` and A/D Converter subroutines:

```
' ----- For Experiment 1: Programmable Battery Charger -----
ChargeLed  PIN      13          ' Charge LED (green)
DrainLed   PIN      11          ' Drain LED (red)
ReplayLed  PIN      9           ' Replay LED (yellow)

ChargeBatt PIN      15          ' to charge transistor base
DrainBatt  PIN      4           ' to drain transistor base
```

```

codePtr      VAR      Nib           ' pointer to code block to execute
codePtr = 0
dataPtr      VAR      Word          ' starting at 0 (Exp_1_Init)
DataPtrMax   CON      400           ' EEPROM pointer for data logging
BattFullChg CON      150           ' maximum EEPROM pointer value
True         CON      1             ' ~3.00 V for both batteries in series
                                    ' Boolean true/false

counter     VAR      Byte          ' counter for averaging every 64 sec.
avgVolts   VAR      Word          ' average battery voltage storage
avgCurrent  VAR      Word          ' average battery current storage
MinVolts    CON      100           ' ~2.00 V for both batteries in series
maxVolts    VAR      Byte          ' maximum measured battery voltage
doneDraining VAR      Bit           ' tell whether battery drained
doneCharging VAR      Bit          ' tell whether battery charged

```

Now it is time to follow up with adding the code for Experiment 1. First we'll add an explanation that documents the purpose of the subroutine. Then, we'll add the executable commands between the `Exp_1:` and `Exp_1_End:` labels, filling in this "floor" of our program "building."

- ✓ Enter the following commented code at the beginning of the Experiment 1 section to document the subroutine that will come next, as shown below:

```

'-----'
' Subroutines
'-----'

'-----'
' Experiment 1: Programmable Battery Charger
'-----'

'Based on codePtr, branch to the appropriate subroutine
'  Exp_1_Init          codePtr = 0
'  Exp_1_Charge        codePtr = 1
'  Exp_1_Drain         codePtr = 2
'  Exp_1_Replay        codePtr = 3

' Exp_1_Init
'   Halt charging and discharging
'   Extinguish all LEDs
'   Zero relevant variables
'   Read last codePtr value from EEPROM,
'     increment it
'     test for overflow
'   Write codePtr back to EEPROM
'   Exit

' Exp_1_Charge
'   Check to see if battery is done charging.  If not,

```

```

' Enable charging
' Disable discharging
' Flash the charging LED
' Average the battery voltage & current for 256 PlotIt cycles (64 seconds)
' Abort charge cycle if the battery voltage is too high at the beginning
' Else, store the average of the sampled voltage and current to EEPROM
' Increment the EEPROM pointer
' Test to see if the timed charge cycle is complete (dataPtr=DataPtrMax)
' If so, set doneCharging = True to begin draining the battery next loop
' Exit

' Exp_1_Drain
' Check to see if battery is drained already.  If not,
' Disable charging
' Enable discharging
' Flash the discharging LED
' Sample the discharge voltage and current
' If battery voltage is below minimum set doneDraining = True,
'     turn the Drain LED on, and deactivate any more draining.

Exp_1_Replay
' Disable charging and discharging
' Flash the replay LED
' For each data pair in EEPROM sampled during the charge cycle
' Display the charge voltage and current using StampPlot, using
' the PlotIt routine's 250 ms delay as a timer for each sample pair
' Exit
'
```

- ✓ Next, fill out the subroutine by adding the code between the between the **Exp_1:** and **Exp_1_End:** labels, as shown below.

```

Exp_1:
    BRANCH codePtr, [Exp_1_Init,Exp_1_Charge,Exp_1_Drain,Exp_1_Replay]

Exp_1_Init:
    LOW ChargeBatt           ' halt battery charging
    LOW DrainBatt            ' and discharging
    LOW ChargeLed            ' and extinguish all LEDs
    LOW DrainLed
    LOW ReplayLed

    avgVolts = 0              ' zero the average battery charging voltage
    avgCurrent = 0             ' zero the average battery charging current
    maxVolts = 0               ' zero the maximum measured voltage
    counter = 0                ' zero the 64 second loop counter
    dataPtr = 0                 ' zero the EEPROM data logging pointer

    READ DataPtrMax+1,codePtr   ' select the routine to execute

```

```

codePtr=codePtr+1           ' by incrementing the codePtr
IF codePtr>3 THEN codePtr=0   ' test for overflow then...
WRITE DataPtrMax+1,codePtr    ' write it's value back to EEPROM
GOTO Exp_1_End                 ' and exit...the selected subroutine will
                                ' run based on the BRANCH instruction above
                                ' the next time thru.....

Exp_1_Charge:                  ' codePtr = 1 : charge batteries
                                ' Jump out if battery is charged already
IF (doneCharging = True) THEN GOTO Exp_1_End

HIGH ChargeBatt                ' activate battery charging transistor
LOW DrainBatt                  ' deactivate battery discharging transistor
TOGGLE ChargeLed               ' flash the charge LED
LOW DrainLED                   ' extinguish the others
LOW ReplyLED

a2dMuxId = a2dMuxId3          ' sample the battery charge current (BCI) and
GOSUB A2D                         ' add it to avgCurrent
ch3 = (255-a2dResult)           ' I = (Vdd - Vcollector)/10 ohms
avgCurrent = avgCurrent+ch3

a2dMuxId = a2dMuxId2          ' sample the battery voltage (BV) and
GOSUB A2D                         ' add it to aveBattVolt
ch2 = a2dResult
avgVolts = avgVolts+ch2

                                ' abort charge cycle - battery is charged
IF (ch2 > BattFullChg) AND (dataPtr <10 ) THEN
  doneCharging = True           ' set flag
  HIGH ChargeLed               ' turn charge LED on steadily
  LOW ChargeBatt                ' deactivate battery charging transistor
ENDIF
counter = counter - 1          ' decrement the averaging counter
IF counter <> 0 THEN Exp_1_End      ' exit if not zero, else

WRITE dataPtr,avgVolts.HIGHBYTE   ' store avg. charging voltage to EEPROM
dataPtr = dataPtr+1                  ' increment the EEPROM pointer

WRITE dataPtr,avgCurrent.HIGHBYTE  ' store avg. charging current to EEPROM
dataPtr = dataPtr+1                  ' increment the EEPROM pointer

' IF avgVolts.HIGHBYTE > maxVolts THEN maxVolts = avgVolts.HIGHBYTE
' IF avgVolts.HIGHBYTE <= maxVolts - 10 THEN codePtr=2:GOTO Exp_1_End

avgVolts = 0                      ' reset for the next set of data
avgCurrent = 0

IF (dataPtr = DataPtrMax) THEN      ' charge cycle complete
  doneCharging = True           ' set flag so we don't charge any more
  HIGH ChargeLed               ' turn charge LED on steadily

```

```

        LOW ChargeBatt           ' deactivate battery charging transistor
        ENDIF

        GOTO Exp_1_End          ' beginning with the next time thru

Exp_1_Drain:
        ' codePtr = 2 : discharge batteries
        ' Jump out if battery is drained already
        IF (doneDraining = True) THEN GOTO Exp_1_End

        LOW   ChargeBatt         ' deactivate battery charging transistor
        HIGH  DrainBatt          ' activate battery discharging transistor
        TOGGLE DrainLED          ' flash the DrainLed
        LOW   ChargeLed          ' extinguish the others
        LOW   ReplayLed

        a2dMuxId = a2dMuxId0     ' sample the load voltage (BLV)
        GOSUB A2D
        ch0 = a2dResult

        a2dMuxId = a2dMuxId2     ' sample the battery voltage (BV)
        GOSUB A2D
        ch2 = a2dResult

        a2dMuxId = a2dMuxId1     ' sample the load current (BLI)
        GOSUB A2D
        ch1 = ch2 - a2dResult
        ' ch1 = I = (Vbatt-Vcollector)/10 ohms
        ' test battery voltage for below min

        IF (ch2 < MinVolts) THEN
            doneDraining = True
            HIGH DrainLed
            LOW DrainBatt
            ' deactivate battery discharging transistor
        ENDIF

        GOTO Exp_1_End

Exp_1_Replay:
        ' codePtr = 3 : replay charge cycle

        LOW   ChargeBatt         ' deactivate battery charging transistor
        LOW   DrainBatt          ' deactivate battery discharging transistor
        TOGGLE ReplayLed          ' flash the replayLed
        LOW   ChargeLed          ' extinguish the others
        LOW   DrainLed

        READ dataPtr,avgVolts.LOWBYTE
        ch2 = avgVolts.LOWBYTE      ' display the voltage and ...
        dataPtr = dataPtr+1

        READ dataPtr,avgCurrent.LOWBYTE
        ch3 = avgCurrent.LOWBYTE    ' current from the last charge
        dataPtr = dataPtr+1

```

```

IF (dataPtr => DataPtrMax) THEN
    datePtr = 0
    HIGH ReplayLed           ' turn replay LED on solid
ENDIF

Exp_1_End:
RETURN

```

- ✓ Whew!!! Save your work!! If you typed this all in by hand rather than downloading the program, you may want to click on the checkmark icon in the BASIC Stamp Editor. This will test your program to see if it tokenizes. If it does not, follow the error message hints to check your code.

How BatteryCharger.bs2 Works

First, the Declarations section additions define useful constants and pin numbers used in the battery charger.

ChargeLed	PIN	13
DrainLed	PIN	11
ReplayLed	PIN	9
ChargeBatt	PIN	15
DrainBatt	PIN	4

The names in front of the **PIN** declarations give meaning to the individual pins. The names make following the code understandable and meaningful, since instead of saying **HIGH 15** to enable the battery charging circuit, it is more obvious to say **HIGH ChargeBatt**. Similarly, to flash the individual LEDs, saying **TOGGLE ChargeLED** is more “illuminating” as compared with saying **TOGGLE 13**. This is why programmers came up with symbolic names in the first place - to make the program code more readable. Also notice that the names for pin declarations begin with a capital letter. This is according to the Parallax “Elements of PBASIC Style”, which can be viewed at anytime by clicking on the Help icon on the top line of the BASIC Stamp Editor.

Now let's look at the declared variables and constants.

```
codePtr      VAR  Nib
codePtr = 0
dataPtr      VAR  Word
DataPtrMax   CON  400
BattFullChg CON  150
True         CON  1

counter      VAR  Byte
avgVolts    VAR  Word
avgCurrent   VAR  Word
MinVolts    CON  100
maxVolts    VAR  Byte

doneDraining VAR  Bit
doneCharging VAR  Bit
```

Starting with the variable `codePtr`, first notice that it and all the other variables begin with a lower-case letter, and constants with an upper case letter. Once again, this is according the Parallax “Elements of PBASIC Style”. It’s mentioned here to acquaint you with how variable names and program labels are portrayed in PBASIC, so that you can begin to write your own code with the same consistency and style. Sloppy code is just that...sloppy! Do it right the first time, and you’ll be rewarded with knowing that you’re programs not only work well, but also look professional.

Also notice that just under the `codePtr` declaration (a 4-bit nibble) we initialize it to zero. We can do this here or in the program code, however making `codePtr=0` here gives special meaning to the operation of the program, so please keep this in mind. We’ll discuss why we did it here very quickly.

The next two variables deal with a pointer to EEPROM (`dataPtr`) and its maximum allowed value (`DataPtrMax`), which the program code uses to store voltage and current data in EEPROM during the charge cycle.

The `BattFullChg` variable is set to a constant, or `con`, of 150 that is equivalent to 3.00 volts. Recall that each level of the A/D converter output represents approximately 0.02 volts, so $150 \times 0.02 = 3.00$ volts.

The constant `True` is set equal to the value of 1. This allows the programmer to check whether something is true just by using the symbolic constant.

Next comes the **counter** variable, which is set to an 8-bit byte value and is used by the program to help in timing the charge cycle. Remember, an 8-bit byte can have up to 256 states and the program uses all of them.

The **avgVolts** and **avgCurrent**, both 16-bit Word variables, store the sum of 256 battery voltage and current values, respectively. You'll see a neat trick (make that "sophisticated programming example" ;) as to how to compute the average value of these 256 summed readings when we get into the code.

Next come the **MinVolts** constant, 100, that corresponds to 2.00 volts by the same reasoning as the **BattFullChg** variable, and the **maxVolts**, a byte that will keep track of the maximum voltage experienced during the charge cycle.

Rounding out the list are bit-sized variables named **doneDraining** and **doneCharging**. These two items act as "flags" to keep track of which state the program is in. For instance, when the batteries are charged, the **doneCharging** variable will be set to **True** (1). Each time the **Exp_1** subroutine is called, the program knows which state it was in – if **doneCharging** is **True**, the program simply turns on the Red LED steadily and exits.

Now with some degree of understanding of the variables and constants involved, let's look at the first instruction of the **Exp_1** subroutine to see how the battery charger program code begins to do its thing.

```
Exp_1:
    BRANCH codePtr, [Exp_1_Init, Exp_1_Charge, Exp_1_Drain, Exp_1_Replay]
```

When the Main routine calls this subroutine with **GOSUB Exp_1**, the first instruction it encounters is the **BRANCH** where the **codePtr** variable determines which of four branches, or jumps, that it will take; i.e., **Exp_1_Init** (**codePtr=0**), **Exp_1_Charge** (**codePtr=1**), **Exp_1_Drain** (**codePtr=2**) or **Exp_1_Replay** (**codePtr=3**). Remember when we declared the variables for this experiment a few paragraphs before, we set **codePtr=0** immediately after declaring it. What this does for us is to make sure that we branch to **Exp_1_Init** after first loading and running the entire program...and ... after every reset...something that is quite important to how we cycle through the rest of this subroutine.

Next, the `Exp_1_Init` subroutine deactivates both the charge and discharge transistors and, also, extinguishes all three LEDs and clears some important variables by the following set of code:

```
Exp_1_Init:  
    LOW    ChargeBatt  
    LOW    DrainBatt  
    LOW    ChargeLed  
    LOW    DrainLed  
    LOW    ReplayLed  
  
    avgVolts = 0  
    avgCurrent = 0  
    maxVolts = 0  
    counter = 0  
    dataPtr = 0
```

Now comes the interesting part. The next instructions not only increment the `codePtr` variable and test for overflow (`codePtr > 3`), they also read and write a byte of EEPROM memory.

```
READ DataPtrMax+1,codePtr  
codePtr=codePtr+1  
IF codePtr>3 THEN codePtr=0  
WRITE DataPtrMax+1,codePtr  
GOTO Exp_1_End
```

So why not store `codePtr` in RAM (Random Access Memory) after it's incremented? Why EEPROM?

The answer lies in the fact that EEPROM does not lose its data after power is removed ...or... after a reset. As a matter of fact, the program code we're discussing is also stored in EEPROM, and will remain there, intact, even when power is removed and reapplied and, also, after the reset button on the Board of Education is pushed. And this is the important part. That is, we can step the program code through the charge, discharge and replay cycles each time the reset button is depressed and released... simply by keeping the `codePtr` variable in EEPROM. You'll learn more about EEPROM in Experiments 4 and 5.

Now look at the above code again.

Each time the reset button is pushed and released, the entire program begins executing from the beginning where the variables and constants are declared and initialized. This includes the `codePtr` variable where it is set to zero. So when the `BRANCH` instruction is encountered, a branch will always be made to `Exp_1_Init` after every reset. It is here that the `codePtr` variable is changed based on its current value in EEPROM.

For example, let's see how `codePtr` is used to step the program through all four states. The `codePtr` in EEPROM can start at any value from 0 to 255, but lets assume it starts at 0. Here's what a series of four reset button depress and release actions will accomplish...

First Reset

- `codePtr` initialized to zero in declarations
- `BRANCH` instruction goes to `Exp_1_Init`
- `codePtr` is read from EEPROM
- `codePtr` is incremented from 0 to 1
- `codePtr` is tested for overflow (>3)
- `codePtr` is written back to EEPROM
- program exits (returns to `Main`)
- next time `GOSUB Exp_1` is executed ...
- `BRANCH` instruction goes to `Exp_1_Charge`
- program stays in `Exp_1_Charge` indefinitely until....

Second Reset

- `codePtr` initialized to zero in declarations
- `BRANCH` instruction goes to `Exp_1_Init`
- `codePtr` is read from EEPROM
- `codePtr` is incremented from 1 to 2
- `codePtr` is tested for overflow (>3)
- `codePtr` is written back to EEPROM
- program exits (returns to `Main`)
- next time `GOSUB Exp_1` is executed ...
- `BRANCH` instruction goes to `Exp_1_Drain`
- program stays in `Exp_1_Drain` indefinitely until...

Third Reset

- `codePtr` initialized to zero in declarations

- **BRANCH** instruction goes to `Exp_1_Init`
- `codePtr` is read from EEPROM
- `codePtr` is incremented from 2 to 3
- `codePtr` is tested for overflow (>3)
- `codePtr` is written back to EEPROM
- program exits (returns to `Main`)
- next time `GOSUB Exp_1` is executed ...
- **BRANCH** instruction goes to `Exp_1_Replay`
- program stays in `Exp_1_Replay` indefinitely until...

Fourth Reset

- `codePtr` initialized to zero in declarations
- **BRANCH** instruction goes to `Exp_1_Init`
- `codePtr` is read from EEPROM
- `codePtr` is incremented from 3 to 4
- `codePtr` is tested for overflow (>3)
- `codePtr` is reset to 0
- `codePtr` is written back to EEPROM
- program exits (returns to `Main`)
- next time `GOSUB Exp_1` is executed ...
- **BRANCH** instruction goes to `Exp_1_Init`
- `codePtr` is read from EEPROM
- `codePtr` is incremented from 0 to 1
- `codePtr` is tested for overflow (>3)
- `codePtr` is written back to EEPROM
- program exits (returns to `Main`)
- next time `GOSUB Exp_1` is executed ...
- **BRANCH** instruction goes to `Exp_1_Charge`
- program stays in `Exp_1_Charge` indefinitely until....

...and so forth. As you can (hopefully) see by now, additional reset actions will continue to step the program through all four states, in order, thus allowing you to step from one program state to another by the push of a button.

As we stated at the beginning of this section, the **BRANCH** instruction is one of the most powerful and versatile instructions in PBASIC. Besides eliminating a lot of **IF...THEN** statements to test the `codePtr` variable, the **BRANCH** instruction can be used for more

sophisticated coding techniques, such as multi-tasking, or quickly switching between program labels without any button pushing, something that we won't do here, but that you should keep in mind as your coding skills advance to more elaborate programs.

ACTIVITY #4: DISCHARGING THE BATTERIES

The first thing we'll do is discharge the batteries. If your batteries are brand new, from the kit, and have never been charged before, then they will probably discharge quite quickly.

The small DC motor with the propeller will act as a load for the batteries. As the fan runs and the batteries discharge, we'll plot the batteries' voltage using StampPlot. Furthermore, let's time how long it takes for the batteries to discharge.

- ✓ Connect power to your board, and look at your watch and note the time.
- ✓ Download the code into the BASIC Stamp module. The Debug Terminal should pop up with the RX LED flashing.
- ✓ Note the COM port being used, then close the Debug Terminal
- ✓ Open StampPlot Pro by clicking on Start → Programs → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_1.spm.
- ✓ Set the COM port in StampPlot to the same one that was being used by the Debug Terminal.
- ✓ Click on the Connect button, and make sure Enable Plotting is checked.
- ✓ Now take a look at the LEDs.

No LEDs illuminated:

- ✓ Go back and recheck your wiring and parts placement.

More than one LED on at the same time:

- ✓ Your LED leads are probably touching and making a short circuit. Recheck your wiring.

Red LED flashing:

- ✓ This is the correct state we want to be in.

Red LED Steady, Green LED Steady, Green LED Flashing, or Yellow LED Flashing:

- ✓ Switch to discharge cycle by pressing the reset button until Red LED starts to flash.

- ✓ The fan motor should now be spinning. Observe your StampPlot graph!
- ✓ Use your watch to time how long it takes the discharge to complete. The Red LED will come on steadily when finished.

 **What do I do if my fan is running backwards?** Simply swap the leads clipped to the back of the DC motor!

If your batteries start from the fully charged state, it could take over two hours for them to discharge. However, if your batteries are brand new, it could take only a few minutes – or even seconds! If your fan ran for more than a few seconds, at least a portion of your StampPlot graph would look similar to what is shown in Figure 2-5.

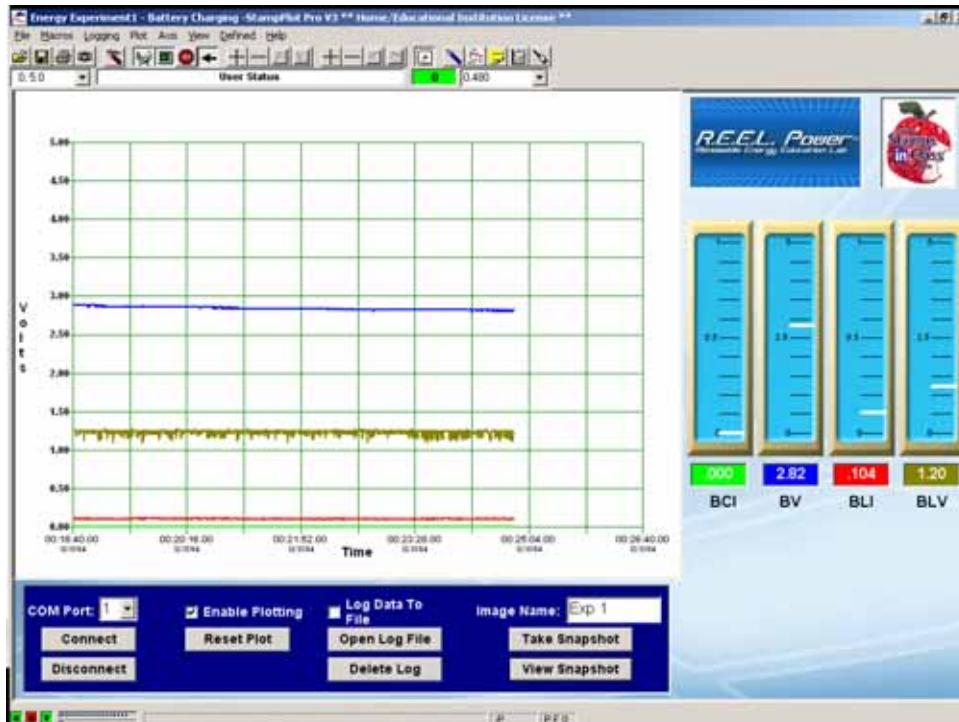


Figure 2-5: Discharging the Batteries with a Fan Motor Load

The main part of the display, including the grid plotting area and the rectangular buttons below it, you've seen before. However, the right side of the display is new. It consists of four vertical gauges that display the measure of voltage and current generated in the battery and load during the charge and discharge cycles. Then below each vertical gauge is a small rectangle that displays the numerical value of the vertical gauge above. And below that is a two or three letter indicator of the parameter itself. From left to right these stand for...

BCI	Battery Charge Current (in amps)
BV	Battery Voltage (in volts)
BLI	Battery Load Current (in amps)
BLV	Battery Load Voltage (in volts)

During the battery charging cycle, only the BCI and BV gauges are active, however during the discharge cycle the BV, BLI and BLV gauges are active. Along with the plotted lines, these gauges give you an exact real-time indication of the voltages and currents involved in each charge, discharge and replay cycle.

Also note that the colors of the rectangles just below the vertical gauges match the colors of the plot lines. This makes viewing the plot and gauges a snap to understand and comprehend.

We'll come back to the StampPlot screen soon with even more information about it, however while allowing the batteries to charge, it would be a good time to start to understand what's happening with the circuit and the code that makes it work.

- ✓ First examine the variables in the Declarations section. The comments after each variable indicate the variable's function.
- ✓ Next, look at the code and again examine the comments after each line of code, including the label names.

Can you figure out what is going on? Check your guesses by following the detailed explanation in the section below. It explains how both the hardware and software work together to make the programmable battery charger a neat experiment.

Now that the first part of the battery charger program (`Exp_1_Init`) has been explained, let's explain the `Exp_1_Drain` subroutine.

By installing the motor-fan load and entering into the battery discharge cycle, several things will become immediately apparent. First, note that the left-most vertical gauge (BCI or Battery Charge Current) is at zero. This is because we have switched to the discharge cycle so no charge current is being generated. Then notice that the right-most two vertical gauges (BLI or Battery Load Current and BLV or Battery Load Voltage) are now active, along with the BV the battery voltage, again.

Let's look at the `Exp_1_Drain` code that makes this happen.

```
Exp_1_Drain:

IF (doneDraining = True) THEN GOTO Exp_1_End

LOW ChargeBatt
HIGH DrainBatt
TOGGLE DrainLED
LOW ChargeLed
LOW ReplayLed

a2dMuxId = a2dMuxId0
GOSUB A2D
ch0 = a2dResult

a2dMuxId = a2dMuxId2
GOSUB A2D
ch2 = a2dResult

a2dMuxId = a2dMuxId1
GOSUB A2D
ch1 = ch2 - a2dResult

IF (ch2 < MinVolts) THEN
    doneDraining = True
    HIGH DrainLed
    LOW DrainBatt
ENDIF

GOTO Exp_1_End
```

The first line of code checks the `doneDraining` flag. If it's `True`, the program simply exits. Since all variables are initialized to zero, the flag will initially not be `True`. Otherwise, a `LOW ChargeBatt` deactivates the Charge transistor, so no charging current flows into the batteries. Instead, the `HIGH DrainBatt` activates the Discharge transistor,

enabling the batteries to drain or discharge. The `TOGGLE` instruction causes the red discharge LED to flash each time through this part of the subroutine. And finally, the last two instructions deactivate the other two LEDs.

The next six lines of code sample the battery load voltage (BLV) as `ch0`, followed by sampling the battery voltage (BV) as `ch2`. The following three lines of code sample the battery load current (BLI), which is really the voltage drop across the $10\ \Omega$ resistor that ties the emitter of the Charge transistor to the collector of the Discharge transistor. In this case the voltage drop across this resistor is the difference between `ch2` and the `a2dResult` value, which we assign to `ch1` for the `Plot_It` routine.

Finally, the battery discharge program ends with a test of the battery voltage (`ch2`) against `MinVolts`. If the batteries have been discharged below `MinVolts`, then they are drained. The `doneDraining` flag will be set true, the red `DrainLed` turned on, and the battery charging transistor, `DrainBatt`, turned off.

ACTIVITY #5: CHARGING THE BATTERIES

In this experiment, we'll be charging our two AAA NiCad batteries. The battery's fully charged "operating" voltage is 2.4 volts when the two batteries are connected in series. The key word here is "operating", since this is the voltage that the batteries should be at immediately after a full charge to provide power to a "load". In this case our load is a small DC motor-fan. The purpose of our battery charger circuit and PBASIC program is to charge the batteries until they are at full capacity, which will actually cause the battery voltage to go beyond the 2.4 volt level, but not overcharge them. For now, let's get to the state where the batteries are charging and the Green LED is flashing.

- ✓ If `BatteryCharger.bs2` isn't running, re-run the program.
- ✓ Note the COM port being used, then close the Debug Terminal.
- ✓ Open StampPlot Pro by clicking on `Start → Programs → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_1.spm`.
- ✓ Set the COM port in StampPlot to the same one that was being used by the Debug Terminal.
- ✓ Click on the Connect button, and make sure the Enable Plotting box is checked.
- ✓ Press the Reset button until the Green LED is flashing.

Your StampPlot plot will look something like the one below. As the batteries charge, you will observe the blue plot, BV, or battery voltage, rising.

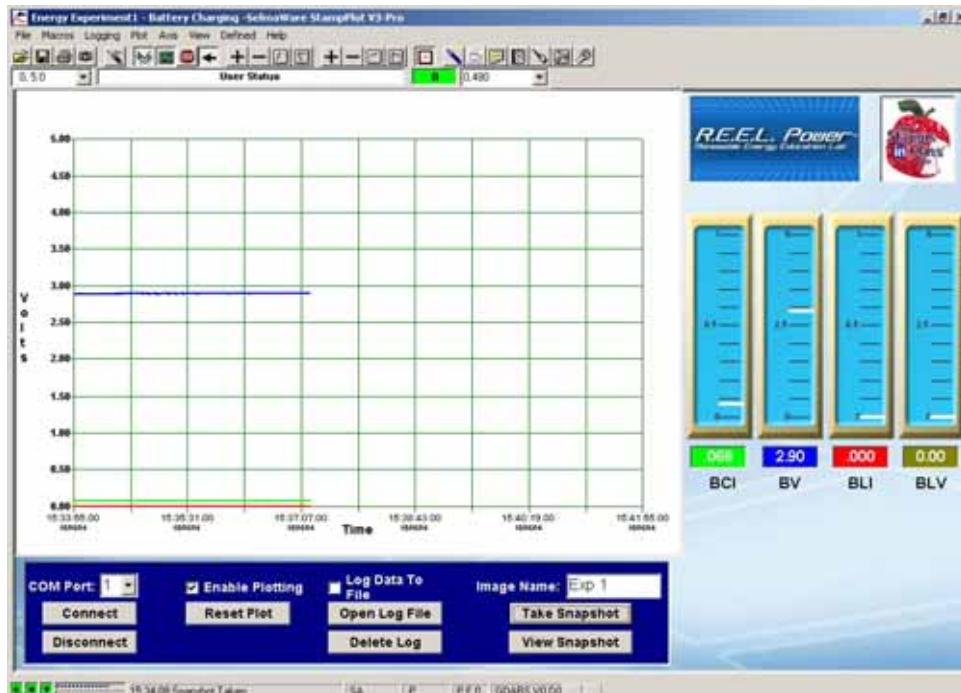


Figure 2-6: Plot of Programmable Battery Charger No-Load Voltage Output

While the batteries are charging, let's take a look at how the programmable battery charger works.

```

Exp_1_Charge:
    IF (doneCharging = True) THEN GOTO Exp_1_End

    HIGH ChargeBatt
    LOW DrainBatt
    TOGGLE ChargeLed
    LOW DrainLED
    LOW ReplayLED
  
```

The first line of code checks the `doneCharging` flag. If it's `True`, the program simply exits. Otherwise, a `HIGH ChargeBatt` activates Charge transistor circuit, which allows current to begin flowing into the batteries (refer to Figure 2-3). Correspondingly, the `LOW DrainBatt` deactivates the Discharge transistor circuit, disabling the batteries to drain or discharge. The `TOGGLE` instruction causes the green charge LED to flash each time through this part of the subroutine. And finally, the last two instructions deactivate the other two LEDs. Now to the next group of instructions...

```
a2dMuxId = a2dMuxId3
GOSUB A2D
ch3 = (255-a2dResult)
avgCurrent = avgCurrent+ch3

a2dMuxId = a2dMuxId2
GOSUB A2D
ch2 = a2dResult
avgVolts = avgVolts+ch2

IF (ch2 > BattFullChg) AND (dataPtr <10 ) THEN
    doneCharging = True
    HIGH ChargeLed
    LOW DrainBatt
ENDIF
counter = counter - 1
IF counter <> 0 THEN Exp_1_End
```

The first two instructions set the A/D Converter channel to `ch3` and then take a voltage sample. While still a voltage sample, this is actually a measure of BCI, the battery charge current, based on the voltage drop across the $10\ \Omega$ resistor connected to Vdd. The voltage drop across this resistor is the difference between Vdd, which is 5.00 volts (or 255 as it would be measured by the A/D converter) minus the sampled voltage (`a2dResult`). The third instruction computes this result and sets `ch3` equal to it. The StampPlot macro further adjusts this voltage reading to the equivalent current value. Finally, the fourth instruction adds the value of `ch3` to the `avgCurrent`. Recall that `avgCurrent` and `avgVolts`, coming up next, were both initialized to zero in the `Exp_1_Init` subroutine, so we're starting off our averaging cycle with a clean slate.

The next four instructions sample the battery voltage directly and store the result to `avgVolts`, with A/D channel `ch2` displayed as BV, or battery voltage, on StampPlot.

Following this, the **IF** statement is placed here to protect from overcharging the batteries by comparing the current battery voltage to the value of **BattFullChg** (CON 150 or 3.00 volts). However, this event must occur early in the charging cycle when **dataPtr** is less than 10. Or said another way, this event must occur within the first 256 seconds of the charge cycle. You'll see what we mean shortly. And setting **codePtr=2** will cause the **BRANCH** instruction to jump to the **Exp_1_Drain** subroutine the next time through.

The following instruction decrements the averaging counter (counter), which was initialized to zero in **Exp_1_Init**. If this is the first time through this subroutine from power on or reset, the counter value will be 255, since decrementing a byte set to zero causes the value to “roll over” to 255. Therefore, it will take another 255 decrements until the counter value is again zero. And this will take just over 64 seconds.

Why?

Because every time we exit from this routine via **Exp_1_End**, the program returns to **Main** and eventually winds up calling (**GOSUB**) the **Plot_It** routine where there is a 250-millisecond delay (**PAUSE 250**). So by multiplying 256 counter states times 250 milliseconds per state, the result comes out as 64 seconds. The reason it takes slightly longer than 64 seconds is because the **Main** routine must also call the other four routines (**Exp_2** through **Exp_5**) before it calls the **Plot_It** routine. And since the other four routines just have **RETURN** instructions at this time, the extra time is quite short. There is also a slight time delay within this routine, as well; that is, from the **BRANCH** instruction to **Exp_1_Charge** and then to the test for counter $\neq 0$ where the jump is made to **Exp_1_End**. However, after 256 passes though **Exp_1_Charge**, counter will again equal zero and the program will fall through to the code below.

```

        WRITE dataPtr,avgVolts.HIGHBYTE
        dataPtr = dataPtr+1

        WRITE dataPtr,avgCurrent.HIGHBYTE
        dataPtr = dataPtr+1

' IF avgVolts.HIGHBYTE > maxVolts THEN maxVolts=avgVolts.HIGHBYTE
' IF avgVolts.HIGHBYTE <= maxVolts-10 THEN codePtr=2:GOTO
Exp_1_End

        avgVolts = 0
        avgCurrent = 0
    
```

```

IF (dataPtr = DataPtrMax) THEN
    doneCharging = True
    HIGH ChargeLed
    LOW ChargeBatt
ENDIF
GOTO Exp_1_End

```

The first instruction writes the “average” value of the current samples (`avgCurrent.HIGHBYTE`) to EEPROM at the current `dataPtr`. Then the `dataPtr` is incremented and the third instruction writes the “average” value of the battery voltage (`avgVolts.HIGHBYTE`) to EEPROM, followed by the `dataPtr` being incremented again. However, one might ask at this point why `avgCurrent.HIGHBYTE` and `avgVolts.HIGHBYTE` are the “average” of these values. Remember, `avgCurrent` and `avgVolts` are both 16-bit words, and we have been adding 8-bit bytes to them 256 times. So to get an average value, one would normally divide `avgCurrent` and `avgVolts` by 256. We could have done this, however it would have required two unnecessary divide instructions. Instead, we decided to use another programming trick and simply use the high byte of `avgCurrent` and `avgVolts` as the average. We know this will work because the average value of all 256 samples will appear in the upper (high) byte of the word. We will lose the remainder of the average, however this is not important to us since our average values must fit into the limits of an 8-bit byte such as `ch3` and `ch2` so that it can be displayed on StampPlot. If you’re still not convinced, try adding up 256 byte values in hexadecimal and then see what the high byte comes to. Another neat programming trick!

Now back to the remainder of the charging part of the code.

Immediately following the writes to EEPROM, the next two instructions are “commented out”.

```

' IF avgVolts.HIGHBYTE > maxVolts THEN maxVolts = avgVolts.HIGHBYTE
' IF avgVolts.HIGHBYTE <= maxVolts - 10 THEN codePtr=2:GOTO Exp_1_End

```

We did this intentionally since the code would have ended our charging cycle prematurely. Instead, we want to charge our batteries on a purely timed basis at first, and then uncomment these two lines of code when we know things are working as planned. We’ll do this towards the end of this experiment.

The next two lines of code clear the `aveCurrent` and `aveVolts` for the next 256 samples, and the `IF...THEN` statement checks `dataPtr` against `DataPtrMax`. If equal then `doneCharging` is set `True` along with the `ChargeLed` being constantly set to `HIGH` and the `ChargeBatt` pin set to `LOW`. Then the routine then exits to the `Main` loop again.

The final lines of code for the charge routine simply clear the `avgCurrent` and `avgVolts` for the next 256 samples. Then `dataPtr` is tested for maximum. If it is at our predefined `DataPtrMax` value, then `codePtr` is set to 2 so that the program will branch to `Exp_1_Drain` the next time through. Otherwise, `codePtr` is left where it's at (`codePtr=1`) and the program will once again branch to `Exp_1_Charge`.

Let's take a look at a sample plot of the charging cycle as illustrated in Figure 2-7 below.

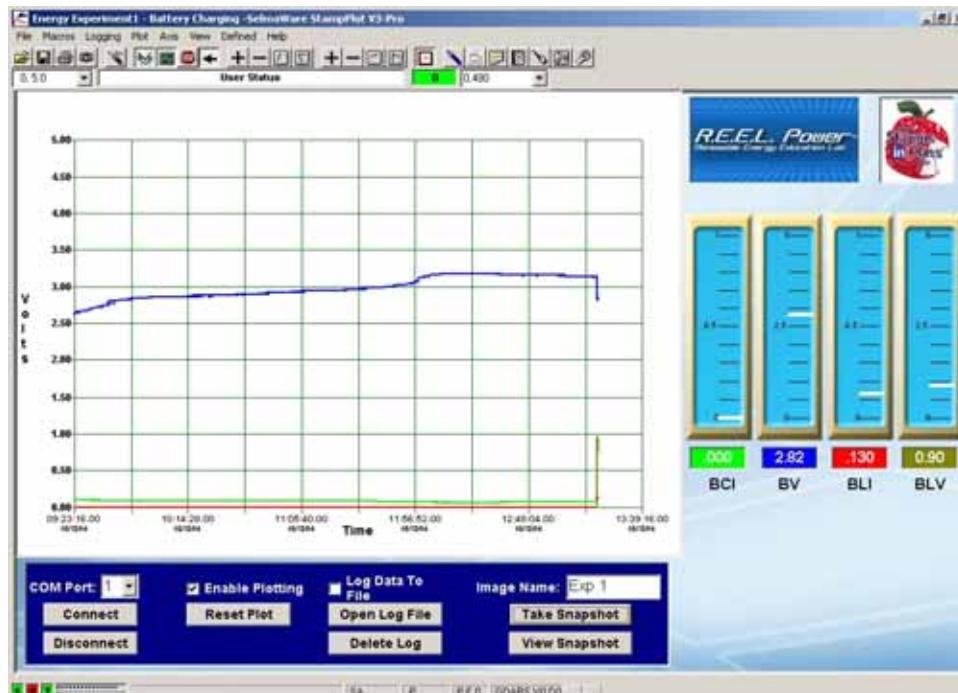


Figure 2-7: Plot of Entire Battery Charge Cycle, with Adjusted Time Scale

This particular plot took almost four hours to generate, so the time scale was expanded to accommodate the necessary time (note the time axis at the bottom of the plot). As you can see, the battery voltage climbs from its normal 2.40 volt value to over 3.00 volts and then drops down again slightly. This is the normal charge cycle, and the voltage “hump” above the 3.00 volt level can be used to detect when the charge cycle is complete. In the code thus far we have purposely ignored detecting this “hump” and have, instead, opted for a timed charge cycle, exclusively. And we did this for a reason; that is, to portray the entire charge cycle without overcharging the batteries and, also, to determine just how an “end of charge” program could be done. This was done by your author with some amount of trial but, fortunately, no error. The time it takes to generate this charge cycle is based partly on the counter variable, the `dataPtr` variable as well as the 250-millisecond delay in the `Plot_It` routine. Here’s how to compute the timed charge cycle:

- 256 counts (`counter`) x .25 s/count (`Plot_It`) x 200 (`dataPtr`) = 12,800 s
- 12,800 s / 3,600 s/hour = 3.56 hours

Also recall from a few sentences ago the “commented out” code. These two lines of code can be used to detect when the charge cycle is complete. Keep them commented out for now and we’ll uncomment them near the end of the experiment.

ACTIVITY #6: REPLAYING THE CHARGE CYCLE

This activity involves replaying the charge cycle in a rapid, fast forward manner. Recall from `Exp_1_Charge` that it took nearly 4 hours to charge the batteries and that during this time, we also took a 64-second average of the Battery Charge Current (BCI) and Battery Charge Voltage (BVI) and stored these values in EEPROM. As a matter of fact, we stored exactly 200 samples, each, of BCI and BVI in EEPROM. This is because `DataPtrMax` is limited to 400, so 400 divided by 2 samples/ EEPROM entry = 200 EEPROM entries approximately every 64 seconds. Therefore, this effort took nearly 4 hours by computations shown previously.

With our average values stored in EEPROM, we can replay the last charge cycle, over and over, in about 50 seconds. To show you what we mean, simply depress and release the reset button until the yellow LED begins to flash. Assuming that StampPlot is up and running, you should witness something like that in Figure 2-8 below.

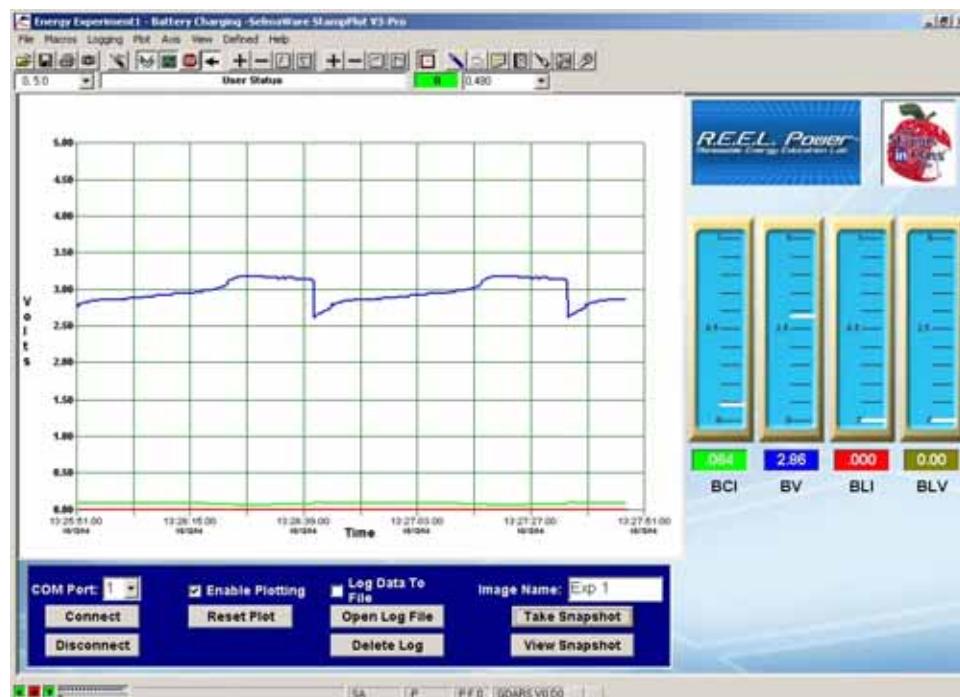


Figure 2-8: Plot of Battery Charge Replay

If everything is working correctly, you should see the plot of the charging cycle repeating over and over again with the blue plot being the battery voltage (BV) and the green plot being the battery charge current (BCI). The other two plots are at zero since nothing has been recorded for them. To see something similar to the above, you will have to decrease the StampPlot time span to about 2 minutes. Do this by clicking on the second minus icon of the StampPlot toolbar about 2 or 3 times.

Take particular notice of the blue battery voltage plot and the relatively quick jump made from the 3.00 volt level to about 3.20 volts and then back down again slightly. This is indicative of a fully charged set of batteries. The abrupt drop in voltage towards the end of the plot in Figure 2-8 is where charging ends and discharging begins and is not meant to be an illustration of the battery's normal end of charge cycle. The reason the NiCad batteries exhibit this type of plot while being charged is due to the fact that when the charge is complete, the nickel electrode begins generating oxygen. The oxygen then

diffuses through the separator and reacts with the cadmium electrode that, in turn, causes the lowering of the cell voltage, which can be used to detect the end of the charge cycle.

Let's look at the code that makes our charge replay cycle work.

```
Exp_1_Replay:  
  
    LOW ChargeBatt  
    LOW DrainBatt  
    TOGGLE ReplayLed  
    LOW ChargeLed  
    LOW DrainLed  
  
    READ dataPtr,avgVolts.LOWBYTE  
    ch2 = avgVolts.LOWBYTE  
    dataPtr = dataPtr+1  
  
    READ dataPtr,avgCurrent.LOWBYTE  
    ch3 = avgCurrent.LOWBYTE  
    dataPtr = dataPtr+1  
  
    IF (dataPtr => DataPtrMax) THEN  
        dataPtr = 0  
        HIGH ReplayLed  
    ENDIF  
  
Exp_1_End:  
    RETURN
```

You've seen the first five instructions before; this is just a variation on them to disable the battery charging and discharging and to flash the yellow LED.

The next three instructions read the average battery voltage stored in EEPROM and set **ch2** equal to that value for the **Plot_It** routine. The **dataPtr** is incremented to point to the corresponding current value that is read by the next instruction. With **ch3** set to the average current the **dataPtr** is once again incremented to point to the next set of voltage-current readings in EEPROM. Finally, **dataPtr** is tested against **DataPtrMax**. If it's at or beyond the maximum, it's reset to zero and the process repeats. And, once again, the only way out of the **Exp_1_Replay** routine is to depress and release the reset button or cycle power to the Board of Education.

If you started this chapter with brand new batteries and did not have much of a chance to play with the discharge function, you may want to return to Activity #4, and discharge your batteries fully before moving on to the last Activity.

ACTIVITY #7: DETECTING THE END OF CHARGE

As an apt conclusion to our battery charger program, we leave it up to you to experiment with automatically detecting when the charge cycle is complete, rather than just depending on a fixed time charge that we did here. We have given you a head start in this area by supplying two lines of code in the `Exp_1_Charge` routine, which are repeated below:

```
' IF avgVolts.HIGHBYTE > maxVolts THEN maxVolts = avgVolts.HIGHBYTE  
' IF avgVolts.HIGHBYTE =< maxVolts - 10 THEN codePtr=2:GOTO Exp1_End
```

Uncomment these two lines and reload the entire program and try it for yourself. Your results will depend mainly on the amount of voltage drop from `maxVolts`. We chose a value of 10 that corresponds to 0.20 volts (10×0.02 volts/A-D Converter step). Therefore to work, your batteries will have to drop at least 0.20 volts for the charge to automatically end. You can certainly change this value by making it anything you want, within reason. We wish you luck and good success in your trials.

SUMMARY AND APPLICATIONS

This experiment introduced you to a popular and useful device to know about... a programmable battery charger. With this basic circuit and program you can go on to create other, more exotic battery charging circuits and programs for yourself and, quite possibly, for a company that hires you. As we mentioned before, there are hundreds if not thousands of battery chargers on the market, each with its own particular niche in the marketplace. The key, then, is to find the right rechargeable battery with the right niche application and then design your next programmable battery charger around that need. Here are some ideas.

Real World Examples

Marine batteries, that is, rechargeable batteries that are built for heavy-duty marine or boating applications, require specialized charging. This is indicated when you see written all over the battery the words “deep cycle”. So what does deep cycle mean? In short, a deep cycle battery demands more than just a “surface charge” which is what your car

battery generally gets as it motors along the highway. A deep cycle marine battery needs a special charging cycle to allow it to produce more amp-hours of capacity in order to start powerful diesel engines, as well as run on-board inverters that “quietly” power AC appliances like lights, refrigerators and air conditioners while at anchor. So, if you take the time to examine what a deep cycle battery needs in terms of charging, you may come away with ideas to improve on a particular charger design. That gets into engineering applications, which is where we hope you’ll apply the knowledge of what you’ve just learned.

Another real world battery charging application involves the fast charging of model airplane batteries. Most radio controlled model airplanes, and for that matter radio controlled boats and cars, use rechargeable batteries that move servos to control the rudder, ailerons and wheels. Of great area of interest to this vast hobby segment is an inexpensive (cheap!!) but effective and reliable fast-charge battery charger. When batteries go dead, the hobbyist wants to recharge them in a hurry. And it seems that every year a new quick charger comes to market to embrace the new battery technologies. So here’s another opportunity to use what you’ve learned to design the next, World’s-greatest, lightning-fast, super-cheap battery charger.

And if you haven’t guessed by now, you have just built and programmed a battery charger that can charge the NiCad version of all those regular alkaline batteries that you just use and throw away. Think of the money you’ll save, not to mention that you can now monitor HOW your NiCad batteries are being charged. Then if you’re really enthused, you can modify both the circuit and code to adapt the battery charger to different types other than just NiCads.

More Information

Here’s where to find more information on batteries and battery chargers:

www.learnonline.com/ewre.html.

Chapter 3: Experiment #2, Dueling Solar Cells

Your second experiment deals with solar cells that produce DC power directly from sunlight. Solar cell technology has been around for about fifty years and was developed primarily for our early space program to power orbiting satellites. We've come a long way since then in terms of technology improvements and applications for solar cells. But one thing that is overlooked by the general public is the fact that electricity can be produced directly from sunlight, which is truly a marvel by itself. Just consider pointing a slab of material at the sun and getting electricity directly from it, something that Thomas Edison probably never thought possible. And consider the "Third Law" of Arthur C. Clarke, the author of *2001 – A Space Odyssey* and a pioneer in communications satellites in the 1960's: "Any sufficiently advanced technology is indistinguishable from magic". Solar cells definitely fit into this category.

Now consider powering your house completely from the sun some ten or twenty years from now; that is, without any connection to the conventional AC power grid. Sound impossible? We don't think so. As solar cell technologies continue to advance beyond today's developments, this promise can come true, but only if inspired people take hold of the current solar cell technology and move it ahead to this future point. As a student or experimenter, you are in a position to do just that; that is, improve on solar cell designs so that you and your (future) family can enjoy the benefits and advantages of clean, un-polluting power. Hopefully, this experiment and the one that follows, "Building a Solar Cell Sun Tracker", will inspire you to meet this challenge and make us all less dependent on conventional energy.

HOW SOLAR CELLS WORK

While this section will provide you with a basic understanding of how modern solar cells function, most of the detailed information is available on the World Wide Web. The web links posted at the end of this section are far more complete in terms of information and illustrations. You are encouraged to examine these links if you want to really understand how this wonderful technology evolved and how it continues to grow and expand.

The Basics

Solar cells are quiet, non-polluting energy converters. They convert one form of energy, light, to another form of energy, electricity. When light energy is reduced or stopped, as when a cloud passes in front of the sun or when the sun goes down in the evening, then

the conversion process slows down or stops completely. When the sunlight returns, the conversion process immediately resumes. Solar cells do not store electricity; they just convert light to electricity when sunlight is available. To have electric power at night, a solar electric system needs some form of energy storage, usually batteries, to draw upon.

What is marvelous about solar cells is that they perform this conversion without any moving parts, noise, pollution, radiation or constant maintenance. These advantages are due to the special properties of semiconductor materials that make this conversion possible. The following discussion will guide you through the conversion process without any mathematical equations or complicated physics.

Semiconductors – The “Current” Solar Cell Material Of Choice

Today the majority of solar cells are made from silicon. While other materials are also used, the fundamental process of how solar cells work is the same as for silicon cells, like transistors. Silicon is basically a “semiconductor” or “semi-material” whose name reflects that it has properties of both a metal (a conductor) and an insulator (a non-conductor). In a conducting metal, like copper, atoms have loosely bound electrons that easily flow when a voltage is applied. Conversely, atoms in an insulator, like glass, have tightly bound electrons that cannot flow even when a strong electric voltage is applied. In between metals and insulators, atoms in a semiconductor material bind their electrons somewhat tighter than metals, but looser than insulators. Another feature of semiconductors is that introducing small amounts of impurities, called “dopants,” into the semiconductor structure can vary the amount of electrons that are available to break away from their atoms and flow under electric voltage. Two dopant elements that are typically used in silicon are phosphorous and boron.

A Little History

In the 1950s, scientists tinkering with semiconductors found that by introducing small, minutely controlled amounts of dopant impurities to the semiconductor matrix, the density of free electrons could be shepherded and controlled. The dopants, similar enough in structure and valence to fit into the matrix, have one electron more or less than the semiconductor. For example, doping with phosphorus, which has five valence electrons, produces a (negative) n-type semiconductor, with an extra electron, which can be dislodged easily. Aluminum, boron, indium, and gallium have only three valence electrons, and so a semiconductor doped with them is (positive) p-type, and has “holes” where the missing electrons ought to be. These holes behave just like electrons, except that they have an opposite, positive charge. (Holes are theoretical, but so are electrons

and either or both may or may not exist, but we know for sure that if one exists, they both do, because we can't create something out of nothing in the physical world!) It is important to understand that, although loosely bonded or extra carriers exist in a substance, it is still neutral electrically, because each atom's electrons are matched one for one by protons in the nucleus.

The fun begins when the two semiconductor types are intimately joined in an NP-junction, and the carriers are free to wander. Being of opposite charge, they move toward each other and may cross the junction, depleting the region they came from and transferring their charge to their new region. This produces an electric field, called a gradient, which quickly reaches equilibrium with the force of attraction of excess carriers. This field becomes a permanent part of the device, a kind of slope that makes carriers tend to slide across the junction when they get close. Figure 3-1 shows a typical solar cell.

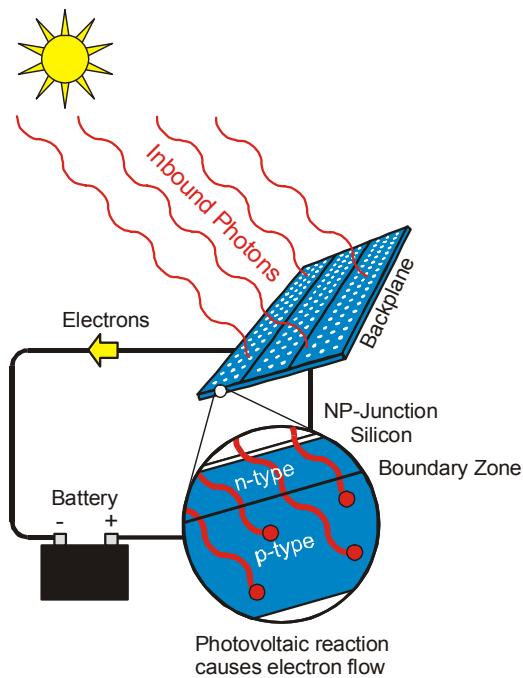


Figure 3-1
Typical Solar Cell

When light strikes a photovoltaic cell, atoms are bombarded with photons, and give up electrons. When an electron gets lopped off an atom, it leaves behind a “hole”, which has an equal and opposite charge. Both the electron, with its negative charge, and the hole, with its positive charge, begin a random walk generally down the gradient. If either carrier wanders across the junction, the field and the nature of the semiconductor material discourage it from re-crossing. A proportion of carriers which cross this junction can be harvested by completing a circuit from a grid on the cell's surface to a collector on the backplane. In the cell, the light “pumps” electrons out one side of the cell, through the circuit, and back to the other side, energizing any electrical devices (like the battery in Figure 3-1) found along the way.

Solar Cell Efficiency

If asked about how much sunlight energy gets converted into electricity, the best commercial solar cells are about 25% efficient, however the typical value is around 15% or less. Why so low? The answer lies in the electromagnetic spectrum of sunlight, which is composed of many wavelengths. We can see some of these wavelengths as visible light while other wavelengths fall below (infrared) and above (ultraviolet) the visible spectrum. Some wavelengths won't have enough energy to move electrons inside the solar cell's material and other wavelengths have too much energy.

The Solar Spectrum: The vast majority of the Sun's radiation is emitted at wavelengths between 290 and 3,200 nanometers. The spectrum of visible light falls within this range. Table 3-1 shows the wavelengths for common colors along with the outer boundaries in the ultraviolet and infrared ranges.

i

Table 3-1: Colors and Approximate Wavelengths of the Solar Spectrum (Values in Nanometers)

Color	Wavelength	Color	Wavelength
Ultraviolet	290-390	Orange	630
Violet	400	Red	780
Blue	470	Near infrared	800-1000
Green	565	Infrared	1000-2000
Yellow	590	Far infrared	2000-3,200

In effect, the solar cell's material can react to only certain energy bombardments from sunlight in order to produce electricity. By changing the solar cell materials the efficiency

may, or may not, improve. This is where a major part of solar cell research is being conducted. The ideal solar cell material, or materials, would absorb more of the wavelengths of sunlight thus producing that much more electrical energy.

Other losses also enter into the efficiency figure of solar cells. One conventional loss has to do with the internal resistance of the material itself. Most silicon-based solar cells have a high internal resistance, which serves to reduce the amount of usable power that is produced by the cell. Another related electrical loss involves where to put the wires that connect to the cell's energy producing material. If the wires are placed at the edge of the cell, then electrons on the opposite edge have further to travel in the high resistance material thus creating additional energy loss. Most modern solar cells use a grid of conductors on the surface to more efficiently route the photovoltaic energy to the wires. However, this grid on the surface of the solar cell also blocks the sunlight from entering the cell's material. So as you can see, creating an efficient solar cell is really a matter of selective "trade offs" or compromises.

Future Solar Cells

The most futuristic approach to new solar cell technology involves arranging nanosize semiconductors in a matrix of plastic-like materials that are expected to be much less expensive to produce. Companies like Nanosys, Inc. are working on "nanorods" that are just 7 nanometers by 60 nanometers in a polymer. Because of their size (a nanometer is about 10,000 times narrower than a human hair) nanorods are arranged by chemical reactions. The manufacturing of nanocomposite solar cells is more like the production of photographic film, which is done in extremely high volumes with miles of precisely engineered materials per day at extremely low costs. According to a company official, the efficiency should be on par with crystalline silicon within three years. Plus, it will be easy to simply "paint" the solar cell material onto existing surfaces like outdoor walls and rooftops, thus making the entire structure produce power when the sun is shining. Yet there are doubts over how quickly such technology might be on the market.

On the horizon: a virtually perfect solar cell! An unexpected discovery at the Department of Energy's Berkeley National Laboratory (LBNL) may push the implementation of photovoltaics (PVs) sooner than later. A serendipitous discovery by the lab about the electronic properties of indium nitride may eventually yield high efficiency solar PVs, by making use of the entire spectrum of the Sun's radiation. While researching the properties of LEDs (light emitting diodes) researchers found that the band-gap energy of indium nitride is 0.7 electron volts, much lower than the 2.0 electron volts previously expected.



Band-Gap Energy: The band-gap energy is the amount of energy needed to free an electron from its atom; a solar cell material can only capture sunlight at energies equal to or greater than its band-gap energy.

The low band-gap energy of indium nitride means that it can capture sunlight at much lower energies than expected, so the material is able to capture low-frequency, red or infrared light, a broader spectrum of radiant energy. This discovery may lead to two-layer PVs that could reach a phenomenal 50% efficiency in converting sunlight to electricity.

Work is proceeding to capitalize on newer, less expensive and more efficient materials and techniques that can make solar cells as ubiquitous as light bulbs in our homes, schools and offices. Much more work and effort remains, so it's not too late to think about a career in this exciting field.

Credit

A portion of this information was reprinted with permission from *The Independent Home* by Michael Potts. His book and many other interesting titles can be obtained from the publisher Chelsea Green at www.chelseagreen.com.

ACTIVITY #1: BUILDING THE DUELING SOLAR CELLS

We named this experiment “Dueling Solar Cells” since we will use two separate, but equivalent, solar panels to measure and compare their individual voltage outputs. We will be removing the Battery Charger circuit, and adding the solar cells and two indicator LEDs to our Analog to Digital Converter circuit.

Parts Required

- (2) Red LEDs
- (2) 470 Ω resistors
- (1) Solar Cell Assembly (from the Solar Kit)
 - 8 solar cells
 - 1 plastic tray
 - 7 spanner bars
 - 16 lock washers
 - 16 nuts
 - 1 spanner wrench
- (3) 30" jumper wires, each with 1 tinned lead

Assembling the Solar Panel

The Solar Kit comes with solar cells that are placed in a convenient tray and are not yet hooked up in any particular configuration. The kit also comes with instructions that describe various hookups for series and parallel arrangements. For this Activity, please set aside these kit instructions and use the ones below. We will be arranging the eight cells into two solar panels, each composed of four solar cells connected in series.

- ✓ Lay the solar cells in the black tray in the configuration shown in Figure 3-2. Viewed from the back of the tray, the (+) and (-) symbols on the cells will be visible through the round holes to assist in the correct placement.
- ✓ Place the metal spanner bars over the solar cell screws in the arrangement shown.

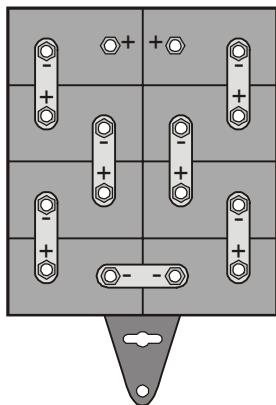


Figure 3-2
Solar Cell Array Layout

Position the solar cells as shown, paying careful attention to their positive and negative terminal positions before connecting them with the spanner bars.

- ✓ Place a lock washer and hex nut over each screw and tighten in place. Make sure to use the supplied plastic spanner wrench to tighten the screws. If you experience low or no voltage during the course of the experiment, a loose connection is probably the cause.

Building the Dueling Solar Cells Circuit

- ✓ Remove the Battery Charger Circuit that was built in the last chapter, but leave the A/D circuit in place.
- ✓ Add the two LED-resistor circuits to the breadboard as shown by the schematic in Figure 3-3 and the wiring diagram in Figure 3-4. Make sure not

to short out the LED wires to each other and, also, tip the LEDs away from each other. This will make it easier to understand what happens when the left and right solar panels are compared with one another.

- ✓ Strip about 1" of coating off of the non-tinned ends of the 30" jumper wires.
- ✓ Connect the stripped end of one 30" jumper wire to the mutual negative terminal of each solar array, and the tinned end to pin 7 of the ADC0834 (ground).
- ✓ Connect the bare ends of the other two 30" jumper wires to the separate positive terminals of the solar arrays, and the tinned ends to the breadboard. One wire should connect the left array to pin 4 (channel 1) of the ADC0834, the other wire should connect the right array to pin 5 (channel 2).

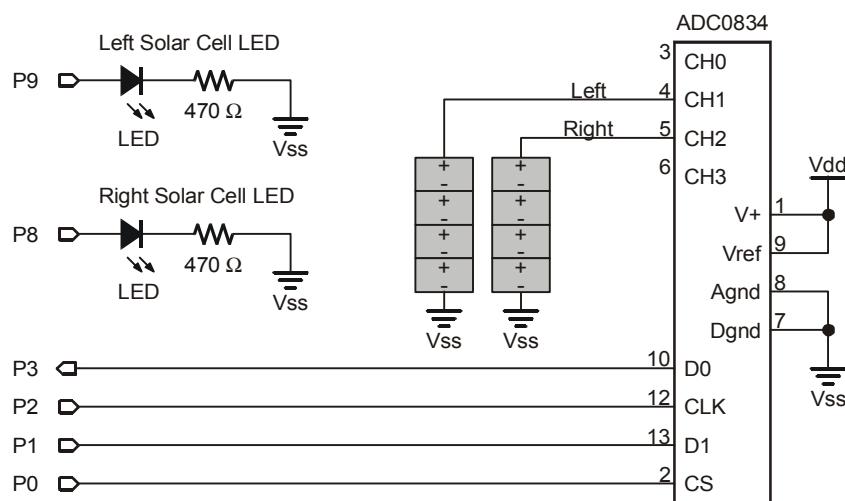


Figure 3-3: Dueling Solar Cells Schematic

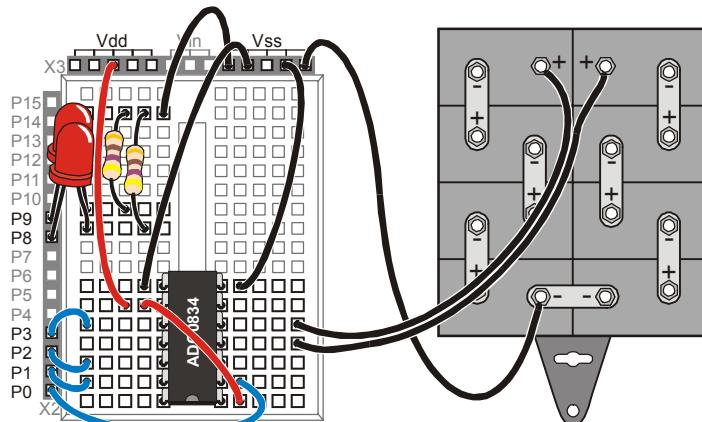


Figure 3-4: Dueling Solar Cells Wiring Diagram

ACTIVITY #2: PROGRAMMING THE DUELING SOLAR CELLS

We will be building upon our previous program, since every experiment uses the **Plot_It** subroutine. These additions let us read and compare the voltages generated by the two solar arrays. You can either download *DuelingSolarCells.bs2* from www.parallax.com, or you can update your current program by following the steps below.

- ✓ In the BASIC Stamp Editor, open BatteryCharger.bs2.
 - ✓ Rename and save the program as DuelingSolarCells.bs2
 - ✓ Update the title to read as follows:

' Experiments with Renewable Energy v1.0 - DuelingSolarCells.bs2
' Indicates whether left or right solar cell has higher voltage reading.

- ✓ Add the following code to the bottom of the Declarations section.

```

' ----- For Experiment 2: Dueling Solar Cells -----
LeftSpLed    PIN      9                      ' Left Solar Panel LED on P9
RightSpLed   PIN      8                      ' Right Solar Panel LED on P8

offSet        VAR      codePtr 'Nib          ' Value proportional to A/D voltage
offset = 0

```

- ✓ Next, in the **Main** routine, add an apostrophe in front of the **GOSUB Exp_1** command as shown below. This will make the entire line a comment, and keep the program from performing the Programmable Battery Charger experiment, which would conflict with the Dueling Solar Cells experiment. In other words, we are visiting the second “floor” of our “building” without stopping on the first!

Now follow up with adding the code for Experiment 2 in the Subroutines section:

- ✓ First add the commented lines to document the subroutine:

```
' Experiment 2: Dueling Solar Cells
' -----
'
' Convert the left solar panel voltage to an 8-bit value
' Convert the right solar panel voltage to an 8-bit value
' Compare the two voltage values:
'   IF both are equal plus or minus offSet
'     Then illuminate both LEDs
'
'   IF the left solar panel voltage > the right solar panel voltage
'     Then illuminate the left LED and extinguish the right LED
'
'   IF the right solar panel voltage > the left solar panel voltage
'     Then illuminate the right LED and extinguish the left LED
'
' Return
```

- ✓ Then fill out the `Exp_2` subroutine as shown:

```
Exp_2:  
    a2dMuxId = A2dMuxId1          ' convert A/D Channel 1  
    GOSUB A2D                      ' which is the left solar panel  
    ch1 = a2dResult                ' set CH1 = converted value
```

```

a2dMuxId = A2dMuxId2           ' convert A/D Channel 2
GOSUB A2D                         ' which is the right solar panel
ch2 = a2dResult                   ' set CH2 = converted value

Exp_2_Compare_CH2:
IF (ch1 > ch2) THEN
  GOTO Exp_2_CH1_Is_Greater
ELSEIF (ch2 > ch1) THEN
  GOTO Exp_2_CH2_Is_Greater
ELSE
  GOTO Exp_2_CH1_Equals_CH2
ENDIF

Exp_2_CH1_Equals_CH2:
HIGH LeftSpLed
HIGH RightSpLed
GOTO Exp_2_End                     ' both CH1 and CH2 are equal, so
                                         ' illuminate the left LED and
                                         ' illuminate the right LED
                                         ' and exit

Exp_2_CH1_Is_Greater:
ch1 = ch1 - offSet
IF (ch1 = ch2) THEN
  GOTO Exp_2_CH1_Equals_CH2
ELSE
  HIGH LeftSpLed
  LOW RightSpLed
  GOTO Exp_2_End                   ' CH1 is > CH2, so
                                         ' subtract the offSet from CH1
                                         ' branch if CH1 = CH2, else
                                         ' illuminate the left LED and
                                         ' extinguish the right LED
                                         ' and exit

Exp_2_CH2_Is_Greater:
ch2 = ch2 - offSet
IF (ch2 = ch1) THEN
  GOTO Exp_2_CH1_Equals_CH2
ELSE
  LOW LeftSpLed
  HIGH RightSpLed
  GOTO Exp_2_End                   ' CH2 is > CH1, so
                                         ' branch if CH2 = CH1, else
                                         ' extinguish the left LED and
                                         ' illuminate the right LED
                                         ' and exit

Exp_2_End:
RETURN

```

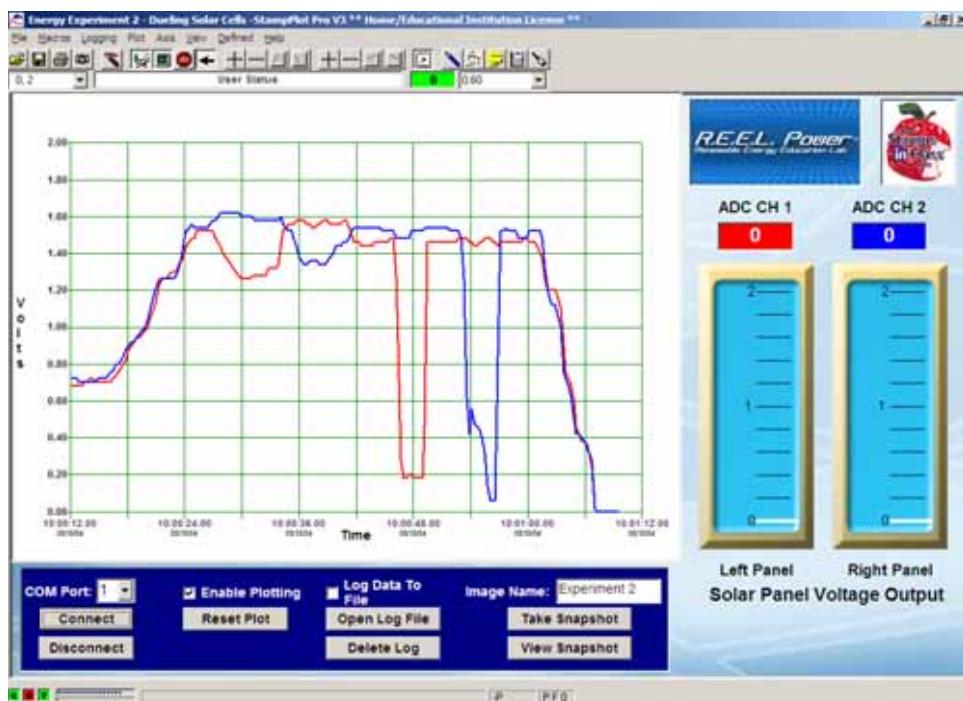
✓ Save your work!

Your Turn: Plotting the Dueling Solar Cells

If possible, have the solar cells exposed to bright sunlight for the next part of this test. If you cannot work near a sunny window, use a bright light bulb close to your solar cells for illumination. We have provided StampPlot illustrations for different light sources.

- ✓ Run the program DuelingSolarCells.bs2. The Debug Terminal should be displayed with the RX LED flashing.
- ✓ Make a note of which COM port the Debug Terminal is using.
- ✓ Close the Debug Terminal in preparation for displaying the solar panel voltages on the StampPlot screen.
- ✓ Open StampPlot via Start → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_2.spm. (If StampPlot is already open, you may select Macros from the toolbar, then Select Start-Up macro, and open sic_ewre_exp_2.spm from the SIC_Energy folder.)
- ✓ In StampPlot, set your COM port to the one that was being used by the Debug Terminal.
- ✓ Click on the Connect button, and make sure Enable Plotting is checked.
- ✓ Next, point the solar panels at a bright light source. DO NOT LOOK DIRECTLY INTO YOUR LIGHT SOURCE!
- ✓ Move the solar panel tray from left to right and vice versa. Try covering up parts of each panel in turn. As you do this, watch how your actions cause the LEDs illuminate, and how the StampPlot graph and voltage meter readings change.

Figure 3-5 was created by placing a solar panel tray underneath a 60 watt bulb on a common adjustable-arm desk lamp. The graph shows the bulb being brought down close to the panels, then the panels being tilted to each side. Then each solar array was covered up one at a time. Finally, both panels were slowly covered at the same time. Can you discern each motion from the graph?



3

Figure 3-5: Dueling Solar Cells Voltage Display

- ✓ Now place the solar cell tray on a flat surface so that both panels are about equally exposed to very bright light.

If both LEDs illuminate, then the program is indicating that the solar panel voltage outputs are nearly the same. If, however, either the left or right LED illuminates by itself the program is indicating that one or the other panel is generating the most voltage. If both LEDs always seem to be illuminated, cup your hands around the solar panel tray to block out any sidelight. Figure 3-6, generated using the 60 watt incandescent bulb, shows that our solar cells are not exactly matched.

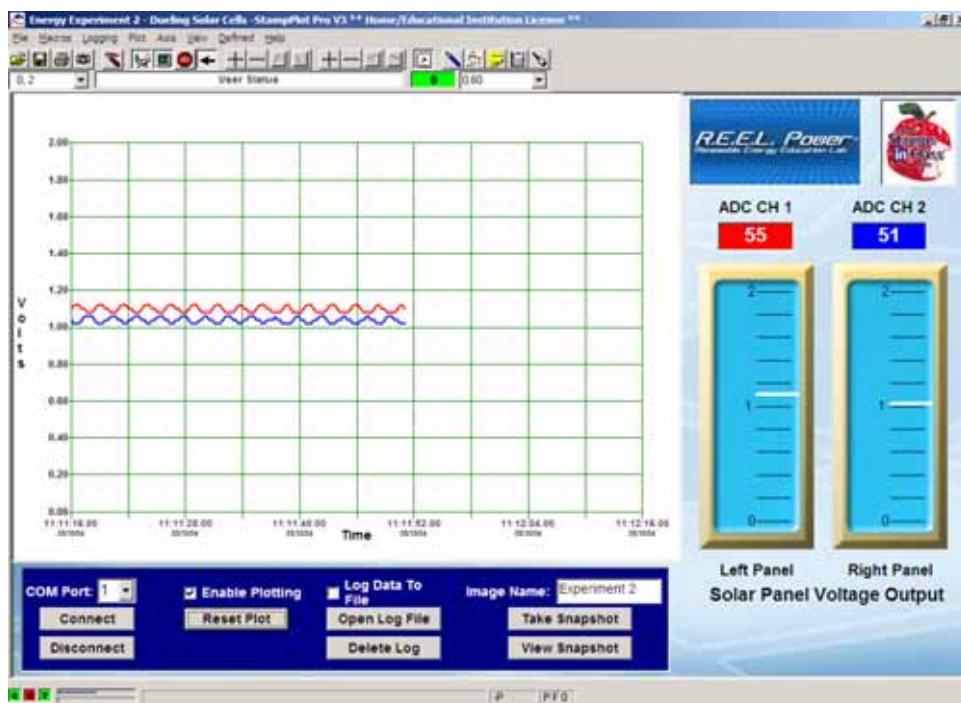


Figure 3-6: Voltage Display For Incandescent Light

Since each solar cell produces approximately 0.40 volts DC under no load, the combination of four cells can produce a maximum of 1.6 volts DC. If you are working indoors under a lamp, you most likely wouldn't get the full energy-producing effects of the sun on the solar cells, but you will still get a measurable voltage. Nevertheless, if the sun isn't shining and you want to do this experiment indoors, consider this option an advantage!

Now look at something else that's interesting in Figure 3-6 and ask yourself why the plotted voltages seem to ripple like sine waves. Once again, this plot was sampled indoors under incandescent light, not sunlight. The ripples, then, represent the 60 Hz AC that powers the incandescent light and modulates its intensity. Your eye probably can't detect this, but the solar cells certainly do.

If you choose to measure the period of the ripples on this plot, they do not match the period of a 60 Hz AC sine wave ($1/60 = 16.67$ milliseconds) simply because our sampling period for the A/D converter is timed by the **PAUSE 250** instruction in the **Plot_It** routine. Therefore, the plot shows selective voltage samples at different peaks and valleys of the 60 Hz AC over a longer period.

The next plot in Figure 3-7 created by placing the solar panels right underneath a 40 watt fluorescent tube. It also shows ripples. Fluorescent tubes also flicker (usually) faster than the human eye can perceive.

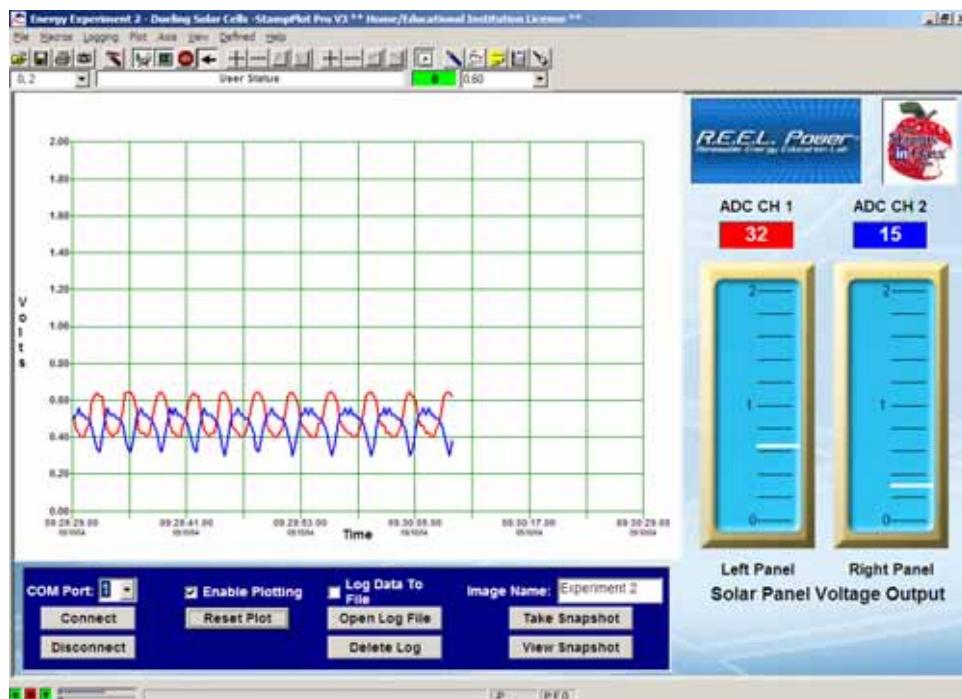


Figure 3-7: Voltage Display For Fluorescent Light

Now turn your attention to Figure 3-8. Here we used a laptop computer to set up the experiment where we had access to direct sunlight. The solar panels are pressed against a window with the full afternoon sun shining on them, although at a high angle which is why the panels are still not generating the maximum 1.6 volts DC.

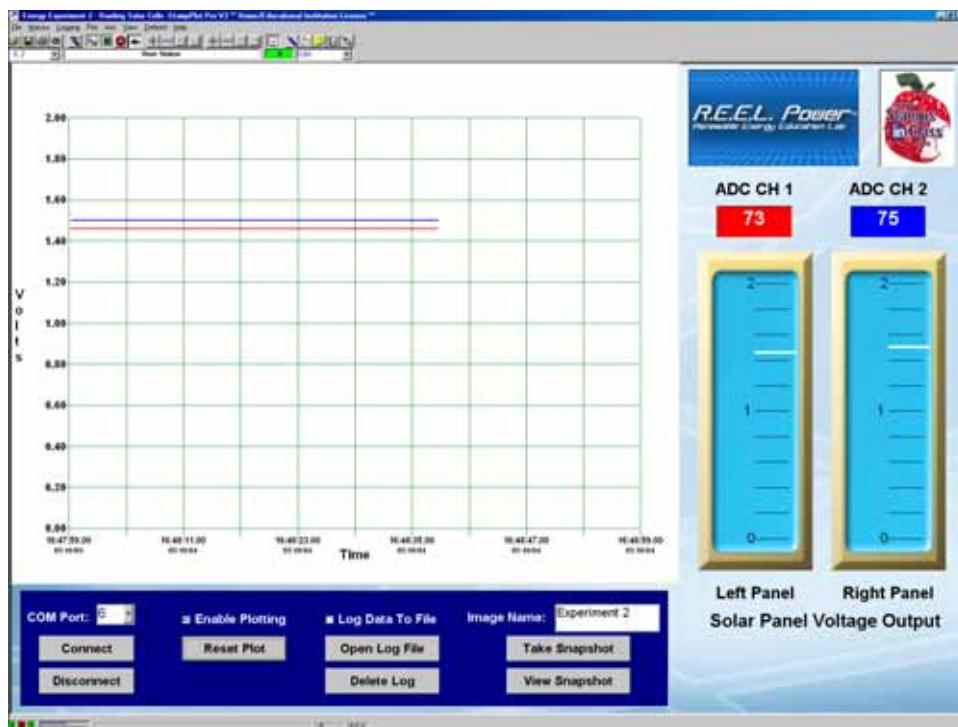


Figure 3-8: Voltage Display For Sunlight

Last, but not least, notice that both traces are holding steady but at slightly different voltages, given the fact that they are both stationary and pointing directly at the sun. This difference is expected since the individual cells that make up the left and right solar panels have slightly different voltage output characteristics due to the imperfections in their materials and manufacturing. But even so, what we want are the voltages from both panels to be equal when pointed at exactly the same light source. But how can we get there from here? Well, it's all in how our experiment is programmed.

ACTIVITY #3: HOW THE DUELING SOLAR CELLS WORK

There were only a few additions to the Declarations section for this experiment. The first two lines are **PIN** declarations giving names to the BASIC Stamp module I/O pins that will control the right and left solar panel indicator LEDs.

```

LeftSpLed    PIN      9
RightSpLed   PIN      8

offSet        VAR      codePtr 'Nib
offSet = 0

```

3

The next two lines declare offSet to be a nibble-sized variable, then initialize the value of that variable to zero.

Why are we saying `offSet VAR codePtr` instead of just saying `offSet VAR Nib`? Can you guess? We are re-using the same nibble of variable space that is utilized by `codePtr` in Experiment 1. Since we will not be performing both experiments at the same time, this technique allows us to make the most efficient use of variable space. You will see this technique employed again in the rest of the experiments as well.

Looking at the program code, the first three lines of the `Exp_2` subroutine sample the left solar panel and store the result in `ch1`. The next three lines do the same for the right solar panel with the result stored in `ch2`.

```

Exp_2:
a2dMuxId = A2dMuxId1
GOSUB A2D
ch1 = a2dResult

a2dMuxId = A2dMuxId2
GOSUB A2D
ch2 = a2dResult

```

Under the next label, the next four lines are using `IF...THEN...ELSEIF...THEN` commands to compare the sampled voltages, and use `GOTO` to branch to the appropriate label depending on which solar panel is receiving the most light.

```

Exp_2_Compare_CH1_CH2:
IF (ch1 > ch2) THEN
  GOTO Exp_2_CH1_Is_Greater
ELSEIF (ch2 > ch1) THEN
  GOTO Exp_2_CH2_Is_Greater
ELSE
  GOTO Exp_2_CH1_Equals_CH2
ENDIF

```

If neither statement is true, the `ELSE` command branches to the next instruction at label `Exp_2_CH1_Equals_Ch2`. And here is where things get interesting. Based on the comparison of `ch1` and `ch2` at label `Exp_2_Compare_CH1_CH2`, one of three conditions is true:

- `ch1` is greater than `ch2` - OR -
- `ch2` is greater than `ch1` or 2 - OR -
- `ch1` and `ch2` are equal

If the third condition is true (`ch1 = ch2`) the program branches to the code at the label `Exp_2_CH1_Equals_CH2`. Here both the left and right LEDs are illuminated with `HIGH` commands for the declared pin names, and then the program exits to `Exp_2_End`.

```
Exp_2_CH1_Equals_CH2:  
    HIGH LeftSpLed  
    HIGH RightSpLed  
    GOTO Exp_2_End
```

To make this happen, however, both sampled voltages must be “exactly” equal, a condition that StampPlot shows is not the case; that is, in terms of the left and right panels NOT producing exactly the same voltages given the same stable light source and position. To make arrive at this label would mean that the panel tray must have been “tilted” to favor one panel over the other, thus making the sampled voltages exactly the same. Since we want to use our dueling solar panels for our sun tracker in the next experiment, we need to come up with a way to adjust out this slight voltage difference between the two panels.

Enter the `offset` variable. As its name implies the `offset` variable adjusts for the minor differences in the left and right solar panel voltage readings. For example, assume that `ch1` is greater than `ch2`. The program will branch to the label `Exp_2_CH1_Is_Greater`. Here the `offset` value is subtracted from `ch1`. If this adjusted value is equal to `ch2`, the program assumes that both channels are equal in voltage with the program branching to `Exp_2_CH1_Equals_CH2`. However, if the adjusted value for `ch1` (`ch1 - offset`) is still greater than `ch2`, the program illuminates the left LED, extinguishes the right LED and exits.

```
Exp_2_CH1_Is_Greater:  
    ch1 = ch1 - offset
```

```

IF (ch1 = ch2) THEN
    GOTO Exp_2_CH1_Equals_CH2
ELSE
    HIGH LeftSpLed
    LOW RightSpLed
    GOTO Exp_2_End
ENDIF

```

3

The same logic applies when **ch2** is determined to be greater than **ch1**.

```

Exp_2_CH2_Is_Greater:
ch2 = ch2 - offSet
IF (ch2 = ch1) THEN
    GOTO Exp_2_CH1_Equals_CH2
ELSE
    LOW LeftSpLed
    HIGH RightSpLed
    GOTO Exp_2_End
ENDIF

```

Your Turn: Fine-Tuning the offSet Variable

We chose 0 for the **offSet** variable to establish a baseline for fine-tuning the voltage difference between both the left and right solar panels. Actually, **offSet** can be set between 0 and 15 (remember that **offSet** is defined as a 4-bit nibble, which only allows values between 0 and 15). For example, when **offSet** is set to 0, the left and right panel voltages must be exactly the same for both LEDs to illuminate. If you are fortunate enough to have solar cells that match up to this criteria, then leaving **offSet** = 0 is appropriate. Otherwise, **offSet** should be adjusted to match your individual solar panel's voltage differences. Here's how:

- ✓ To determine the correct value for **offSet**, place your solar panel tray on a flat stationary surface so that both panels have equal exposure to your bright light source.
- ✓ Verify that the BASIC Stamp Editor Debug Terminal is closed, then switch to the StampPlot screen and click on the Connect button, making sure that the Enable Plotting box is checked.
- ✓ Look at the values in the text boxes just above the vertical voltage gauges. These show the output value for channels 1 and 2 of the analog-to-digital converter.

- ✓ Subtract the higher number from the lower number. It doesn't matter if `ch1` or `ch2` is higher. Simply subtract the higher number from the lower number.
- ✓ Then take the result and divide by 2. Your divided result should be the correct `offset` value.
- ✓ If you get a fractional result when dividing (like $5 / 2 = 2.5$) then round the value up (3) or down (2); it doesn't matter since this is the closest you can come to equalizing both channels in terms of their voltage outputs.

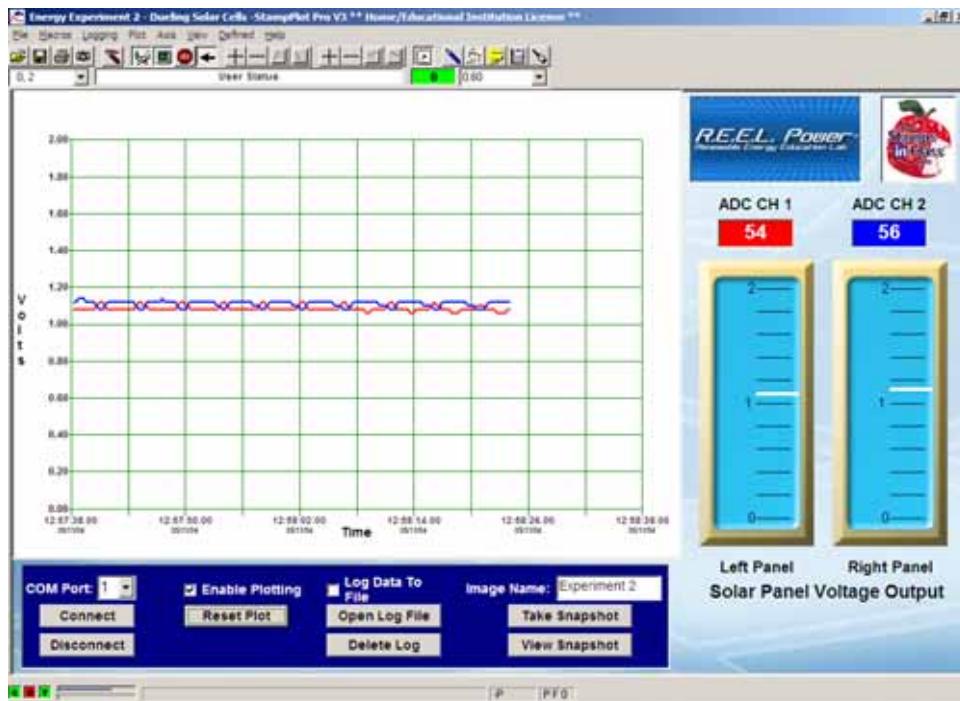


Figure 3-9: Solar Panels before offSet Voltage Adjustments Applied

For our solar panel output shown in Figure 3-9, we have $56 - 54 = 2$, then $2 / 2 = 1$. So, we will change the `offset` variable for our example from 0 to 1.

By dividing your `offset` value by two, you are actually establishing a midpoint converted voltage value. In other words, if you add the `offset` value to the lower

channel's converted voltage number, then subtract the `offset` value from the higher channel's converted voltage number, both numbers should be equal within the resolution of the A/D converter, which is 0.02 volts per count. For example, if the left channel's converted value is 38 (38×0.02 volts/count = 0.76 volts) and the right channel's converted value is 42 (42×0.02 volts/count = 0.84 volts), the `offset` value equals 2 [$(42-38)/2 = 4/2 = 2$]. Adding 2 to 38 equals 40 while subtracting 2 from 42 still equals 40. This is the method the program uses to adjust out the voltage differences between the left and right solar panels.

- ✓ To install the `offset` value into the program, exit the StampPlot screen by clicking on the Disconnect button to deselect it.
- ✓ Bring up the BASIC Stamp Editor and change the `offset` value in the Declarations section for Experiment 2.
- ✓ Next, save the program and re-load it into the BASIC Stamp module by clicking the Run icon or by typing Ctrl-R.

If the solar panels are still in the same place as when you determined the new `offset` value, the left and right LEDs should now be illuminated. One or the other LEDs may be flashing but generally both LEDs should remain illuminated.

- ✓ If not, go back to the beginning of this section and check your results for the `offset` value, again.
- ✓ If you've rounded down because of a non-integer result, try rounding up instead.

Notice in our example in Figure 3-10, both plots tend to converge with spikes here and there. This is because the program adjusted `ch1` or `ch2` by subtracting the `offset` that we computed. You should have similar results with your plot.

- ✓ Once again, look at the ADC values in the boxes above the voltage gauges on your plot.
- ✓ If `ch1` and `ch2` are within one count of each other, this is the best you can do to balance the solar panel voltages.
- ✓ If not, go back and readjust the `offset` value until this happens.

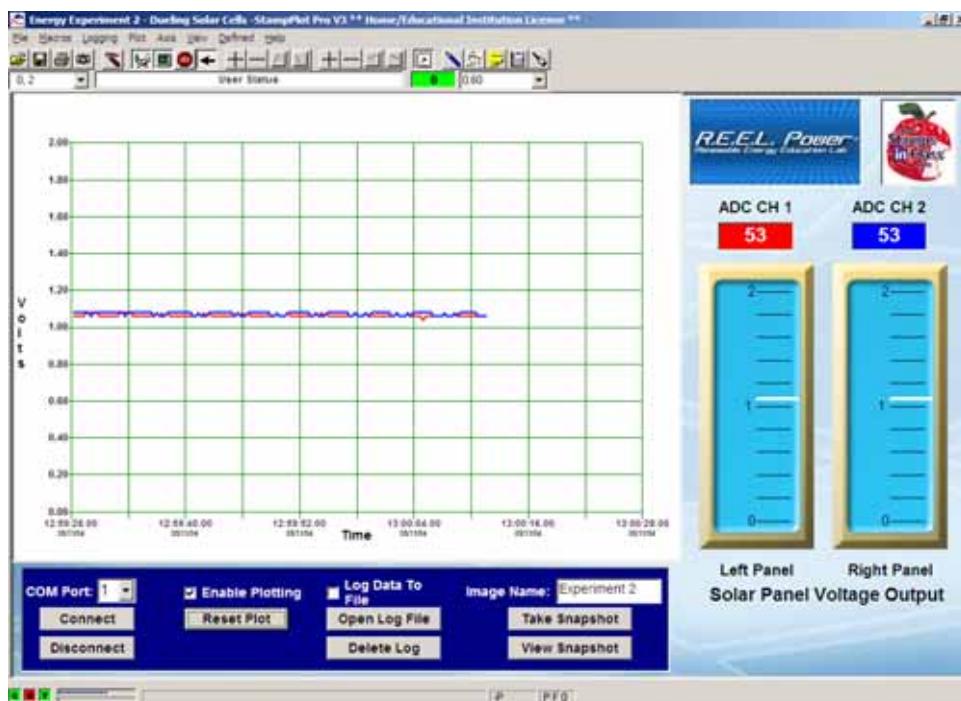


Figure 3-10: Solar Panels after offSet Voltage Adjustments Applied

As the title of this section indicates, the `offset` variable is a way to fine-tune the voltage differences between the left and right solar panels. In effect, the smaller the value for `offset`, the finer the difference between the solar panel voltage outputs. If the `offset` value is too small, the left and right LEDs may never illuminate together. If you find that this happens, you can simply add one more to the `offset` value and try again. Eventually you will come up with the correct value. However, as the value for `offset` increases, so does the “coarseness” in the tuning. In the next experiment, we will use the adjusted `ch1` and `ch2` readings based on the `offset` value you’ve computed in Experiment 3, so make it as small as possible to light both LEDs using the sun or a bright light to illuminate both left and right solar panels equally.

- ✓ If you are going to continue on now, click on the Disconnect button in StampPlot so you can run a new program in the BASIC Stamp Editor in next activity.

SUMMARY AND APPLICATIONS

In this experiment you were introduced to a modern marvel called a solar cell that produces electricity directly from sunlight. Solar cells, like the batteries you studied in Experiment 1, can be placed in series or parallel with the same voltage and current results. That is, solar cells in series add the voltages of all the cells while the total current remains at that produced by a single cell. Conversely, solar cells in parallel add the currents produced by all cells with the total voltage being equal to just a single cell. Naturally, you can combine all sorts of arrangements of series and parallel hookups to produce the voltage and current you want for your application.

These are just some ways solar cells mimic the way conventional batteries are used in day-to-day applications. However unlike even rechargeable (secondary) batteries, solar cells cannot store energy nor do they need recharging. They are either producing power from the sun, or not. This is why we started with a rechargeable battery experiment to let you understand that solar cells need a means to store the energy they collect.

On a more career-oriented note, by your choosing to perform this and the next experiment you may begin to understand the potential impact of solar cell technology on the production of electricity as we begin the 21st century. Recall that the Wright brothers first demonstrated heavier-than-air flight in 1903 –something that was deemed impossible prior to that time. In less than seventy years from that momentous flight, man successfully landed on the moon, which was also deemed an impossible task. Now here you are at the beginning of the 21st century with what appears as the ultimate in technology all around you, with nothing very significant to invent or improve upon. People felt the same way 100, 200 and more years ago, as well.

Solar cell technology has become just one example of this myopic view of the future. For years the “promise” of solar cells powering our homes and businesses never quite came true. We are still dependent on fossil fuels as our primary power source. Certain scientists and engineers still tell us that solar technology can never exceed certain technical and cost boundaries, so we are stuck with conventional fuels for the foreseeable future. Their predecessors also told the Wrights and Edison the same thing in their time about the airplane (it can never happen) and the electric light bulb (an inventor’s pipe dream). However, just like the Wright brothers and Thomas Edison who both single-handedly pioneered their chosen fields of endeavor, all it takes is one or two enterprising individuals to bring solar cell technology into the “light” where it can become the dominant power source for clean renewable energy from the sun. Maybe one of those individuals is you. We certainly hope so!

Real World Examples

Solar energy can be captured and used in basically two forms – heat and electricity – the latter being called PV or photovoltaic energy. Solar energy in the form of heat is used to heat swimming pools as well as homes. For our experiments, the focus was and is on the photovoltaic energy produced by solar cells. For example, some rural homes are powered entirely from the sun and wind. With a combination of solar and wind, or just one or the other by itself, people can now live where ever they choose with all the conveniences of modern living like telephones, television with a satellite feed, the Internet and World Wide Web and even a toaster and electric range to cook with. Even 30 years ago this was an unthinkable circumstance. Now anyone with the correct installation of solar panels, inverters and batteries can live wherever they want with all the comforts of a grid-powered home.

But you don't have to live in the wilds to experience the impact of solar power on your day to day living. These days, solar cells are quite ubiquitous - you can find them powering everything from calculators, to portable radios, to hand-held games and even cell phones for battery recharging. Larger solar panels are used to power emergency telephones, which are placed along major highways. And very large solar panel arrays are erected on building roofs. Here they are used to generate AC voltages, using a device called an inverter that contributes to powering the building's lights, air conditioners and office equipment, thus saving power (and the cost of power) from the grid.

Real World Applications

So how can you find real world applications for solar cells other than the ones just mentioned? Well how about the family car's battery? It normally gets it charging from the alternator that, in turn, is powered by the car's engine by means of a fan belt. But what if the car isn't driven for days or even weeks?



Figure 3-11
Automobile Dashboard
Solar Panel

Photo by John Gavlik

In this case the battery will run down and not provide enough starting power for the car. The solution is a solar panel that sits on the dashboard, like the one owned by the author shown in Figure 3-11. It plugs into the cigarette lighter socket, these days also called a DC adapter socket. If the car is left out in the sun during the day, the solar panel will supply enough current to keep the car's battery "topped off" so that it is ready to start the car. You can find these kinds of solar panels at auto parts stores as well as electronics hobby stores. They sell anywhere from \$19.95 to \$49.95 depending on their particular voltage and current generating capabilities.

Another application for solar panels is on sailboats. Here the solar panels can be molded into the deck of the boat with clear resin, or hung on the railings like the one in Figure 3-12, or on a roof like in Figure 3-13. This way the panel keeps the boat's batteries constantly charged and also allows the crew to walk on the deck without destroying the panel.



Figure 3-12
Sailboat with Solar Panel Mounted on Rail

Photo by John Gavlik

Sailboats generally don't run their inboard engines very often, so keeping a battery charged is of primary importance when and if the engine needs to be started. Also, even modest sailboats like to have the convenience of light at night to read with and also to power the running lights, which are the red, green and white lights required by law to be on during nighttime cruising. Solar panels are quite useful in the marine environment since "getting a jump start" is something quite inconvenient, if not impossible, depending upon where your vessel is located at sea.



Figure 3-13
Sailboat with Solar
Panel Mounted on Roof

*Photo Courtesy of
HaveBlue*

More Information

Here's where to find more information on solar cells: www.learnonline.com/ewre.html.

Chapter 4: Experiment #3, Solar Cell Sun Tracker

In Experiment 2 you constructed two similar solar panels and fine-tuned their voltage outputs to match each other. This experiment will build on Experiment #2, allowing you to track the sun with these “calibrated” panels. To do this you will physically mount the solar panel array to a servo. This system can be programmed to track the apparent east-to-west movement of the sun. You can do this in a window that has an unobstructed view of the sun for most of the day, or you can take your Board of Education and solar panels outdoors in good weather and use a 9-volt battery for power. Of course, the principle can be tested and demonstrated indoors with a desk lamp “sun,” as we have done here.

While this experiment is neat in terms of what it can do, the question to ask is “Does it have any real significance to real-world requirements for commercial sun tracking devices?” The answer is definitely YES! Many large solar arrays use electro-mechanical sun trackers to maximize the energy producing effects of sunlight on the solar panels. Rather than maintain the solar arrays in a fixed position, a sun-tracking device always keeps the panels pointed directly at the sun, thus absorbing the maximum energy.

REAL WORLD SUN-TRACKING SOLAR ARRAYS

While this is certainly a specialty area related to solar arrays, sun tracking systems do represent a significant investment in engineering and cost. The intended return of “more consistent” electrical power during daylight hours as compared with a fixed solar array must be worthwhile.

Like their solar cell counterpart, sun tracking system’s initial development dates back to the 1950s where satellites needed to keep their solar panels pointed at the sun while they orbited the earth. It was only by this means that the satellite could maintain its electronic systems in an uninterrupted manner.

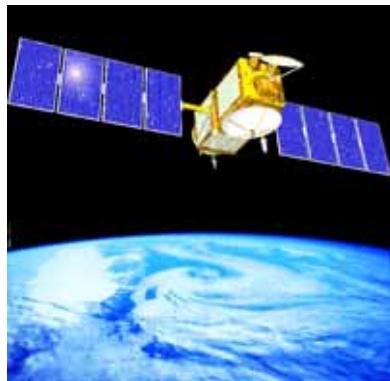


Figure 4-1:
Satellite Sun Panel Solar
Tracking Array

*Image Courtesy NASA/JPL-
Caltech*

With our country's space program fully funded at the time, little thought was given to the expense involved in developing and deploying such systems. This is still true today when it comes to purely scientific or military space-based systems, including the international space station, which continues to be built at the time of this writing.

The International Space Station's eight-part solar arrays will be 112 ft long by 39 ft wide. With all of the arrays installed, the complete Space Station will be large enough to cover a football field. Because the International Space Station (ISS) needs very high power levels, the solar arrays will require more than 250,000 silicon solar cells. The solar cells provide the ISS with 120 VDC which will generate 110 kW (kilowatts) total power, about as much as 55 houses would typically use. The solar hardware is provided by the U.S. and Russia.



Figure 4-2: The International Space Station

Photo Courtesy of NASA

Military and government spending for the development of high-risk experimental systems such as a sun tracker often pays off, both economically and scientifically, when commercial systems begin to use the same technology. Essentially, the government pays up front for the development of systems and technology that are beyond the budgets of even the most profitable commercial companies. The payback comes when commercial companies get enough technology from government research and development to continue to advance the technology on their own. Here the cycle becomes complete with commercial companies selling back their much-improved technology to the government and military. This was certainly the case in terms of government-funded programs such as the transistor, the laser and early communications satellites. And it is also the case for commercial sun trackers.



Figure 4-3: Sun Trackers - 1 kW system (Canary Islands)

Photo Courtesy of Poulek-Solar, Co. Ltd.

Many commercial companies now manufacture sun tracking systems for medium to large solar arrays, and they come in a wide variety of configurations. By adding single-axis tracking to a solar array, electrical output is increased by about 20 percent as compared to a non-tracking solar array. By moving to two-axis tracking, an increase of 10 percent over a single-axis tracker and about 30 percent over a fixed array can be achieved.

Single-Axis Tracking Systems

These have a single left-right axis on which the solar array rotates. This is the axis. This is the least complicated tracking system with a fairly simple motor and control system that turns the solar array from east to west each day. This keeps the array pointed in a close approximation to the sun.

Dual-Axis Tracking Systems

These use both left-right and up-down axes to position the solar array. Over the course of a year, the dual axis system will produce the most power, since the elevation of the sun

changes with the seasons. At the same time, the tradeoff is that it is far more expensive and complicated to design, construct and maintain.

Passive Trackers

These trackers follow the sun without having any motors to drive them. The trackers are carefully balanced. The tubes on each side of the tracker are filled with a gas. As the sun heats the gas on one side, the gas expands and flows into the other side of the tracker. This shifts the delicate balance, and the solar panels automatically tilt toward the sun.

The Zomeworks tracker has no motor. It is a passive tracker in which the shifting weight of DuPont Freon® refrigerant tips the tracker to follow the sun. The Freon moves to the cooler of two side canisters, which because of the position of the shades, causes the rack to follow the sun. This simple yet elegant and cost-effective system demonstrates that even modest solar panel installations can benefit from commercial sun trackers, as long as the increased efficiencies justify the added cost.

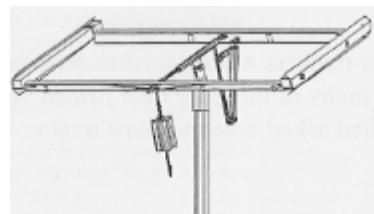


Figure 4-4:
Zomeworks Corp. Sun Tracker

Photo Courtesy of Zomeworks

Trackers with Parabolic Panels

Pictured below are two examples of sun trackers with parabolic panels that keep the sun's rays concentrated at a central point. Figure 4-5 shows the SHEC solar concentrator unit that uses a system of mirrors to concentrate the sun's energy much like a satellite dish concentrates data frequencies to a central receiver. Called the SHEC-Labs Hydrogen Generator, this device uses a proprietary catalyst, a new solar collection technology and solar tracking plus process software developed by the company to produce hydrogen

from natural gas, or by electrolyzing water at temperatures of 800 degrees Celsius (1472 degrees Fahrenheit). The solar technology shown here allows for virtually pollution-free manufacturing of hydrogen.



Figure 4-5:
SHEC-Labs Hydrogen
Generator

*Image Courtesy
SolarAccess.com*

However, SHEC isn't the only company with the same goal in mind either. The Stirling Energy Systems (SES) 25 kW solar energy concentrator dish in Figure 4-6 also uses sun tracking as a means to keep its solar array always aimed directly at the sun. So whether the answer to mass-produced hydrogen from the sun is solved by SHEC, the SES dish, or some other approach altogether, it's clear that hydrogen has become the next major frontier for energy experimentation. Sun trackers will be at the heart of any such system.



Figure 4-6:
The Stirling Energy Systems
Solar Dish

*Image Courtesy
SolarAccess.com*

Detrimental to Tracking

Tracking adds cost and mechanical fallibility. On larger commercial systems, the increased energy output more than offsets the added costs. However, on small rooftop systems the added costs typically outweigh the benefits. Fixed panel systems can be the best solution for some applications, like the one pictured below.



Figure 4-7
Fixed-panel solar
system for a traffic light.

Photo by Rich Allred

Figure 4-7 uses two solar panels, rather than using a tracking device, to power a signal light in Folsom, CA. The smaller panel faces south where the sun's energy is strongest, and the larger panel faces west. As the understanding of solar panels increases we will continue to see more and more real world applications, some simple and some complex, but each designed to meet the economic realities of the situation.

Making the Decision

The final question is usually “What is the cost of adding a sun-tracking device to my solar panel?” There are actually two costs to consider – one is the monetary cost and the other involves an energy-related cost.

The monetary cost involves the money needed to design, build and maintain the sun-tracking device. In a commercial installation these costs must be justified by a greater return on investment in the solar panel “system,” meaning the solar panel itself plus the sun-tracker. If the added cost of a sun-tracker cannot be justified by a better return on investment (ROI) as compared to a fixed solar panel installation, the sun-tracker will not be constructed.

Then there are the energy related costs. Since the sun-tracking equipment must get its power from somewhere, it makes sense to first determine if the additional power derived from the solar panels tracking the sun will actually provide MORE power than is used by the sun tracking device itself. If not, the sun-tracker will still not be considered.

These last two paragraphs point out a fundamental fact of “engineering” life. That is, if you’re working for a company that is in business to make a profit (as most are) then any project, large or small, done by the company must take into account both engineering AND economic considerations. So if you’re thinking about a technical career you should bear this in mind, since you will most likely be asked from time to time to justify why you should, or shouldn’t, be engaged in certain technical activities.

But since this experiment is “not for profit” and “all for fun and enlightenment”, let’s get going with building our servo-driven sun tracker, and “hang the costs!”

ACTIVITY #1: SHIELDING THE SOLAR PANELS

Parts Required

This experiment uses the same circuits from the previous experiments to read the solar panel voltages. A cardboard hood and a servo will be added to the solar panels, and the servo will be connected to the board. You will also need the following household (or school-room) items that are not included in the kit:

- Cardboard box or sheet of heavy paper for a hood
- Tape
- Scissors

Before mounting the solar panel to the servo it will first be necessary to add a box-like hood around the solar cells. The reason for this is to block out any sidelight that will affect the voltages being generated by both the left and right solar panels. In effect, the hood will act to “tunnel” the light coming directly from the sun onto the solar panels. This will make tracking the sun a lot more precise, since the hood greatly reduces reflected side lighting. You can use any convenient material like cardboard or rigid construction paper to make a hood. We used corrugated cardboard for our example.

- ✓ Construct a simple box out of heavy paper or cardboard by bending a strip at least 4 inches wide (the longer the hood, the better the light tunneling) around all four edges of the solar panel tray and taping it in place.

- ✓ Make sure to surround the tray on all sides and make a notch for the triangular protruding section.
- ✓ Take some time to do this because you will want the hood to remain in place for several days, or longer, as you experiment with the sun tracker. Refer to Figure 4-8 for some ideas.
- ✓ Don't mount the servo to the solar panel just yet; we'll get to that in a bit.

4



Figure 4-8: Adding A Hood

With the hood in place it's now time to test it effects using StampPlot.

- ✓ If you disconnected your solar panels from your circuit to tape on the hood (which you probably did) reconnect the leads now, referring back to Figure 3-4 on page 91.
- ✓ In the BASIC Stamp Editor, run DuelingSolarCells.bs2.
- ✓ Make note of the COM port in use, and then close the Debug Terminal.
- ✓ Open StampPlot via Start → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_3.spm. (If StampPlot is already open, you may

- select Macros from the toolbar, then Select Start-Up macro, and open sic_ewre_exp_3.spm from the SIC_Energy folder.)
- ✓ Hold the hooded panel in your hand and point it at the sun or a bright light.
 - ✓ Move the panel back and forth across the light source, then hold it underneath the light source and tilt the tray from side to side.

Notice how the voltages peak then fall off when the solar panels are not pointing directly at the light source, visible in the first two peaks on left side of the plot in Figure 4-9. Following that, the two channels show greater separation when tilting the solar panels from side to side. This voltage difference between the two channels will be used by the sun tracker program to direct a servo motor to turn the panels toward the light source.

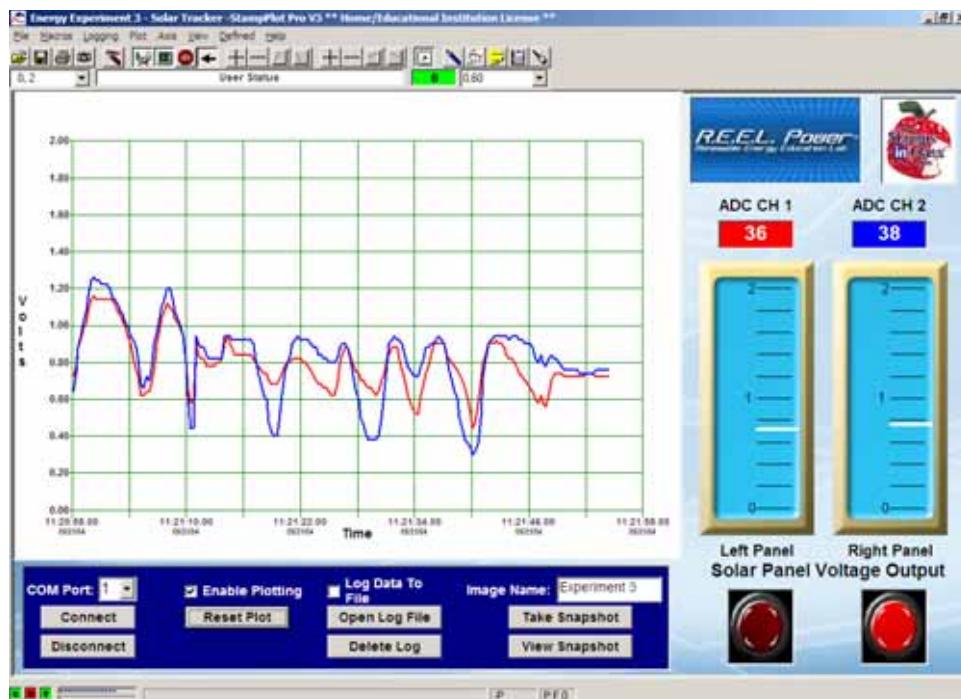


Figure 4-9: Voltage Outputs From Hooded Solar Panels

Your results will vary depending on the color, transparency and reflective qualities of the material you use for the hood. Indeed, experimenting with different materials could be

another avenue of exploration. Nevertheless, try to design your hood to be as deep as possible to help make the sun tracking experiment produce quickly-changing voltage peaks and valleys, which demonstrate the effectiveness of the hood's tunneling effect. You should also view the left and right LEDs on your circuit to confirm that they both illuminate ONLY when the hooded solar panels are pointed directly at the light source.

- ✓ If you are going to continue on now, click on the Disconnect button in StampPlot so you can run a new program in the BASIC Stamp Editor.

ACTIVITY #2: CENTERING THE SERVO

Before we can use the Parallax Continuous Rotation Servo, it must be “centered.” This servo motor is designed to hold a steady position when receiving 1.5 ms pulses, and to rotate when it receives pulses less than or greater than this “center” value. Our Sun Tracker program design makes use of this characteristic, but the servo must first be manually calibrated to make sure it does hold steady when receiving the 1.5 ms pulse.

Parts Required

(1) Parallax Continuous Rotation servo

Parallax Screwdriver

If you are using a Board of Education Rev B or A, you will also need:

- (1) 3-pin header
- (3) jumper wires

- ✓ Disconnect power to your Board of Education.
- ✓ If you are using the Board of Education Rev C, refer to Figure 4-10. Set the jumper between the servo ports to Vdd, then plug the servo into the P12 Servo Port with the red and black leads lined up with the labels on the board.

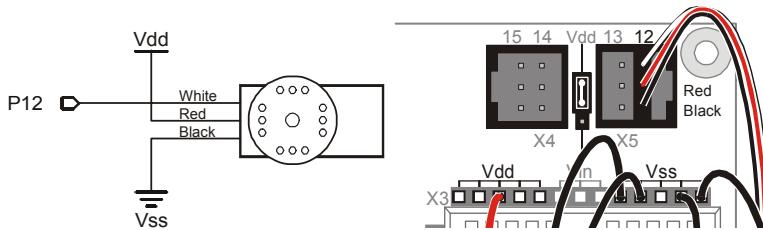


Figure 4-10
Board of
Education Rev C
Connection for
Centering the
Servo

- ✓ If you are using the Board of Education Rev B or A refer to Figure 4-11. Build the servo connection on the breadboard, using the 3-pin header to connect the servo to the board. Make sure that the servo's black lead connects to Vss, the red lead connects to Vdd, and the white lead connects to P12. Note: do not use the servo ports on the Board of Education Rev B for this activity.

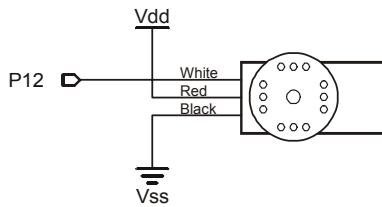
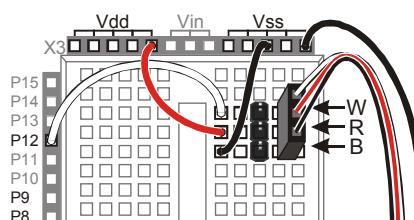


Figure 4-11
Board of Education Rev B
or Rev A Connection for
Centering the Servo



Note: Do not use the Board of Education Rev B servo ports for this activity.

- ✓ Double-check your servo connection, and jumper if necessary. If you plug the servo in backwards, or connect it to Vin, and reconnect the power, you will damage your servo.
- ✓ When you are sure your connections are correct, reconnect power to your board. (If you are using a Board of Education Rev C, the 3-position switch must be in position-2 to power the servo ports.)
- ✓ Enter and run the program CenteringServo.bs2.

```
' Experiments with Renewable Energy v1.0 - CenteringServo.bs2
' Delivers a 1.5 ms pulse to P12 so servo can be centered.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG    "Sending 1.5 ms pulse to servo on P12"

DO
PULSOUT 12, 750
PAUSE 20
LOOP
```

If your servo is new and has not been centered before, it is probably rotating. If it is not moving, it is probably already centered. Either way, follow the steps below.

- ✓ Very carefully, insert the tip of the Parallax screwdriver into the potentiometer access hole in the servo case, as shown in Figure 4-12. Don't apply too much pressure or you will damage the servo.
- ✓ Gently rotate the screwdriver a little bit, first one way, then the other. You should see that the servo speeds up when you turn the screwdriver one way, and slows down or even reverses the other way.
- ✓ Gently twist the screwdriver in tiny increments until the servo holds still, indicating that it is exactly centered.
- ✓ Disconnect power to your board.
- ✓ Unplug the servo from the Board of Education before continuing on.



Figure 4-12
Centering the Servo

Gently insert the screwdriver tip into the access hole, and slowly twist it until the servo stops rotating. Do not apply too much pressure to prevent damage to the servo.



Learn more about the **PULSOUT** command and Parallax Continuous Rotation servo control in *Robotics with the Boe-Bot*, another Stamps in Class title from Parallax, Inc. This text can be downloaded free from the Educational Downloads page on the Education menu at www.parallax.com.

ACTIVITY #3: ADDING THE SERVO TO THE SOLAR PANELS

Parts Required

- (1) Parallax Continuous Rotation servo
- (1) 1 ½" hexagonal standoff, pre-drilled
- (1) 1/4" panhead Phillips screw
- (1) #4 lock washer
- (2) #4 flat washer
- (2) 7/8" panhead Phillips screws
- (2) #4 hex nuts

Tools Required

Parallax screwdriver

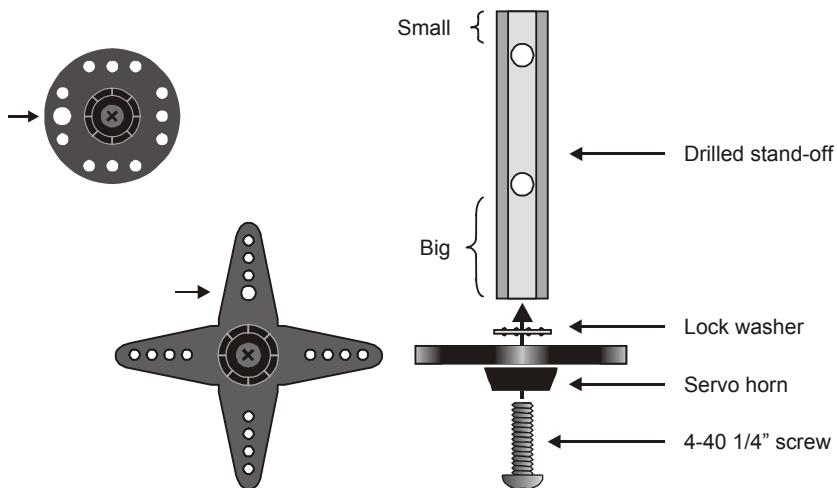
Hand drill with a #36 bit -OR-

Hobby knife (such as an X-acto® knife) with a slim, sharp tip

Optional: Diagonal cutters or other flush clippers (such as toenail clippers!)

Below are the directions for mounting the servo on the solar panel tray, as shown in Figure 4-13 and Figure 4-14.

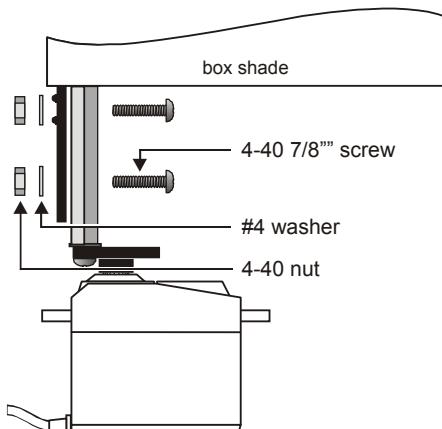
- ✓ Unscrew the servo horn from the servo, being careful to keep track of the tiny black screw.
- ✓ Using the hand drill or the tip of a hobby knife, enlarge one of the holes in the servo horn closest to the center, making it large enough to accommodate the 1/4" panhead screw. (Your servo may have a round horn or a 4-prong horn.)
- ✓ Thread the screw through the enlarged hole in the servo horn from the underside.
- ✓ Slip the lock washer onto the screw.
- ✓ Thread the standoff onto the screw, using the end farther from the drilled holes.
- ✓ Find that tiny black screw, and reattach the servo horn to the servo shaft.
- ✓ Line up the holes drilled in the standoff with the holes in the solar panel tray tab, and thread the ½" screws through the holes, as shown in Figure 4-14.
- ✓ Thread washers and hex nuts on the screws and gently tighten. If you find that tiny nubs of plastic on the tray tab interfere, you may clip them off with the diagonal cutters (or toenail clippers!)



4

Figure 4-13: Mounting the Standoff on the Servo Horn

Note that the 4-40 1/4" screw is going through the enlarged hole in one side of the servo horn, and not through the center of the horn itself.

**Figure 4-14**
Attaching the Servo to the
Solar Panel Tray

Adding the Servo Circuit

- ✓ Make sure you have disconnected the power to your Board of Education (as you should always do before modifying circuits!)
- ✓ Add the servo to your Dueling Solar Cells circuit as shown by the schematic in Figure 4-15.

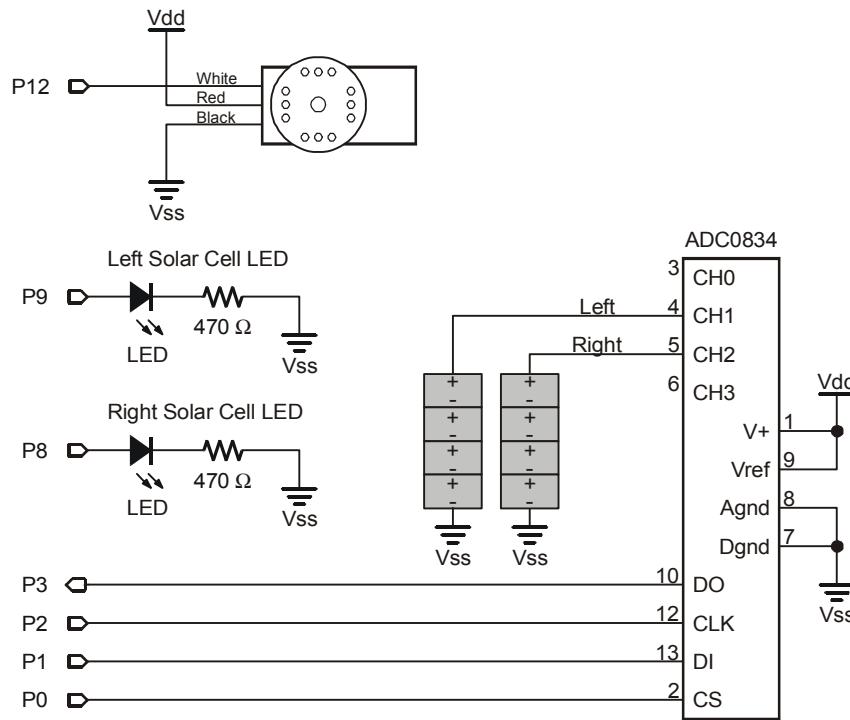


Figure 4-15: Sun Tracker Schematic

- ✓ For a Board of Education Rev C, refer to the wiring diagram in Figure 4-16 to make sure the servo leads are lined up with the labels by the P12 servo port, and that your jumper is set to Vdd.

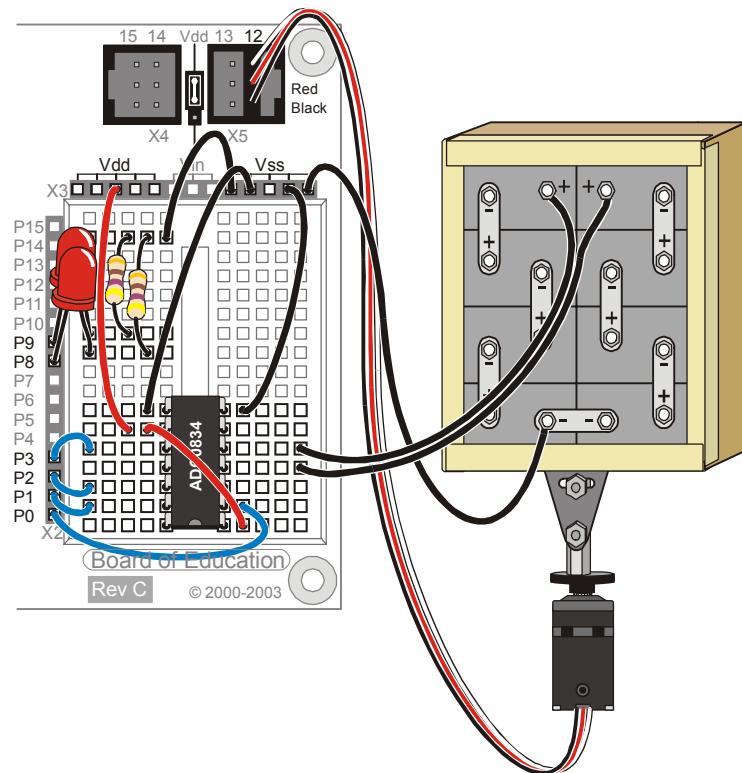


Figure 4-16: Sun Tracker Wiring Diagram for Board of Education Rev C

- ✓ For a Board of Education Rev B or A, refer to Figure 4-17 to make sure your servo's black lead is connected to Vss, the red lead to Vdd, and the white lead to P12, via the 3-[in header on the breadboard.

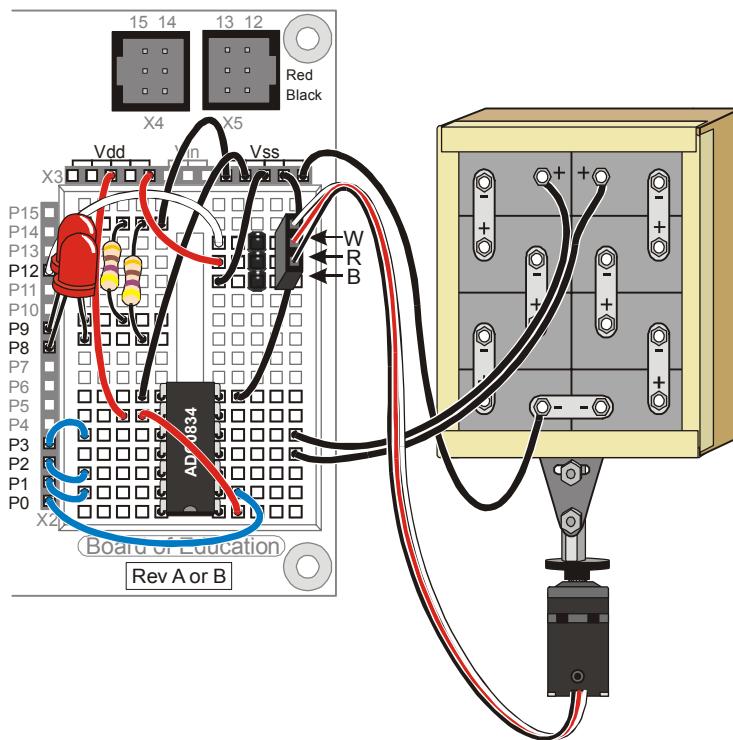


Figure 4-17: Sun Tracker Wiring Diagram for Board of Education Rev B or A

Note: Do not use the servo headers in the Board of Education Rev B for this activity.

- ✓ After you have made sure your servo is plugged in properly, reconnect power to your board. (If you are using the Board of Education Rev C, you will need to have your 3-position switch in position-2 to power the servo.)

ACTIVITY #4: PROGRAMMING THE SUN TRACKER

Once again, we will be building upon our previous program. These additions let us read and compare the voltages generated by the two solar arrays. Remember, you may either

follow the directions to build on your program for each experiment, or download the programs from www.parallax.com.

- ✓ In the BASIC Stamp Editor, open DuelingSolarCells.bs2.
- ✓ Rename and save the program as SunTracker.bs2.
- ✓ Update the title to read as follows:

```
' Experiments with Renewable Energy v1.0 - SunTracker.bs2
' Tracks the sun with 2 solar panels attached to a servo motor.
```

- ✓ Now add the following code to the end of the Declarations section:

```
' ----- For Experiment 3: Solar Cell Sun Tracker -----
Servo      PIN      12          ' Sun tracker servo on P12
waitCount  VAR      Nib         ' Dampens servo's "hunting"
WaitVal    CON      1          ' initial value of waitCount
```

Now we'll follow up with adding the code for Experiment 3 in the Subroutines section as shown below:

- ✓ First add the commented lines to document the subroutine:

```
' -----
' Experiment 3: Solar Cell Sun Tracker
' -----
'
' Based on the values for CH1 and CH2 from Experiment 2
' Determine if the servo should stop, move left or move right
'
' By moving the servo left or right, the individual solar cell voltages
' will eventually become equal and the program will cause the servo to stop.
' That is, until the sun moves and the voltages become unequal again thus
' causing the servo to reposition the solar panels.
'
' Return
' -----
```

- ✓ Then fill out the **Exp_3** subroutine as shown:

```
Exp_3:
  IF (ch1 = ch2) THEN Exp_3_Stop
  IF (ch1 < ch2) THEN Exp_3_Move_Left
  IF (ch1 > ch2) THEN Exp_3_Move_Right
```

```

Exp_3_Stop:
    PULSOUT Servo, 750
    GOTO Exp_3_Set_Wait_Count

Exp_3_Move_Left:
    waitCount = waitCount - 1
    IF (waitCount <> 0) THEN
        GOTO Exp_3_End
    ELSE
        PULSOUT Servo, 755
        GOTO Exp_3_Set_Wait_Count
    ENDIF

Exp_3_Move_Right:
    waitCount = waitCount - 1
    IF (waitCount <> 0) THEN
        GOTO Exp_3_End
    ELSE
        PULSOUT Servo, 745
        GOTO Exp_3_Set_Wait_Count
    ENDIF

Exp_3_Set_Wait_Count:
    waitCount = WaitVal

Exp_3_End:
    RETURN

```

- ✓ Save your work!

Your Turn: Tracking the Sun

- ✓ In the BASIC Stamp Editor, run SunTracker.bs2.
- ✓ Note the COM port being used, then close the BASIC Stamp Editor's Debug Terminal.
- ✓ Open StampPlot Pro by clicking on Start → Programs → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_3.spm.
- ✓ Set the COM port in StampPlot to the same one that was being used by the Debug Terminal.
- ✓ Click on the Connect button, and make sure the Enable Plotting box is checked.
- ✓ Hold the tracker by the servo so that it is pointing towards your light source, but not directly at it. The servo should begin moving the solar panels, little by little, towards the light.
- ✓ While this is happening observe the left and right LEDs on your circuit.

If one LED is illuminated and the other is extinguished, the servo should be moving the solar panels in the direction of the extinguished LED. The servo movement should stop momentarily when both LEDs become illuminated. The servo may now begin to “hunt” or move back and forth in an effort to equalize the left and right solar panel voltages.

- ✓ At the same time observe the StampPlot display.

As illustrated in Figure 4-18, the voltages of the left and right solar panels are not equal at first. Then as the servo moves the panels towards the light, the voltages begin to converge culminating in two lines overlaying one another. As we are using a 60 W desk lamp as our “sun” you can see the familiar wave pattern to each trace.

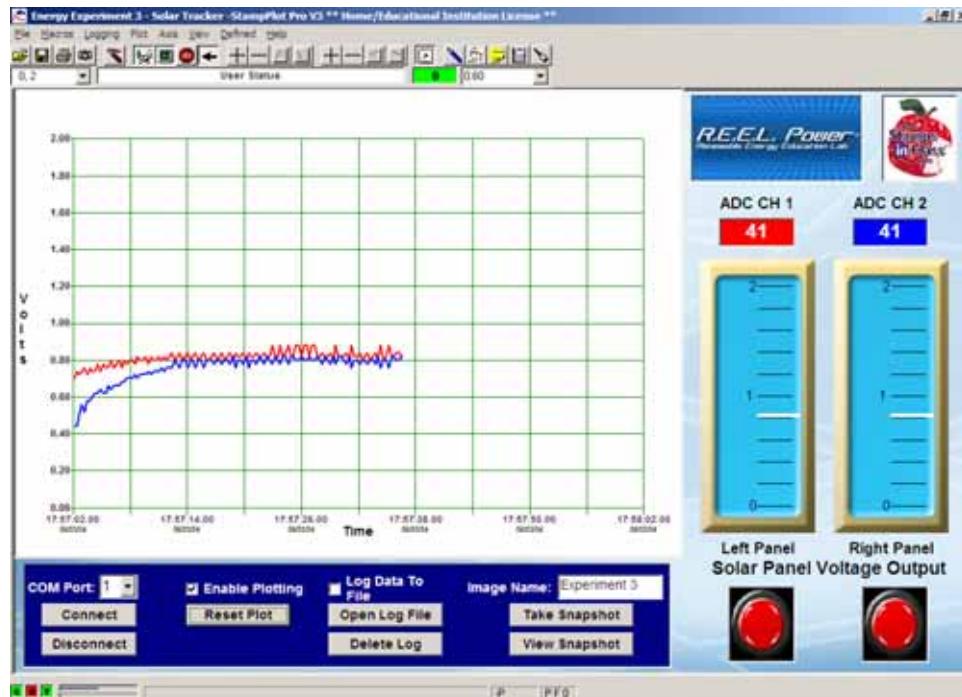


Figure 4-18: Sun Tracker in Tracking Mode

Just for fun, move your sun tracker or your lamp to simulate the changing relative positions that would occur in a system working out of doors with the real sun. Or, set up

your experiment outdoors (in dry weather!) or in a window that gets all-day sun as would a south-facing window in the Northern Hemisphere.

ACTIVITY #5: HOW THE SUN TRACKER WORKS

The sun tracker works by having the servo move the solar panels in a direction that will equalize both of their voltages. This translates into the servo continuously pointing both solar panels directly at the light source, thus tracking it if it moves. It makes no difference what voltages the solar panels are outputting at the time; the only thing that does matter is that the voltages be the same. Experiment 2 showed that this occurs when the left and right panels are pointing directly at the sun or other bright light source AND when the **offset** variable is set to calibrate the individual panels. With the correct **offset** value in the program, both panels will deliver an equally compensated voltage when pointed directly at the light source.

To make the servo track the sun we begin with our declared variables and constants as shown below:

Servo	PIN	12
waitCount	VAR	Nib
WaitVal	CON	1

The **Servo PIN 12** instruction allows us to easily control the servo through the BASIC Stamp I/O P12. The **waitCount** variable, a 4-bit nibble, is used to “dampen” the servo’s hunting while **waitVal** is the initial value of **waitCount** set by the program.

```
Exp_3 :
  IF (ch1 = ch2) THEN Exp_3_Stop
  IF (ch1 < ch2) THEN Exp_3_Move_Left
  IF (ch1 > ch2) THEN Exp_3_Move_Right
```

The first three **IF...THEN** commands evaluate which solar panel is receiving more light, and directs the program to the appropriate label. If the first condition (**ch1 = ch2**) is true, the solar panels are generating equal voltage, and the tracker is probably pointing right at the light source, or close to it. The servo does not need to turn and seek the light, so **THEN** directs the program to the **Exp_3_Stop** label. If the condition is not true, the program falls to the next **IF...THEN** statement and tests again; one of the two statements will be true, and the program will go to the corresponding label. If the left panel’s voltage is less than the right panel’s voltage, the program branches to **Exp_3_Move_Left**.

Finally, if the left panel's voltage is greater than the right panel, the program branches to **Exp_3_Move_Right**. Following these basic branch instructions, the remaining part of the code does the actual tracking work.

```
Exp_3_Stop:
    PULSOUT Servo, 750
    GOTO Exp_3_Set_Wait_Count
```

At **Exp_3_Stop** the program executes a **PULSOUT** command to the servo for 750 two-microsecond units, or 1.5 milliseconds. This value should stop the servo. If it doesn't and the servo still "creeps" a bit to the left or right, you can point the solar panels directly at your light source, then adjust the servo by inserting your Parallax screwdriver into the adjustment port and very gently twisting it until the servo is still when both LEDs are illuminated.

```
Exp_3_Move_Left:
    waitCount = waitCount - 1
    IF (waitCount <> 0) THEN
        GOTO Exp_3_End
    ELSE
        PULSOUT Servo, 755
        GOTO Exp_3_Set_Wait_Count
    ENDIF

Exp_3_Move_Right:
    waitCount = waitCount - 1
    IF (waitCount <> 0) THEN
        GOTO Exp_3_End
    ELSE
        PULSOUT Servo, 745
        GOTO Exp_3_Set_Wait_Count
    ENDIF
```

Continuing, the next instruction branches to **Exp_3_Set_Wait_Count** where the **waitCount** variable is set to the **waitVal** that has already been declared. We'll get to what **waitCount** and **waitVal** are all about in a moment.

The **Exp_3_Move_Left** and **Exp_3_Move_Right** subroutines are basically the same except for the value of the **PULSOUT** command. To move the servo slightly left the program adds 5 to the neutral 750 "stop" value, while the program subtracts 5 to 750 to make the servo move right. These 745 and 755 values do not represent anything like

moving the servo one degree at a time (which is what it may look like). Instead, they were chosen to move the servo with enough precision so that the voltages from the left and right panels would converge without overshooting one another. Later on you will be asked to experiment with these values to fine-tune your sun tracker to your particular liking.

Another thing that both subroutines do is decrement `waitCount` with the command `waitCount = waitCount - 1`. After decrementing, if `waitCount` is 0, the servo is moved in the indicated direction. However if `waitCount` is not zero, the subroutine simply exits and does not move the servo.

Since our program declared that `waitVal` is equal to 1, `waitCount` will always be decremented to 0 and the servo will always be moved left or right a bit. This, then, causes the “hunting” that we referred to earlier. That is, the servo moves left or right every time the solar panel voltages are not equal. To mitigate the hunting action we incorporated the `waitCount` and `waitVal` into the program so that the program could “dampen” the servo hunting. In other words, if `waitVal` is set to 3 instead of 1 in the declaration section, the program will ignore any voltage imbalance for three consecutive passes before moving the servo. Let’s try it:

Your Turn – Adjusting the `waitVal` Variable

- ✓ First, disable StampPlot by deselecting the Connect icon and return to the BASIC Stamp Editor.
- ✓ Locate the `waitVal` constant in the Experiment 3 declarations and change it from 1 to 3 so it now reads:

```
' ----- For Experiment 3: Solar Cell Sun Tracker -----
Servo      PIN      12                      ' Sun tracker servo on P12
waitCount  VAR      Nib                     ' Dampens servo's "hunting"
WaitVal    CON      1                       ' initial value of waitCount
```

- ✓ Next, save the modified code and reload it into the BS2 by clicking on the Run icon in the BASIC Stamp Editor or type Ctrl-R.

You should now see the “dampening” effects of the new `waitVal` constant on the servo’s actions.

- ✓ Watch the left and right LEDs and notice that when both are not illuminated, the servo does not move every time.

Instead, the subroutines are decrementing `waitCount`. Another thing happens to `waitCount` when both LEDs become illuminated, even briefly. Notice that the second instruction after the label `Exp_3_Stop` causes the program to branch to `Exp_3_Set_Wait_Count`. Here the `waitCount` variable is re-initialized to 3, the new `waitVal` constant. So even after the `Exp_3_Move_Left` or `Exp_3_Move_Right` subroutines decrement `waitCount`, if the two solar panels suddenly become equal in voltage WITHOUT moving the solar panels, the `waitCount` value is re-initialized.

4

The effect of all of this is to not overreact to slight, random imbalances in the solar panel voltage outputs. And this is what's called "damping" because these random imbalances are taken out of the servo action thus making the servo tracking much smoother. If the solar panels do, in fact, remain at different voltage values after `waitCount` is decremented to 0, the servo will step in and move the panel in the correct direction.

A Typical Problem

In your experimental trials with the sun tracker you may notice that while it follows the sun or a bright light source, it may not point directly at the light. This goes back to Experiment 2 when you calibrated both panels using the `offset` value. Perhaps this value is not correct for the light source that you're trying to track in this experiment, or maybe you rushed through this part before. Nevertheless, in order to recalibrate `offset` value:

- ✓ First turn off the power to your Board of Education.
- ✓ Unplug the servo.
- ✓ Reconnect the power to the Board of Education.
- ✓ Rotate the hooded panels by hand so that they point DIRECTLY at the light source that you are attempting to track. Make sure that the light falls evenly on ALL eight solar cells; i.e., no shadows or reflections from the sides of the hood.
- ✓ Next, go back and repeat the procedures in Experiment 2 for adjusting the `offset` variable under the heading Your Turn: Fine-Tuning the `offSet` Variable on page 101.
- ✓ Following this, reconnect the servo to your circuit and resume your sun tracking.

SUMMARY AND APPLICATIONS

Experimenting with a sun tracker is both fun and rewarding. Here you learned the basics of how a servo works to control the back and forth motion of the solar panels. Indirectly, however, you also learned about a “closed-loop” control system.

What is a “closed loop” control system? A closed-loop control system provides automatic adjustment of a process by collecting and evaluating data and responding to it accordingly, with the aim of maintaining the desired output under changing conditions.



What is an “open loop” control system? An open-loop control system lacks the mechanism for self-adjustment, therefore, it can only deliver a desired output if the process is well understood and all conditions affecting the process are constant.

Read more about open-loop and closed-loop systems in *Industrial Control* by Martin Hebel, another popular title in the Parallax Stamps in Class curriculum.

In order to better illustrate the differences, think back to Experiment 2 where you balanced the solar panel voltage differences with the `offset` variable. You used StampPlot to adjust out any differences between the left and right panels. By viewing the StampPlot plots as well as the ADC values above the voltage gauges, you used your eyes and brain to calculate a value for `offset`. Then you applied the value to the program and then retested the voltage outputs from both the left and right solar panels, again using StampPlot. While effective, this is still an example of an “open-loop” system. That is, once you “manually” adjusted the voltage `offset` value; the solar panels were on their own in terms of generating possibly different voltage offsets due to a different light source including the effects of reflections. In effect, there is no constant “feedback” to readjust the `offset` value. Therefore, the system used in Experiment 2 is of an open-loop type.

On the other hand, by using a servo to track the sun you are using a closed-loop system. What’s closed about it? Well, the servo is constantly “feeding back” positional information to keep the solar panel voltages in balance. This constant feedback is the defining element in all closed-loop systems and provides the positional “evidence” you need for your sun tracking activities. So by providing feedback to the system, you have “closed the loop” in terms of making corrections for the solar panel’s voltage imbalance when the sun or light source moves.

There may not be too many applications for hobby-type sun trackers like the one you just built, but there are certainly thousands, if not millions, of applications for open and

closed loop systems. As a matter of fact the math and physics behind both types of systems are to be found in college courses that go by the name Control Theory or a variation of it. Control Theory is a fundamental course for all mechanical, electrical and electronic engineering studies and teaches how to design systems for optimum performance in an application. Sometimes an open-loop system is fine while other times a closed-loop system is necessary. Doing Experiments 2 and 3 have informally introduced you to both.

Getting The Most From Your Sun Tracker

Solar panels work best when they are pointed perpendicular to the sun, which is when they capture the sun's maximum energy. Not only does the solar panel need to perpendicularly track the sun in an east-west direction during the day, it should also track the sun at the correct north-south angle during the year. The sun tracker you built and programmed in Experiment 3 can be adjusted to acquire between 10% to 40% more power if you know how to keep it correctly pointed at the sun not only over the course of a day, but also as the seasons change throughout the year.

You may have already noticed that when you adjusted the solar panels so that they are perpendicular to the sun at "solar noon", they are not even close to being perpendicular in the morning or afternoon. Solar noon is when the sun is at its highest point in the sky, which is somewhere between 11:30 am and 12:30 pm local time. This daily east to west solar motion is called the "solar azimuth". During the year the sun's apparent height in the sky also changes from summer to winter. This yearly north to south motion is called the "solar declination". Astronomers and sailors use azimuth and declination to plot the position of the sun during the day and the stars at night. And you can apply these same simple mathematical principles to keep your sun tracker "on track" to get the most power from the sun.

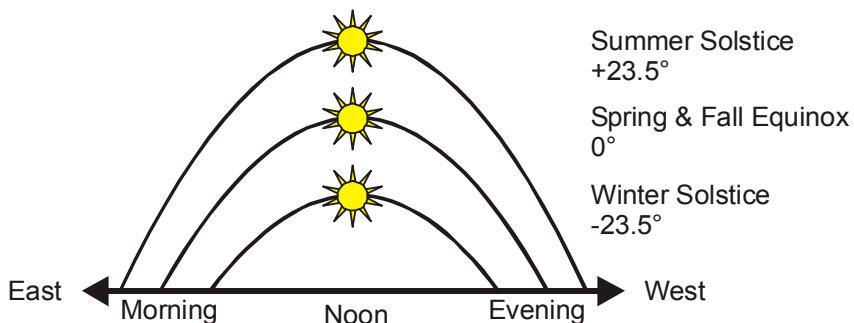


Figure 4-19: Solar Azimuth and Declination

As Figure 4-19 illustrates, the solar noon varies in apparent height above the horizon depending upon the season of the year. Once again, this apparent height above the horizon is called “declination” and is measured as an angle from the horizon to the solar noon point. Because of the Earth’s tilt, the declination is greatest at the Summer Solstice on June 22 (+23.5 degrees) and the least at the Winter Solstice on December 22 (-23.5 degrees). In between these times come the Spring and Fall Equinox when the declination is zero degrees. This is when the sun is directly over the Equator.

But what does all this mean to you? In other words, how can you adjust your sun tracker for the best declination at any time of the year? The answer lies in the application of a little algebra. Using the following formulas developed by Richard Perez and Sam Coleman, appearing here courtesy of *Home Power* magazine, you can compute the correct declination angle for any day of the year.

$$D = 23.50 * \sin ((T / 365.25) * 3600)$$

Where T is the number of days to the current date measured from the Spring Equinox of March 21.

Sound a little confusing? It’s not when you consider that the declination angle is zero at both the Spring and Fall Equinox. Therefore, all you need to do is count the number of days from March 21 and plug it into the formula. For example, let’s assume four dates that will show you what we mean.

For March 21 (Spring Equinox)

```
D = 23.5° * sin ((T / 365.25) * 360°)
D = 23.5° * sin ((0 / 365.25) * 360°)
D = 23.5° * sin ((0) * 360°)
D = 23.5° * sin (0°)
D = 23.5° * 0
D = 0°
```

For June 21 (Summer Solstice)

```
D = 23.5° * sin ((T / 365.25) * 360°)
D = 23.5° * sin ((91 / 365.25) * 360°)
D = 23.5° * sin ((.25) * 360°)
D = 23.5° * sin (90°)
D = 23.5° * 1
D = 23.5°
```

For September 23 (Fall Equinox)

```
D = 23.5° * sin ((T / 365.25) * 360°)
D = 23.5° * sin ((183 / 365.25) * 360°)
D = 23.5° * sin ((0.50) * 360°)
D = 23.5° * sin (180°)
D = 23.5° * 0
D = 0°
```

For December 22 (Winter Solstice)

```
D = 23.5° * sin ((T / 365.25) * 360°)
D = 23.5° * sin ((274 / 365.25) * 360°)
D = 23.5° * sin ((0.75) * 360°)
D = 23.5° * sin (270°)
D = 23.5° * -1
D = - (23.5°)
```

As you can see our formula matches exactly with the cardinal points on the progression of the Earth as it travels around the sun. However, we're not done yet in terms of figuring out the angle to point our sun tracker during the year. The formula for this is even simpler.

$$A = L + D$$

Where A is the angle of the solar panel and L is your latitude in degrees.

You can find your latitude by going to a world globe and move up or down, above or below the Equator depending on your particular location in the Northern or Southern Hemisphere, respectively. Either way, plug in your latitude to determine the best angles for the times of the year just computed. Your author's home latitude in Southern California is approximately 34.5 degrees North (that's North of the Equator), so let's compute the following sun tracker angles for this position:

For March 21 and September 23 (Spring and Fall Equinox)

$$\begin{aligned}A &= L + D \\A &= 34.5^\circ + 0^\circ \\A &= 34.5^\circ\end{aligned}$$

For June 22 (Summer Solstice)

$$\begin{aligned}A &= L + D \\A &= 34.5^\circ + 23.50^\circ \\A &= 58.0^\circ\end{aligned}$$

For December 22 (Winter Solstice)

$$\begin{aligned}A &= L + D \\A &= 34.5^\circ + (-23.5^\circ) \\A &= 11.0^\circ\end{aligned}$$

What these results imply is that from summer to winter there is a 47-degree change in the sun's declination regardless of your particular latitude. In any case we hope that this helps you to determine the correct "up and down" angle for your sun tracker during any part of the year, so that you can get the best power output that the sun can supply. Remember that for best results you should have a clear, unobstructed view of the sun from east to west between morning and afternoon.

More Information

Here's where to find more information on sun trackers: www.learnonline.com/ewre.html

Chapter 5: Experiment #4, Half and Full Wave Rectification

5

This experiment in half and full wave rectification begins your study of AC or alternating current energy principles. Alternating current, as opposed to direct current, which you studied in the first three experiments, is used primarily for the transmission of high voltage electrical power over long distances. In addition, low voltage alternating current is used to directly power many common household devices such as light bulbs and motors. While your television and computer are also powered by conventional 110 volt, 60-cycle AC house current, they both need a device to convert the AC voltage to DC. This device is called a power supply, and all power supplies use a principal called “rectification” to convert AC to DC, which is what this experiment is all about.

Rectification = Modifying a signal so that either the positive or the negative portions of the signal are eliminated. There are two types of rectification, half-wave and full-wave.

Half-wave rectification = Only the positive (or negative) parts of a signal remain, the other portions having been truncated:

Full-wave rectification = Half of the signal is inverted, so the entire signal is either positive (or negative).



Unrectified Signal

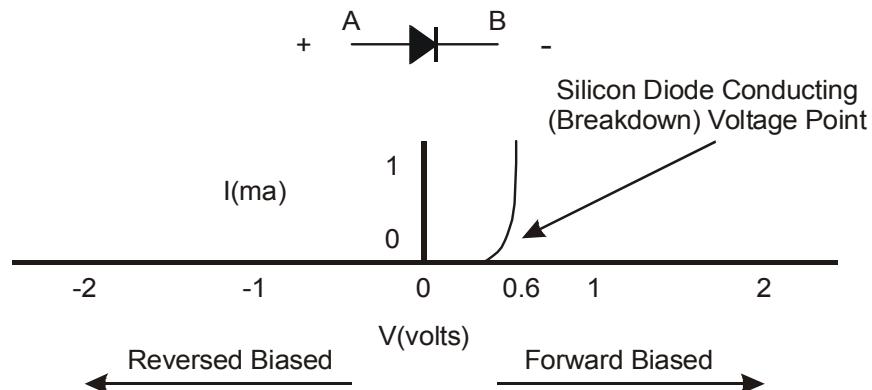


Half-wave rectification



Full-wave rectification

In this context the word rectification means, “to change”; that is, to change a signal from alternating current (AC) to direct current (DC). The two basic forms of rectification, half wave and full wave, both use a diode, like the one we used in our Battery Charger circuit, to begin the conversion from AC to DC. You can think of a diode as a kind of one-way valve that allows current to flow in only one direction, from positive to negative. Engineers call this “forward-biasing” the diode. There are a wide variety of diodes ranging from small signal diodes used in transistor radios to power diodes that can pass huge amounts of current as is needed for medium to large power supplies. Regardless of the type, every diode has a positive anode and negative cathode, like the diode pictured in Figure 5-1; current flows from the positive anode “A” to the negative cathode “B”.

**Figure 5-1:** Diode Diagram and Signal Characteristics

And because most small signal diodes, like the ones we use in this text, are made from silicon, a certain minimum voltage must be applied before the diode “conducts” or begins to pass current from anode to cathode. In the case of a silicon diode this minimum “breakdown” voltage is between 0.6 to 0.7 volts. Diodes made from germanium, another semiconductor material, have a lower breakdown or conducting voltage of around 0.4 volts. Both types are used in electronic circuits, and the choice of one over the other is usually a decision based on either lower-cost (silicon) or lower-conducting voltage (germanium); a lower-conducting voltage may be necessary for the other parts of the electronic circuit to function correctly thus justifying the extra cost. Generally, germanium diodes are more expensive as compared with silicon diodes; however both have their place in modern electronics. You are using silicon diodes for the experiments in this text.

If you’re familiar with power supplies as a product, you would think by now that every possible power supply has already been developed and is on the market for sale. That may be true up to a point; however industry is constantly demanding more efficiency (power) from power supplies for less money spent. Remember, you can find a power supply in every computer, computer monitor, television and stereo that uses regular house current for its primary source of input power. Because of the huge volumes that are sold, this “economy of scale” serves to drive the cost of power supplies down lower and lower each year. At the same time, engineers are hard at work to make power supplies even more efficient and inexpensive. So in spite of current-day technologies being so advanced as viewed in the context of the present, much more is left to be done in terms of

electronic design and cost reductions in the very near future. Therefore, the power electronic industry has an ongoing demand for clever power supply designers. Perhaps this experiment in half and full wave rectification will inspire a career in the power electronics industry for you. We hope so, because such a career is both exciting and rewarding.

ACTIVITY #1: THE PRINCIPLES OF HALF WAVE RECTIFICATION

Half wave rectification is the simpler of the two basic forms of rectification; however it is also at best half as efficient as compared to full wave rectification. A half-wave rectifier circuit generally consists of two passive components, a diode and a resistive load, plus a source of alternating current as illustrated in Figure 5-2. Notice that the DC output across the load resistor consists of only the positive part of the AC input. This is because the diode only conducts the positive portion of the AC wave from A to B. In addition, only the portion of the AC positive wave above 0.6 volts gets through the diode due to its inherent voltage breakdown condition. Again, this is called the forward-biased condition.

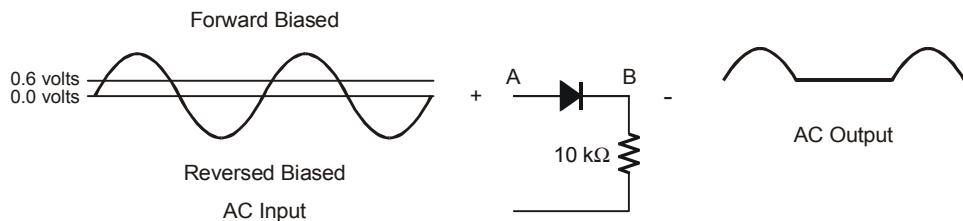


Figure 5-2: Example Half Wave Rectifier Circuit

During the negative portion of the AC input the diode is said to be “reversed-biased” since current is trying to flow in the opposite direction from B to A. However, no current can flow due to the diode’s one-way valve operation, so the resulting DC output across the load resistor is zero during the negative portion of the wave. Therefore in a half wave rectifier circuit, only the positive portion of the AC input above the 0.6-volt silicon diode breakdown voltage gets distributed across the load resistor. While this is not true DC as yet, the diode and resistor form the beginning of a simple half-wave rectifier power supply circuit.

ACTIVITY #2: ASSEMBLING THE THREE-PHASE AC ALTERNATOR

The three-phase AC alternator kit is used to generate the AC voltage for the half wave and full wave rectifier circuits. While we could have supplied you with a simple wall transformer that converts 110 VAC to a much lower voltage like 10 VAC, that would not have been as much fun. Instead, we chose to supply you with a miniature wind-powered AC alternator that will be both fun to build as well as fun to use. Plus, you will use it in the next experiment to demonstrate how three-phase power is generated.

Parts Required

(1) Turbine Kit:

- (1) Stator (thick disk with embedded wires, may be black or white)
- (1) Rotor (black disk with magnets)
- (3) Rotor Blades
- (2) Triangular hubs
- (1) 8" threaded rod, $\frac{1}{4}$ " diameter
- (1) Jam nut (thin nut)
- (1) Hex nut (thick nut)
- (2) Spacer washers
- (1) Split lock washer
- (1) Acorn nut
- (15) 3/8" pan head Phillips screws
- (4) Vinyl self-stick feet

Tools Required

Parallax Screwdriver

7/16" wrench (not included)

WD-40 or other lubricating material (not included)



Figure 5-3
Parts for the Wind
Turbine

5

CAUTION: POWERFUL MAGNETS!

Before building the three-phase AC alternator you must be made aware of the potential damaging effects of the powerful neodymium magnets used in the rotor section. While small in size these six magnets can be dangerous to your fingers, your heart and your computer. So here are some common sense rules to follow when constructing the three-phase AC alternator.

- Don't allow the magnets to come in contact with any other ferrous metal like iron, steel or copper. The magnets will immediately bind to the metal with the result being possible pinched fingers. Also, the magnetic hold may be powerful enough to tear the magnets from the round rotor disk when you try to separate them from the metal object.
- Don't allow the magnets or, for that matter the entire three-phase AC alternator, near your computer or computer disks, since there is the distinct possibility of erasing the data on the disks. This is particularly true for your computer's hard drive where the bulk of your programs are stored.
- Don't let anyone with a heart pacemaker near the three-phase alternator. The magnets can certainly interfere with a pacemaker's function.

The three-phase AC alternator is composed of a stator with the protruding six-conductor cable, a round rotor disk with six magnets, a three-bladed wind turbine and a section of $\frac{1}{4}$ " threaded rod along with some washers and nuts to hold everything together. Here's how to build it.

- ✓ Referring to Figure 5-4 and using 4 of the 12 3/8" Pan Head Phillips screws, begin the turbine blade assembly by attaching both triangular hubs on the opposite ends of a single blade.
- ✓ Make sure that both triangular hubs are mounted with the washer ends facing away from the interior of the blade assembly. DO NOT tighten the screws yet.



Figure 5-4
Attaching the First
Blade of the Wind
Turbine

- ✓ Referring to Figure 5-5, attach the other two blades in the same way, squaring up the blades with the triangular hubs and tightening all twelve screws so that the three blades fit securely to the two triangular hubs.
- ✓ Make sure that the blades are squarely aligned with the hubs; this may require un-tightening some screws and then re-tightening them until the blades are correctly aligned.



Figure 5-5
Rotor Blades Attached
to Hubs

- ✓ Referring to Figure 5-6, attach the completed blade assembly to the non-magnet side of the rotor disk with the remaining 3 screws by inserting each screw through a hole in the magnet side of the rotor disk and then into a corresponding hole on the triangular hub.
- ✓ Try not to allow the screws or screwdriver to attach itself to the magnets (but perhaps it is unavoidable).
- ✓ Now tighten the screws so that the rotor disk attaches firmly to the blade assembly.

5



Figure 5-6
Assembled Wind
Turbine and Rotor

- ✓ Referring to Figure 5-7, screw the jam nut (the thinner nut) to one end of the threaded rod, making sure that it fits “flush” with the end of the rod itself.
- ✓ Next, insert the $\frac{1}{4}$ " split lock washer on top of the jam nut, and then insert the free end of the threaded rod through the bottom (non-wire) side of the stator.



Figure 5-7
Threaded Rod
Assembly #1

- ✓ Referring to Figure 5-8, on the wire side of the stator, screw the $\frac{1}{4}$ " hex nut (the thicker nut) all the way down the long part of the threaded rod.
- ✓ Tighten the bottom (jam) nut with the wrench so the threaded rod is securely in place. Next, drop one (1) spacer washer down the long length of threaded rod so that it sits on top of the hex nut.



Figure 5-8
Threaded Rod
Assembly #2

- ✓ As shown in Figure 5-9, slip the blade assembly over the threaded rod being careful not to have the magnets attach themselves to the rod. If they do, carefully detach the magnet rotor so as not to pull any of the magnets off the rotor disk.
- ✓ Finally, attach the acorn nut to the top of the threaded rod; finger tight on the acorn nut will be fine.



Figure 5-9
Assembled Wind Turbine
three-phase AC Alternator
Assembly

5



What if my turbine is a mirror image of this turbine? It is possible to assemble the turbine to rotate in a clockwise or counterclockwise direction. In this book, we are using a turbine that rotates counterclockwise. If you assembled yours to turn clockwise, some of your graphs may display a sequence backwards from those shown in the plots in this book.

- ✓ As shown in Figure 5-10, attach four rubber feet to the bottom of the stator so that the stator sits level on a flat surface. If the assembly wobbles, then make sure that the jam nut on the bottom side of the stator assembly is flush with the threaded rod. Release and retighten, if necessary.



Figure 5-10
Attach the Rubber
Feet

- ✓ Now set the unit upright and give the turbine blades a spin. The blades should spin freely on the threaded rod, and the magnets should be rotating about a 1/16" to 1/8" above the stator.
- ✓ If you hear any scraping noise, check the separation between the rotor magnets and the stator to make sure the proper gap is present. You may need to add another spacer washer on top of the one now there.
- ✓ If you need to reduce the “clatter” of the blade assembly against the threaded rod, attach a piece of transparent tape around the threaded rod where the triangular hub’s bushing touches it – both top and bottom. The tape will act as a bearing to make the blade assembly rotate smoother and quieter. However, make sure not to use too much tape; otherwise the blade assembly will drag against it.



Figure 5-11
Adding
Transparent Tape
to Reduce
Clattering

ACTIVITY #3: BUILDING THE HALF WAVE RECTIFIER

With the three-phase AC alternator assembled, it's time to construct the half wave rectifier circuit on your Board's prototyping area. You can remove all but the A/D Converter circuit from the breadboard area since we are done with the DC portion of our experiments.

Parts Required

This experiment uses the A/D converter, and adds to it:

- (1) 1N4148 diode
- (1) 1 k Ω resistor

(1) Assembled wind turbine (three-phase, wind driven AC alternator)

(1) Table fan (not supplied)

Wire strippers (not supplied)

- ✓ Remove all but the A/D Converter Circuit from your breadboard.
- ✓ Build the Half-Wave Rectifier circuit shown in Figure 5-12, setting aside the 1000 μ F capacitor for the moment.
- ✓ If necessary, use your wire strippers to expose $\frac{1}{2}$ " on the ends of the orange wire and the orange-white wire.

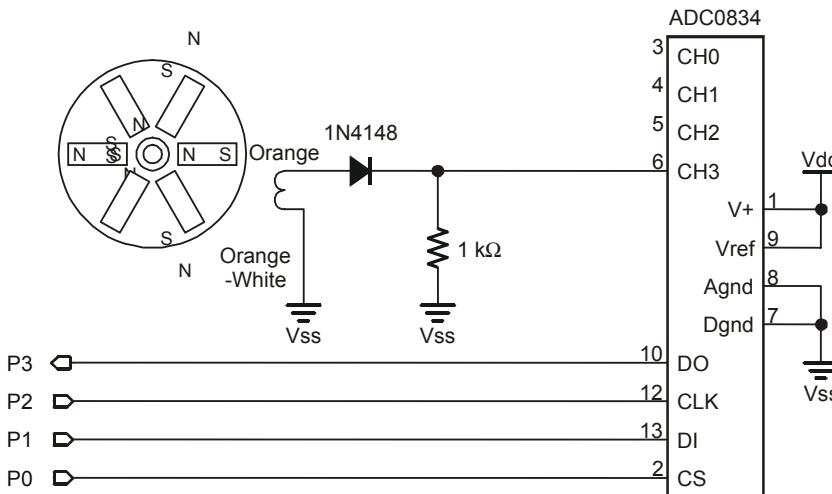
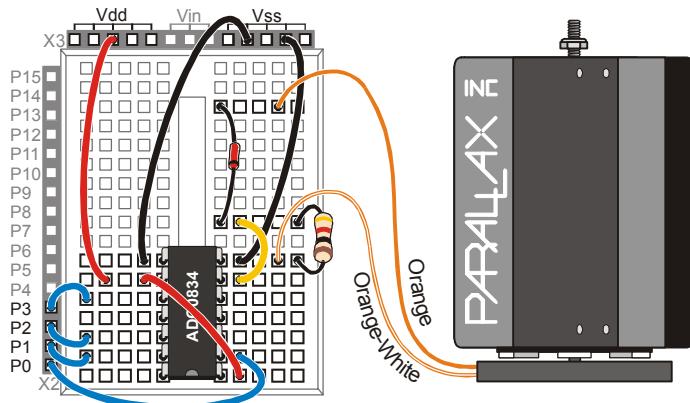


Figure 5-12: Half Wave Rectifier Schematic

- ✓ Check your parts placement against the wiring diagram shown in Figure 5-13.

We will be using the CH3 A/D converter input (pin 6) for our tests. As its name implies, the three-phase AC alternator has three sets of coils, each with two wire conductors that connect to the ends of coil. For this experiment, choose the orange and orange-white wires that emanate from the 6-wire cable. We will use the other coil wire pairs later on in Experiment 5.

**Figure 5-13: Half Wave Rectifier Wiring Diagram**

Since the half wave rectifier circuit is tied directly to one phase of the three-phase AC alternator, its AC input depends directly on the speed at which the rotor magnets spin over the stator coils. The faster they spin, the higher the amplitude and frequency of the sine wave that's produced. The wind captured by the turbine blades power the rotor's spin, however under normal circumstances the wind may not be blowing fast enough or consistently enough to render a valid experimental result. Therefore, we suggest that you find a small electric fan to power the turbine blades for this experiment and the next. The idea is to keep the turbine blades spinning at a reasonably fast and constant rate of speed. That way, the AC sine wave will have a consistent amplitude and frequency, which is what our experiment needs. In doing this, be sure to anchor the stator to the surface it's sitting on so that it does not get blown over by the pressure of the wind from the fan. We've found that a little tape or bracing the bottom stator against a book works fine.

ACTIVITY #4: PROGRAMMING THE HALF WAVE RECTIFIER

Once again, we will be building upon our previous program. These additions let us read and compare the voltages generated by the two solar arrays. Remember, if you don't want to modify and update your program as shown here, you may download all of the source code for this text from www.parallax.com.

- ✓ In the BASIC Stamp Editor, open *DuelingSolarCells.bs2*.
- ✓ Rename and save the program as *Half-FullWaveRectifier.bs2*.
- ✓ Update the title to read as follows:

' Experiments with Renewable Energy v1.0 - Half-FullWaveRectifier.bs2
 ' Takes AC voltage from wind-powered alternator and rectifies it to DC.

- ✓ Add the following code to the bottom of the Declarations section.

```
' ----- For Experiment 4: Half and Full Wave Rectification -----
sampleCount  VAR      waitCount 'Nib      ' Number of A/D samples per period
sampleArray   VAR      Byte(10)       ' 10-element array for A/D samples
```

- ✓ Next, in the **Main** routine, add an apostrophe in front of the first three **GOSUB** commands as shown below, “commenting out” the previous experiments which would conflict with the Half and Full Wave Rectification experiment.

```
' -----
' Main Routine
' -----
Main:
DO
  ' GOSUB Exp_1                      ' Programmable Battery Charger
  ' GOSUB Exp_2                      ' Dueling Solar Cells
  ' GOSUB Exp_3                      ' Solar Cell Sun Tracker
  GOSUB Exp_4                        ' Half&Full Wave Rectification
  GOSUB Exp_5                        ' Three-Phase AC Alternator
  GOSUB Plot_It                      ' Plot Data w/ StampPlot Pro
LOOP
```

- ✓ Then document the **Exp_4** subroutine as shown:

```
' -----
' Experiment 4: Half and Full Wave Rectification
' -----
'
' Set the A/D converter to convert only on ch3
'
' If sampleCount is not equal to 10
'   Then plot existing samples
'   Else prepare to plot new samples
'
' To plot new samples first wait for the rectified wave to equal zero
' Once zero, wait for the beginning of the first peak
' Then take ten (10) samples as rapidly as possible
' Zero sampleCount to begin plotting next
'
' Plot each sample in the order taken using Plot_It
' Increment sampleCount and
```

```
' Exit
'
```

✓ Then fill in the code between the **Exp_4** and **Exp_4_End** labels as shown:

```
Exp_4:
    a2dMuxId = A2dMuxId3                      ' Set the A/D to convert Channel 3
    IF (sampleCount <> 10) THEN
        GOTO Exp_4_Plot
    ENDIF

Exp_4_Wait_For_Zero:                         ' Loop until the rectified wave is zero
    GOSUB A2D
    IF (a2dResult <= 5) THEN
        GOTO Exp_4_Wait_For_First_Peak
    ENDIF
    sampleCount = sampleCount + 1
    IF (sampleCount = 10) THEN                  ' If the rectified wave stays above zero
        GOTO Exp_4_Take_Samples                ' for 10 counts, assume that
    ENDIF                                       ' a smoothing capacitor is attached
    GOTO Exp_4_Wait_For_Zero

Exp_4_Wait_For_First_Peak:                    ' Loop until the first peak is detected
    GOSUB A2D
    IF (a2dResult > 5) THEN Exp_4_Take_Samples
    GOTO Exp_4_Wait_For_First_Peak

Exp_4_Take_Samples:
    FOR sampleCount = 0 TO 9                  ' As rapidly as possible take 10
        GOSUB A2D                            ' A/D samples and save them in the
        sampleArray(sampleCount) = a2dResult   ' sampleArray
    NEXT
    sampleCount = 0                          ' Re-initialize sampleCount
                                            ' for plotting

Exp_4_Plot:
    ch3 = sampleArray(sampleCount)           ' Get the next A/D sample into CH3
    sampleCount = sampleCount + 1            ' and increment sampleCount

Exp_4_End:
    RETURN
```

✓ Save your work!

ACTIVITY #5: HOW THE HALF WAVE RECTIFIER PROGRAM WORKS

Remember our floor and superstructure analogy from the earlier part of this manual. Well, by commenting out these calls, what we've done is sealed off the first three floors

of our software building while we work on the final two floors, just as we sealed off the Battery Charger experiment on the first floor while we worked with the solar cells. Also notice how simple, yet elegant, this is. By just a single comment in each call to `Exp_1`, `Exp_2` and `Exp_3` we have effectively disabled their software actions, since the BASIC Stamp program will skip these calls entirely and branch directly to `Exp_4`. Make a note of this software technique because professional programmers use it all the time to segment their programs for debugging and, also, to bring in or take out elements of the program at the proper times. So now that the DC experiments are “patched out”, let’s continue with our current experimental code.

First, look at what we added to the Declarations section:

```
sampleCount  VAR      waitCount  'Nib
sampleArray  VAR      Byte(10)
```

The nibble variable, `sampleCount`, counts the number of times the A/D converter samples voltage readings (up to 16 counts max, the capacity of a nibble). Again, we are re-using variable space from a subroutine that won’t be called in this experiment. Next comes `sampleArray`, a ten-element array for storing these converted voltage values. Don’t worry too much about understanding these variables right now; their meanings will become clearer once you see them in the body of the code, which we’ll get into right now.

```
Exp_4:
    a2dMuxId = A2dMuxId3
    IF (sampleCount <> 10) THEN
        GOTO Exp_4_Plot
    ENDIF
```

At `Exp_4` the first instruction sets the A/D converter to CH3 (pin 6). The next instruction tests `sampleCount` for any value other than 10. If `sampleCount` is not 10 it implies that a previous sample of A/D values have been taken, so the program branches to the `Exp_Plot` label. However when `sampleCount` does equal 10 the `IF...THEN` statement fails and the program prepares to take another 10 samples beginning at `Exp_4_Wait_For_Zero`.

```
Exp_4_Wait_For_Zero:
    GOSUB A2D
    IF (a2dResult <= 5) THEN
        GOTO Exp_4_Wait_For_First_Peak
```

```

ENDIF
sampleCount = sampleCount + 1
IF (sampleCount = 10) THEN
    GOTO Exp_4_Take_Samples
ENDIF
GOTO Exp_4_Wait_For_Zero

```

The code segment under **Exp_4_Wait_For_Zero** gets a little ahead of our experiment at this point since the concept of a “smoothing capacitor” hasn’t been introduced as yet. Nevertheless, the primary function of the code is to detect when the half wave rectified wave goes to zero. What we’re trying to do is “sync up” to the next positive going wave that comes along, but in doing so we first need to find out where we are on the wave at this point. If the value of the A/D converted voltage is above zero (actually 0.10 volts where $5 * 0.02\text{volts/bit} = 0.10$ volts due to possible biasing conditions) we may have captured the wave somewhere during its positive cycle. Or it may never go to zero if a smoothing capacitor is attached (there’s more to come on this, so be patient). To properly sync up, we need to capture the wave at the very beginning of its positive cycle, and this is why we need to first capture the zero part of the wave; that is, to ensure that we will begin our samples at the beginning of the next positive wave cycle, which is what the code under **Exp_4_Wait_First_Peak** does.

```

Exp_4_Wait_First_Peak:
    GOSUB A2D
    IF (a2dResult > 5) THEN Exp_4_Take_Samples
    GOTO Exp_4_Wait_First_Peak

Exp_4_Take_Samples:
    FOR sampleCount = 0 TO 9
        GOSUB A2D
        sampleArray(sampleCount) = a2dResult
    NEXT
    sampleCount = 0

```

When the code branches to **Exp_4_Take_Samples** we are “synced up” with the next positive wave. Therefore, the remaining code takes ten (0 – 9) A/D samples as quickly as possible and stores them to **sampleArray**. Once this is done, the **sampleCount** is re-initialized to zero so that the **Exp_4_Plot** routine can begin at the first sample of this group.

```

Exp_4_Plot:
    ch3 = sampleArray(sampleCount)

```

```
sampleCount = sampleCount + 1  
  
Exp_4_End:  
RETURN
```

The **Exp_4_Plot** routine reads the next value of converted data from the **sampleArray** into variable **ch3** and increments **sampleCount** for the next time through **Exp_4**. The **Plot_It** routine handles the plotting of the **ch3** value. The **sampleCount** is incremented to the next sample and the program exits. When **sampleCount** increments to the value of 10, the first part of the program under label **Exp_4:** detects this and will refresh the display by taking ten new samples.

5

Your Turn: Sampling and Displaying the Half Wave Rectifier Voltage Outputs

In order to take some reasonable samples of the half wave rectifier output, the table fan should be turning the turbine blades at a reasonably fast rate, and the stator should be securely fastened to a flat surface.

- ✓ Arrange your wind turbine and table fan so that the turbine blades are turning steadily and briskly.
- ✓ In the BASIC Stamp Editor, run Half-FullWaveRectifier.bs2, make a note of the COM port being used, then close the Debug Terminal.
- ✓ Open StampPlot Pro by clicking on Start → Programs → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_45.spm.
- ✓ Set the COM port in StampPlot to the same one that was being used by the Debug Terminal.
- ✓ Click on the Connect button, and make sure the Enable Plotting box is checked.

What follows is the plot of half wave rectified voltage values using StampPlot (Figure 5-14).

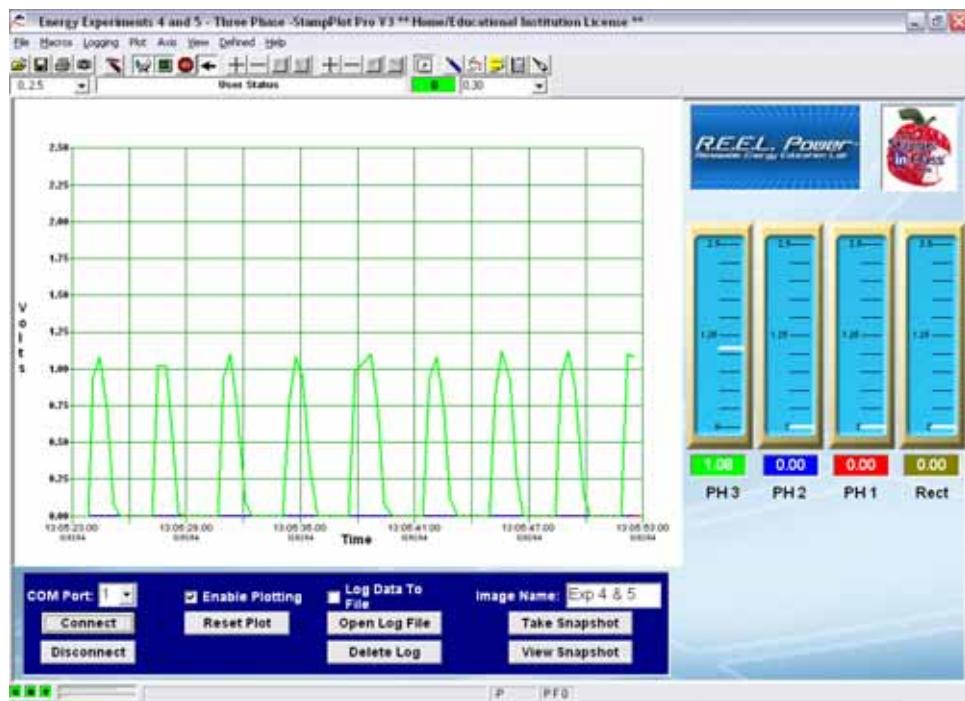
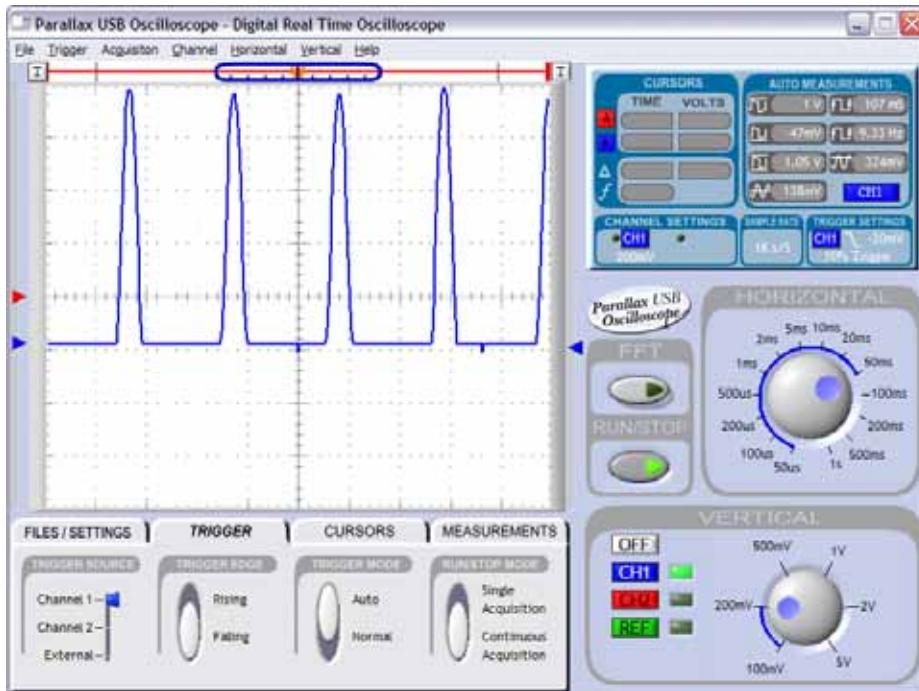


Figure 5-14: StampPlot Plot of Half Wave Rectified Voltage

Depending on how fast your turbine is spinning your plot should look close to, but not exactly like this one. Figure 5-14 used a 12 second time segment to capture this sample. As you can see, the plot shows a series of sharp peaks followed by flat, no voltage readings in between the peaks. If you were using an oscilloscope the plot would produce a much smoother waveform like that shown in Figure 5-15.



5

Figure 5-15: Oscilloscope Plot of Half Wave Rectified Voltage

What our program has done is to very quickly capture multiple samples of the half wave rectifier voltage data so that it can be displayed much more slowly using StampPlot. Remember we said earlier that StampPlot is not as fast as an oscilloscope. Therefore, in order to make allowances for its lack of display speed, StampPlot displays one sample every 250 milliseconds, the speed of the `Plot_It` routine. Try adjusting the fan speed to witness what happens to the peak values of the wave and the time between peaks. The slower the fan speed, the lower the voltage peaks and the longer the time between peaks.

Your Turn: Smoothing Things Out

To get closer to the performance of a true linear DC power supply we need to begin to smooth out the half wave rectified peaks. This is done by adding a “smoothing” capacitor to the output of the circuit as shown in Figure 5-16. By adding a 1000 μ F “polarized”

capacitor, the half wave rectified peaks will become much smoother and approach a nearly constant DC level.

Additional Parts Required

- (1) 1 k Ω resistor (brown black red)
- (1) 1000 μF electrolytic capacitor



Warning: This electrolytic capacitor has a positive (+) and a negative (-) terminal. The negative terminal is the lead that comes out of the metal canister closest to the stripe with a negative (-) sign. Always make sure to connect these terminals as shown in the circuit diagrams. Connecting one of these capacitors incorrectly can damage it. In some circuits, connecting this type of capacitor incorrectly and then connecting power can cause it to rupture or even explode.

- ✓ Disconnect power to your Board of Education.
- ✓ Add the 1000 μF capacitor to your circuit as shown in the schematic in Figure 5-16 Because the capacitor is polarized, meaning that it has a positive (+) and negative (-) terminal, take care to add the capacitor correctly as illustrated in Figure 5-17.
- ✓ Check your wiring before reconnecting power to your circuit.

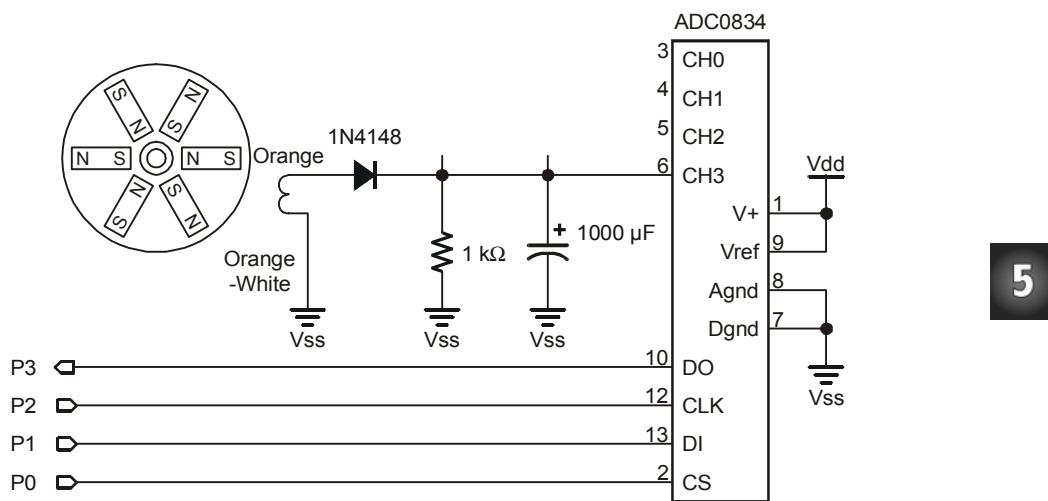


Figure 5-16: Half Wave Rectifier with Smoothing Capacitor Schematic

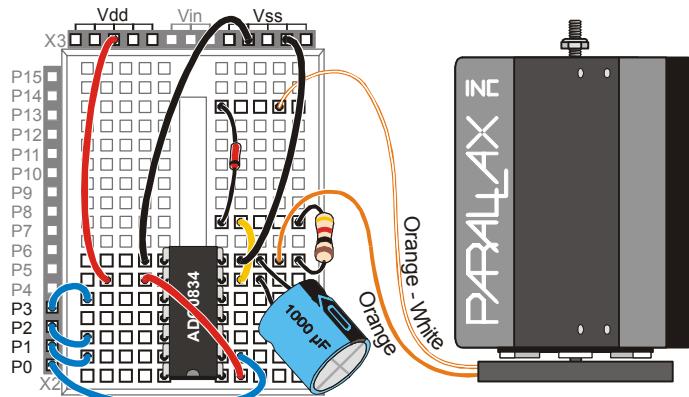


Figure 5-17: Half Wave Rectifier with Smoothing Capacitor Wiring Diagram

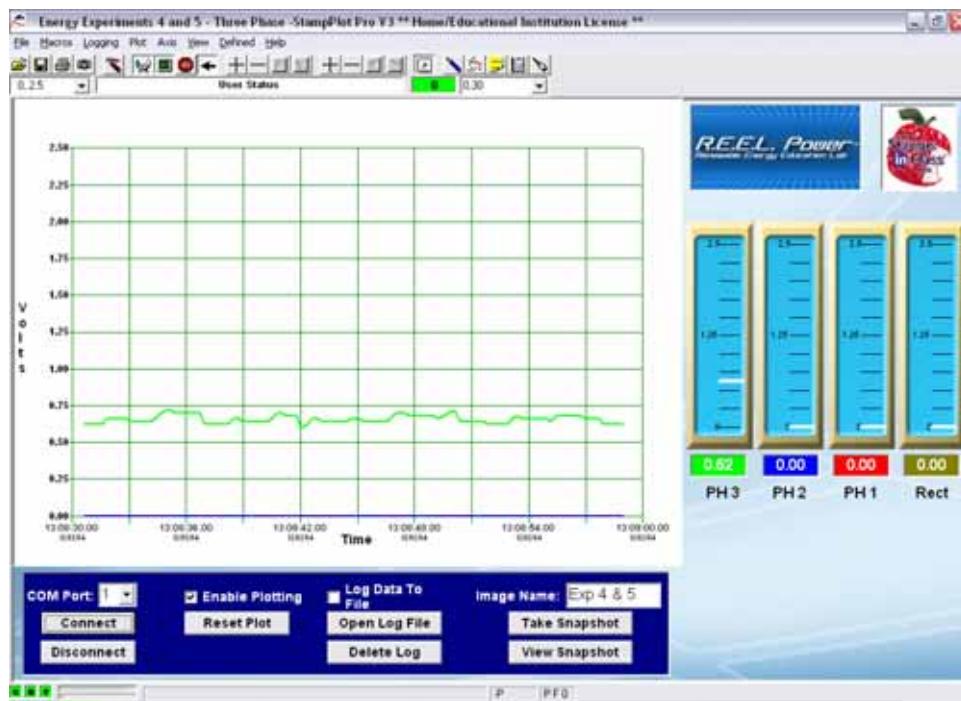


Figure 5-18: StampPlot Plot of Smoothed Half Wave Rectifier Output

Now, if you re-run your program and view the output in StampPlot, you should see something like Figure 5-18. The equivalent oscilloscope plot is in Figure 5-19.

- ✓ Re-run Half-FullWaveRectifier.bs2.
- ✓ Close the Debug Terminal, and then bring up StampPlot and connect.

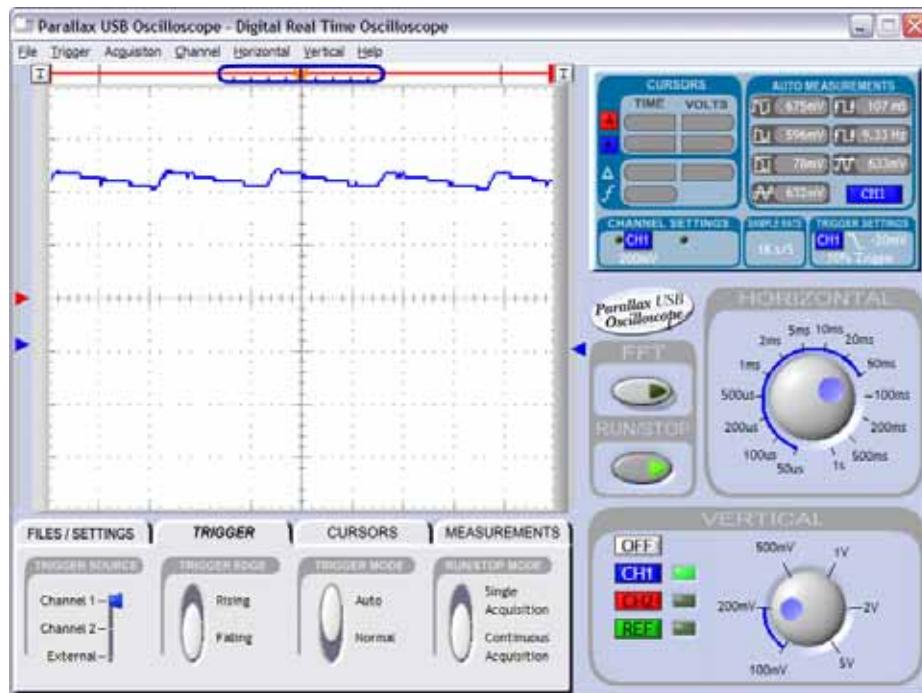


Figure 5-19: Oscilloscope Plot of Smoothed Half Wave Rectifier Output

Notice that the waveform peaks and then slowly begins to decay but never goes to zero. The reason for this is explained next. Please refer to Figure 5-20 as a visual aid to understanding the following textual explanation.

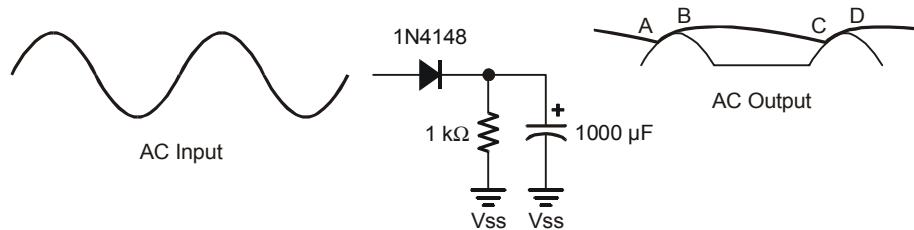


Figure 5-20: Effects of Smoothing Capacitor on Half Wave Rectified Output

When the rectified positive wave forward biases the diode, part of the current flows through the $1\text{ k}\Omega$ load resistor and the remaining part charges the smoothing capacitor to nearly the peak value of the AC Output between A and B. As the peak of the AC Output decreases to zero, the smoothing capacitor begins to discharge its energy through the $1\text{ k}\Omega$ load resistor as illustrated between B and C. This accounts for the droop in V_{out} between B and C. Then the charge-discharge cycle repeats again between C and D keeping V_{out} from dropping to zero volts.

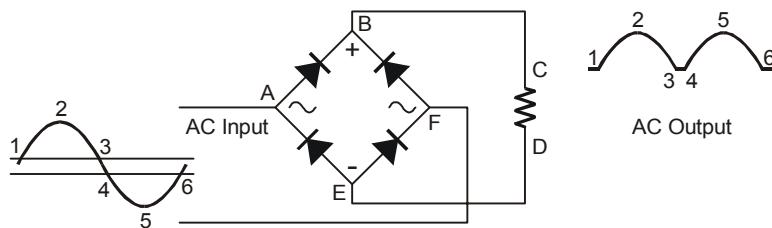
As you may have already guessed, the discharge rate of V_{out} is mainly dependent on the values of R and C, in this case the $1\text{ k}\Omega$ load resistor, R, and the smoothing capacitor, C. If we were to increase the value of the smoothing capacitor, the discharge rate of V_{out} would not decrease as fast. Correspondingly, if we were to decrease the value of the load resistor, R, the discharge rate between B and C would decrease faster along with the average DC level.

Try it!

- ✓ Disconnect power to your board.
- ✓ Add another $1\text{ k}\Omega$ resistor in parallel across the existing $1\text{ k}\Omega$ resistor. The effective resistance with two $1\text{ k}\Omega$ resistors in parallel is $500\ \Omega$, or one half the original value.
- ✓ Reconnect power to your board, and re-run your program.
- ✓ Close the Debug Terminal, bring up StampPlot, and connect.
- ✓ Now inspect the StampPlot output to verify that V_{out} decreases faster along with the average DC level.

ACTIVITY #6: FULL WAVE RECTIFIER OPERATION

Before building a full wave rectifier, let's first attempt to understand what it is and how it operates. While our half wave rectifier used only the positive portion of the AC input, a full wave rectifier uses both the positive and negative portions, thus doubling the output power. The way it does this is also quite interesting. In order to capture the full positive and negative portions of the input AC wave, a full wave rectifier uses a combination of four diodes arranged in circuit like that in Figure 5-21. This diode arrangement is generally called a "bridge rectifier" and here's how it works.

**Figure 5-21: Full Wave Rectifier Circuit****5**

The AC Input cycle is illustrated with numbers ranging from 1 to 6. The AC Output is similarly numbered to show its correspondence with the input wave.

At point 1 on the AC input wave, point A is more positive as compared with point F on the bridge rectifier. Furthermore, the voltage is just high enough to forward bias diode AB allowing it to conduct and deliver current to resistor CD. Current cannot flow into diode BF since it is reversed biased at this point. At the same time diode AE cannot conduct current since the voltage at point A, the cathode, is essentially the same as it is at point E, the anode. Therefore, current only flows from diode AB through resistor CD and then through diode EF thus completing the return path to the AC input source.

When the AC input wave reverses and goes negative at point 4, point F becomes more positive as compared with point A; this is important to note since it is key to the operation of our full wave rectifier.

To capture the negative cycle of the AC input wave, diode FB becomes forward biased at point 4 of the AC input wave, and current begins to flow through resistor CD then through diode EA, which is also forward biased at this point. The current flow then returns to the other side of the input AC wave thus completing the circuit in the negative direction. The other two diodes are reversed biased during the negative portion of the AC input.

Notice that we are labeling the components in the direction of current flow to make the description clearer.

Therefore, the four-diode bridge rectifier uses two diodes, AB and EF, to conduct current through the resistor during the positive half of the AC input cycle, while diodes FB and EA conduct current through the resistor during the negative half of the AC input cycle. The result is that the AC Output voltage contains twice as many positive peaks as

compared with the half wave rectifier, which is significant in terms of capturing nearly all of the AC input's energy.

ACTIVITY #7: BUILDING THE FULL WAVE RECTIFIER CIRCUIT

Now it's time to build the full wave rectifier circuit. To do so you will need all the components you used in the half wave rectifier circuit plus:

Additional Parts Required

(3) 1N4148 diodes

The schematic is shown in Figure 5-22. The parts placement should look like that in Figure 5-23. Examine how the four diodes are inserted into the breadboard.

- ✓ Disconnect power to your board.
- ✓ Remove the 1000 μ F capacitor for the time being.
- ✓ Add the diodes to the circuit, being careful to observe the directions of the diodes' anodes and cathodes.
- ✓ When you are confident your wiring is correct, reconnect power.

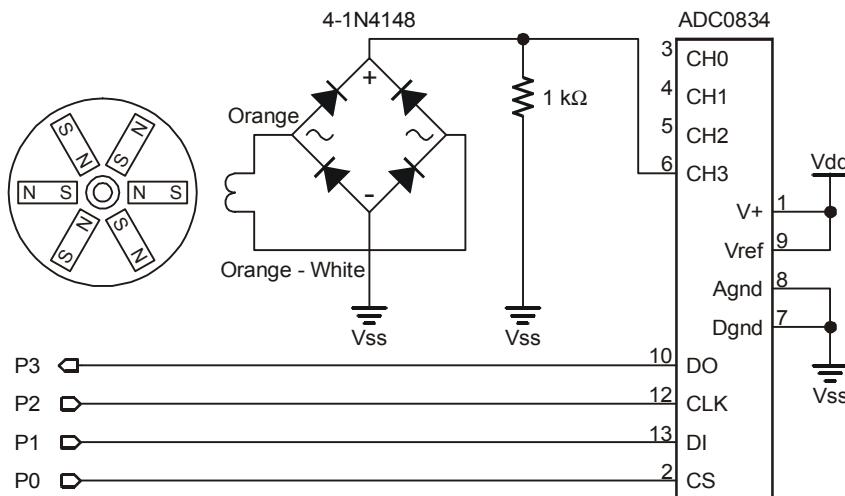


Figure 5-22: Full Wave Rectifier Schematic

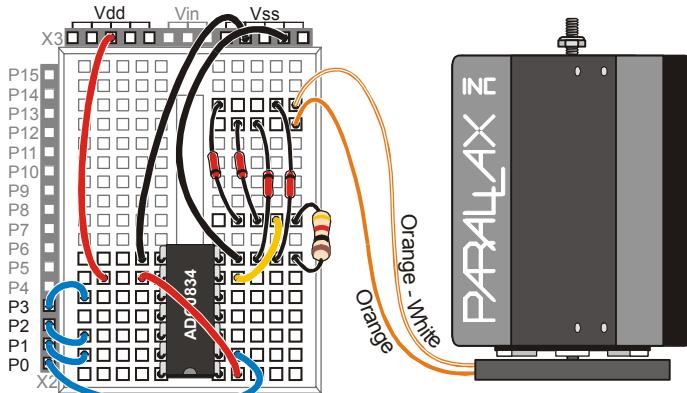


Figure 5-23: Full Wave Rectifier Wiring Diagram

Your Turn: Sampling and Displaying the Full Wave Rectifier Voltage Outputs

We will use the same code as we did for the half wave rectifier circuit, which will now take samples of the full wave rectifier output. Once again, the fan should be turning the turbine blades at a reasonably fast rate, and the stator should be securely fastened to a flat surface.

- ✓ Re-run Half-FullWaveRectifier.bs2.
- ✓ Close the Debug Terminal, bring up StampPlot, and connect.

What follows is the plot of these voltage values using StampPlot as illustrated in Figure 5-24. Once again, our plot used a 12 second time sweep.

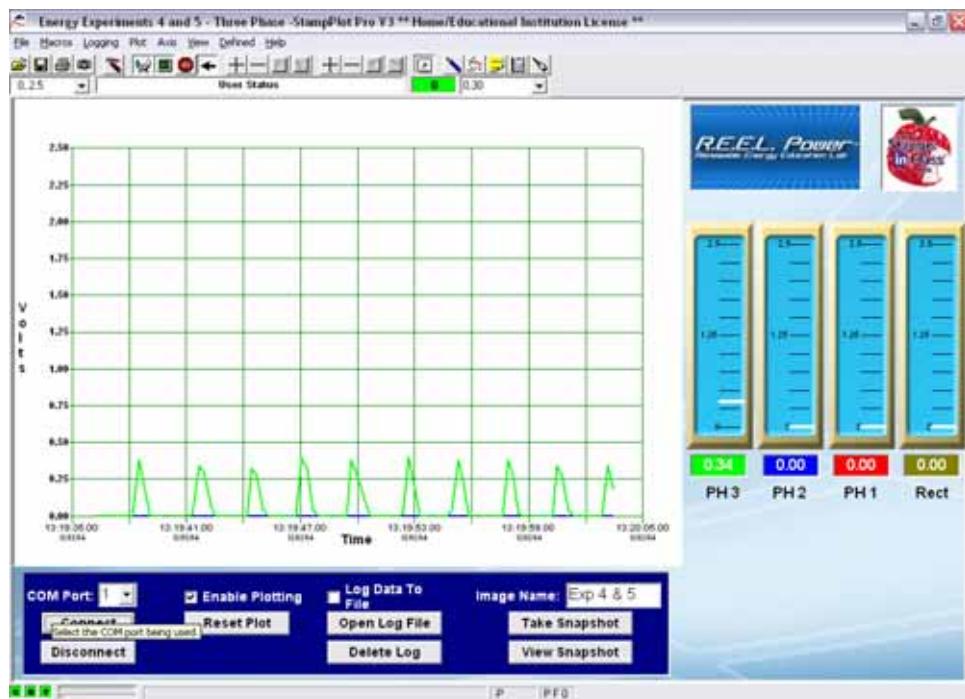
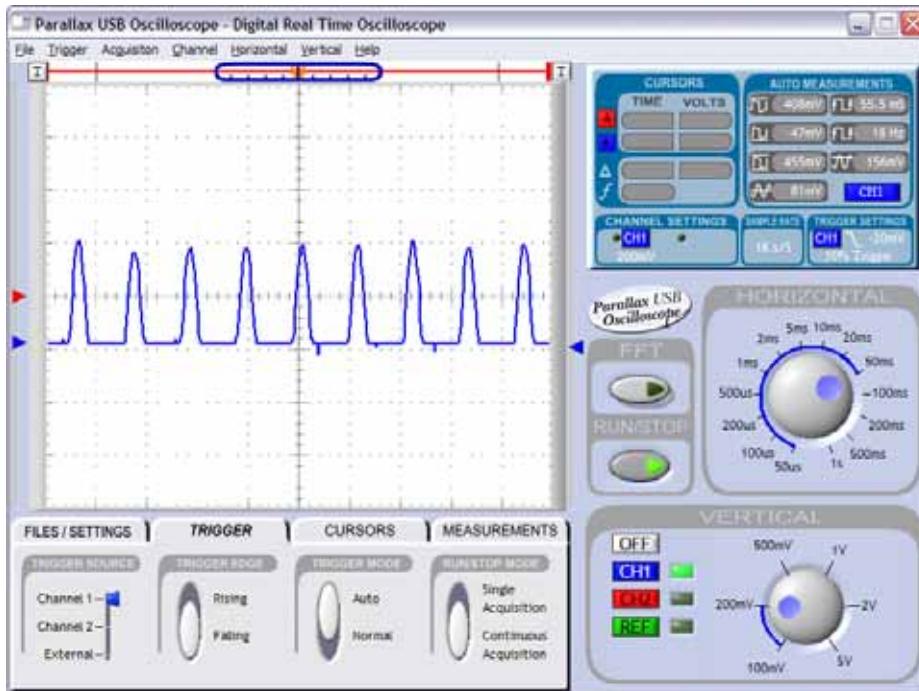


Figure 5-24: StampPlot Plot of Full Wave Rectified Voltage Values

The equivalent oscilloscope plot is illustrated in Figure 5-25.



5

Figure 5-25: Oscilloscope Plot of Full Wave Rectified Voltage

If you compare these plots to Figure 5-14 and Figure 5-15, respectively, you can see that the full wave rectified output has double the number of peaks as compared with the half wave rectified output. To be consistent in your own comparisons, make sure that the time scale you used for sampling the half wave rectified signal is the same as the one you used to sample the full wave rectified signal.

Your Turn: Adding a Smoothing Capacitor

Once again, let's add a smoothing capacitor to the full wave rectified circuit to see its effect on our full wave rectified voltage output. We'll add the same $1000\ \mu\text{F}$ polarized capacitor across the load resistor as shown in Figure 5-26.

- ✓ Replace the $1000\ \mu\text{F}$ capacitor as shown in Figure 5-27. Remember the capacitor is polarized so take care to add the capacitor with its + and - leads correctly as illustrated.

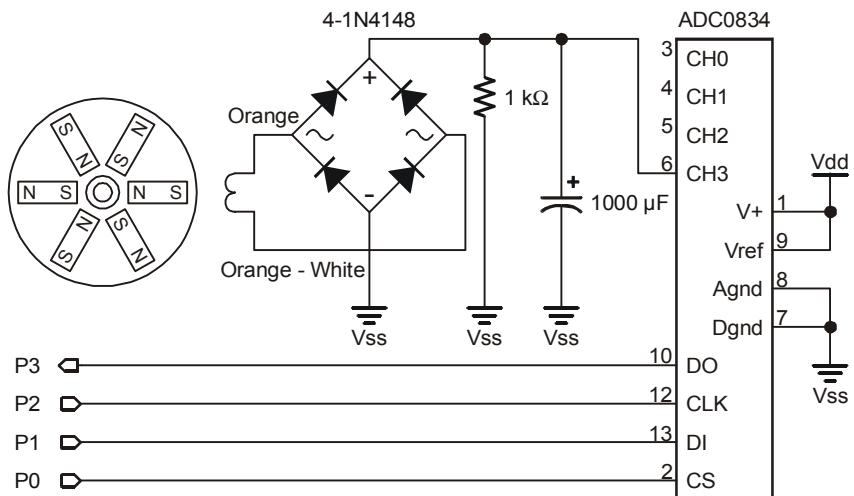


Figure 5-26: Full Wave Rectifier with Smoothing Capacitor Schematic

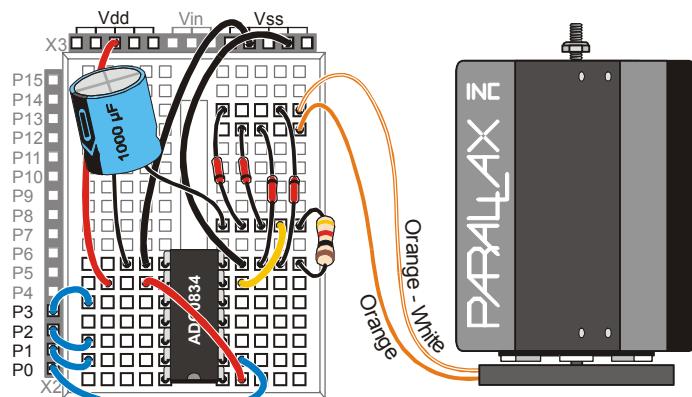


Figure 5-27: Full Wave Rectifier with Smoothing Capacitor Wiring Diagram

You should see something like Figure 5-28.

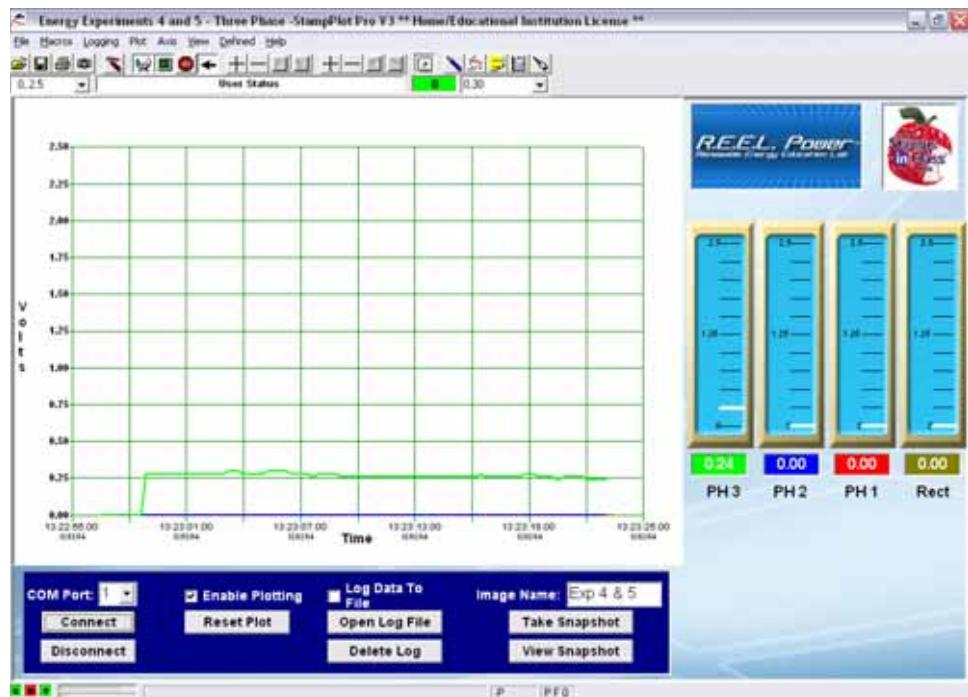


Figure 5-28: StampPlot Plot of Smoothed Full Wave Rectifier Output

The equivalent oscilloscope plot is in Figure 5-29.

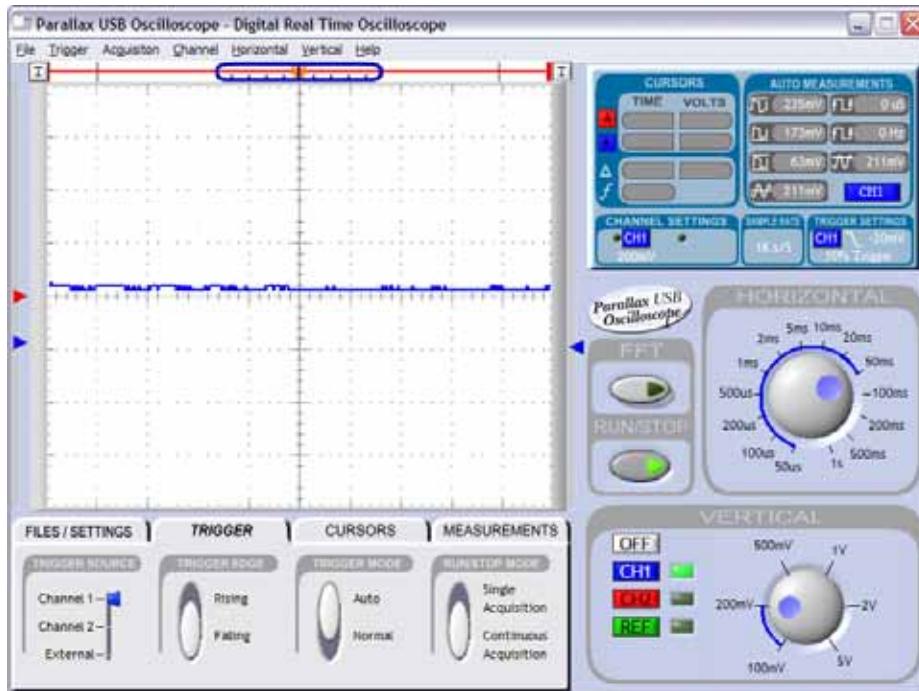


Figure 5-29: Oscilloscope Plot of Smoothed Full Wave Rectifier Output

Notice how much smoother the full wave rectified signal is in these plots as compared with the half wave rectified plots in Figure 5-18 and Figure 5-19, respectively. This, of course, is due to the full wave rectified circuit capturing twice as much energy from the AC input wave as compared with the half wave rectified circuit.

SUMMARY AND APPLICATIONS

In this first of two experiments in AC energy you were introduced to a fundamental concept in AC to DC conversion, rectification. Using a simple passive component called a diode, the conversion from AC to DC is made possible.

Beginning with vacuum tubes, diodes have been with us ever since radio was invented. In addition to their application in power supplies, diodes are also used to capture the signal portion of amplitude modulated, AM, radio waves. Every radio uses a diode circuit to “de-modulate” the AM carrier wave to create the audio that we hear. What is important

to note, however, is that prior to the 1940's there were essentially no oscilloscopes around to view the voltage characteristics of AC signals. Engineers and scientists used crude meters and incandescent lamps to "measure" these signals. As a matter of fact, the first oscilloscopes were thought to be toys by engineers, and not worthy of their academic "predictions" of waveforms. Fortunately, science has progressed to the point that waveform measurement using oscilloscopes is now commonplace. And you are fortunate to benefit from these technical advances.

In this experiment you used small-signal silicon diodes, which are also the same variety of diodes used in radios, televisions and stereos. On the other end of diode technology are the power diodes that handle large currents, sometimes in the kilo-amp range. Regardless of their current handling capabilities, all diodes operate in the same way allowing current to pass only in one direction, from anode to cathode. Therefore, by learning how diodes work to rectify AC signals, you are ready to expand this learning into the many and varied applications for diodes.

While half wave rectification is simpler than full wave rectification, full wave rectification is preferred for modern power supplies. By learning the fundamentals of both half and full wave rectification, you have learned the basis for standard linear power supply design.

REAL WORLD POWER SUPPLY DESIGN

Power supplies come in a wide variety of voltage outputs and packaging. Here are just a few examples of them.

The photo is one of the most common multi-voltage switching power supplies used in desktop computers. This power supply will fit nicely into almost every kind of PC enclosure. Its typical efficiency is 75% at full load with minimum ripple and noise. These power supplies also have auto voltage select (AVS) functions, which means that they can automatically detect either 110 VAC or 220 VAC inputs and adjust for either input voltage accordingly.



Figure 5-30
Computer Power Supply

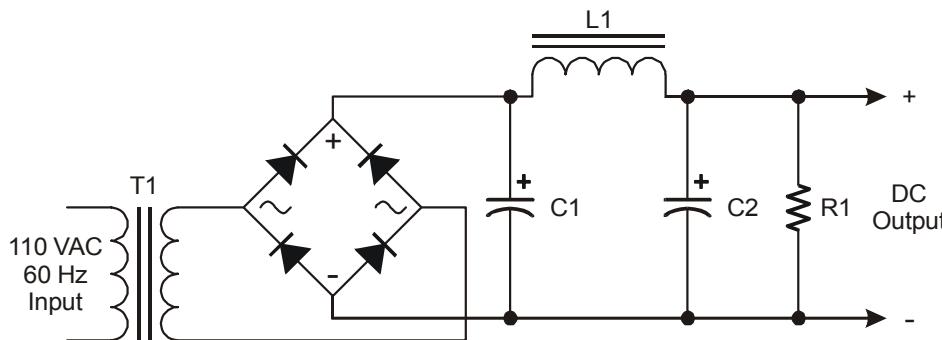
Pictured below is another common type of power supply called an “open frame” power supply. The “open frame” designation comes from the fact that the power supply components are not enclosed in a metal box like the computer power supply above. This not only saves cost but also provides for more air circulation around the individual components, especially the diode rectifiers, which are attached to the black metal heat sinks. Notice the cut out for a fan in the computer power supply above. Because it's enclosed, a fan is needed to circulate cool air around its components, which, by the way, look quite similar to the open frame model pictured here.



Figure 5-31
Open Frame Power Supply

Linear Power Supply Design

At the end of Experiment 4 we mentioned that this chapter would have more information on power supply design, and specifically on how to convert the full wave rectified circuit you built into a classic linear power supply. True to our word, here are more details on linear power supply design beginning with Figure 5-32.



5

Figure 5-32: Linear Power Supply Schematic

The first thing that's different is that the wind-powered three-phase AC alternator has been replaced by a conventional transformer, T1, which isolates the rest of the circuit from the 110 VAC, 60 Hz input and also steps the voltage down according to the power supply's DC output requirements. Every transformer has a "primary" and a "secondary" winding. The part of the transformer that is connected to the 110 VAC source is called the primary winding, or just the primary, and the secondary winding, or secondary, is connected to the 4-diode bridge rectifier. Some transformers also have multiple primary and secondary windings, as well.

In general, the higher the DC output, the higher the secondary voltage. In this simple power supply circuit the DC output is approximately equal to the secondary voltage multiplied by 1.414, however this is a rather simplistic calculation and does not take into consideration the many variables that can affect this assumption like transformer loss, output ripple and especially varying loads. At light loading this rule can be applied without much concern and it will be accurate enough for most applications. However when an appreciable amount of current is drawn, this simple approach may not yield the proper results. Nevertheless, we will not stress our power supply hard enough to violate this assumption in this example.

Since the applied 110 VAC spends so much of its time at a voltage lower than that of the capacitor, there is no diode conduction during this time. You learned this in Experiment 4 when you added a smoothing capacitor to the circuit. During the periods when the diodes do conduct, the transformer must replace all the energy that drained from the capacitor in the intervening 8 to 9 ms assuming a 60Hz AC input and full wave rectification as we

have here. Our smoothing capacitor helps to do this but unfortunately it's not enough. As you saw in Figure 5-33 – Oscilloscope Plot of Smoothed Full Wave Rectifier Output (repeated here) the smoothing capacitor still leaves a bit of “ripple” on the DC output.

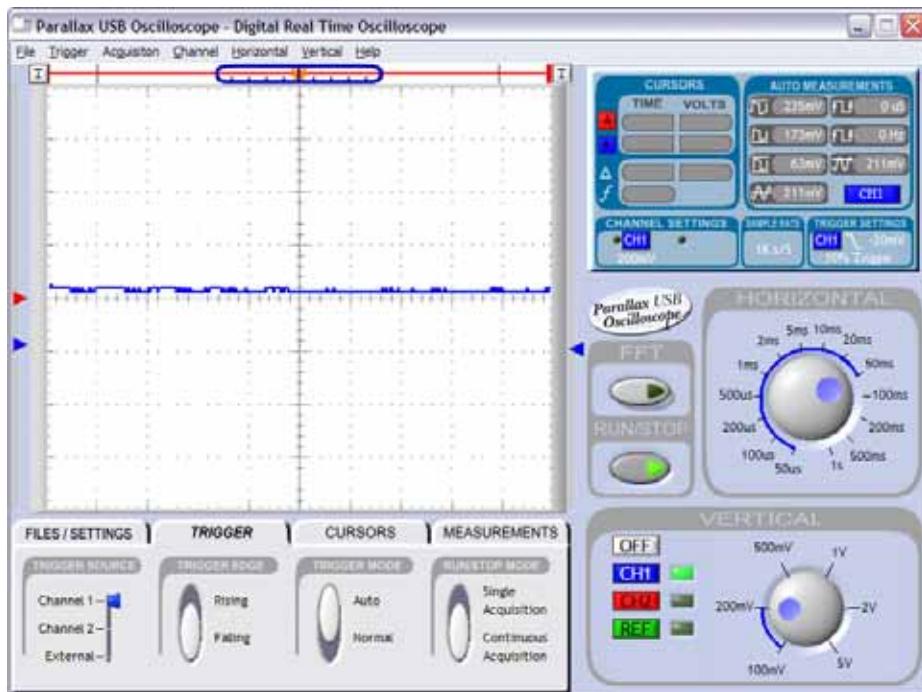


Figure 5-33: Oscilloscope Plot of Smoothed Full Wave Rectifier Output

So, what is “ripple” and why is it not desirable? Ripple is a slang, but accepted, term for the voltage variations that “ride on top” of the DC output. It’s not desirable because when the power supply output is applied to power an audio amplifier, for example, this ripple will translate into an annoying hum on the speakers. There are other deleterious effects that ripple can cause, however this is most pressing of them all for the majority of commercial products. The idea is to add a “choke” inductor in series with the DC output to help eliminate the ripple.

In our schematic diagram in Figure 5-32 our choke inductor, L1, serves to smooth out the ripple by “choking off” the AC component of the DC output, which is why engineers and

technicians long ago coined the word for it. While a capacitor will not pass DC and allow AC to charge it up, an inductor will not pass AC but will allow DC to pass without much loss. These are very simplistic definitions for capacitors and inductors, however they will suffice for this example. With its’ “back EMF” capability inductor, L1, acts to cancel the AC component that rides on the DC output. The combination of capacitor, C1, and inductor, L1, form a low-pass filter that helps to smooth out the DC output.

Capacitor, C2, is generally called a filter capacitor since its purpose is to cancel the remaining AC ripple even more. So with a constant resistive load as represented by resistor, R1, we have a linear power supply that is effective in delivering a reasonably “clean” DC signal to whatever it happens to be connected to.

Switching Power Supplies

Power supply designs have come a long way since the linear power supply was developed. Most power supplies in use today are of the “switching” type rather than the linear type described above. And you can find switching supplies in every computer sold.

Switching power supplies use higher frequency techniques to convert the 110 VAC, 60 Hz power source into clean, reliable DC. Their designs are also far more efficient in terms of converting AC to DC. The big problem lies in how to cancel out all the high frequency components that they generate internally before they become part of the DC ripple output. So while switching power supplies are cheaper, more cost and power efficient, weigh less and come in smaller packages as compared with linear supplies, their big drawback is in their design complexity.

To aid in lessening the burden of designing switching power supplies, many integrated circuit manufacturers have embarked in making ASICs (Application Specific Integrated Circuits), which can be used in all sorts of ‘switcher’ designs. One particular company, National Semiconductor Corporation has a website dedicated to simple, yet effective, switching power supply designs. A design engineer can log on to their WebBench site at <http://www.national.com/appinfo/power/> and design a good switching power supply by just “filling in the blanks” (below).

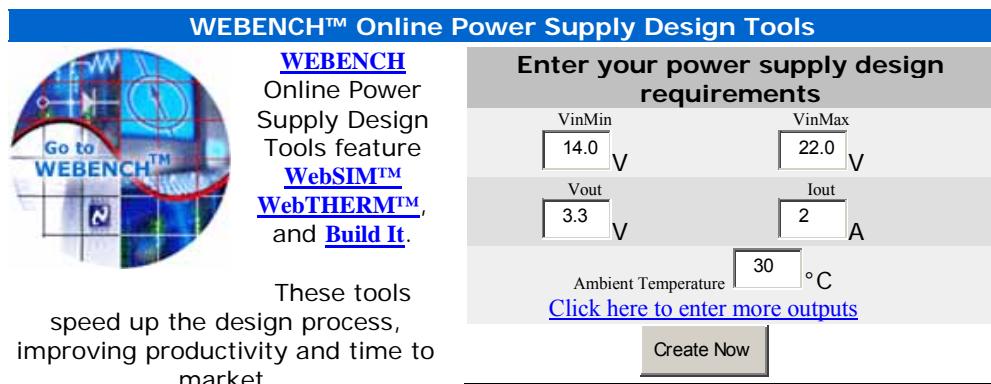


Figure 5-34: National Semiconductor's WEBENCH™ Software

Images © National Semiconductor; used with permission.

POWER ELECTRONIC SYSTEMS

Power electronic systems, as opposed to simply power supplies, widen the scope of energy conversion and control. In Experiment 4 you learned about the fundamentals of converting AC to DC using two forms of rectification – half wave and full wave. While certainly interesting, it is just the beginning as far as learning about power supply design and power electronic systems. This section will help you to fill in the gaps.

In the Preface we said that power electronics would be the dominant electronic technology for the beginning of the 21st Century. Apparently at least one prominent University agrees with this assessment. According to an excerpt from the Rensselaer Polytechnic University (Troy, NY) web site....

Estimates indicate that as much as 80% of all electric energy may be electronically processed within the next decade. Such rapid proliferation of power-electronic systems makes it imperative for power engineers to understand the principles by which power-electronic converters are designed and operated, and the implications of the converters on the source and load systems. We are currently engaged in the design and development of power-electronic systems

and their control with application to electrical machinery, power system control, lighting, and power-quality improvement. [<http://www.rpi.edu/>]

The key words in this paragraph occur in the first sentence – “electronically processed” –, which implies that control, or more precisely “micro-control”, will be responsible for 80% of all electricity that is generated by power plants and converted by power supplies, inverters, motor controllers, etc. No longer do engineers “design blind” by leaving out the most prolific and powerful component available; the microcontroller. Microcontrollers like the BASIC Stamp are now being used in nearly every electronic design, and especially in power electronic systems. The reason is simple. A microcontroller can monitor and control the operations of power systems as well as gather and store (log) data and output the data using a conventional RS-232 serial port. Knowing that microcontrollers can do all this puts you ahead of many other seasoned engineers who still attempt to resist the incursion of these devices into their designs. So consider yourself fortunate in so far as you are on the cutting edge of a new revolution in power electronic systems; that is, power systems that have “brains” courtesy of the microcontroller you are studying in this course.

5

Universities Involved in Power Electronic Systems

In the United States a consortium of five universities are heavily involved in the study of Power Electronic Systems. While not the only universities involved in power electronics research, the Center for Power Electronic Systems, or CPES, is recognized worldwide for its unique contributions to the study of power electronics. CPES was established in 1998 as one of our nation's relatively few National Science Foundation Engineering Research Centers. Its vision is to provide the nation with the capabilities to become a world leader in power electronics

The five Universities involved in CPES are Virginia Tech, the University of Wisconsin-Madison, Rensselaer Polytechnic Institute, North Carolina A&T State University and the University of Puerto Rico – Mayagüez. Each University has its own particular research specialty, however all five Universities work together in a coordinated fashion in order to share their research with each other as well as industry. Many student programs are also available that teach the state of the art science and math behind modern power electronics.

You can learn more about CPES by visiting their website at <http://www.cpes.vt.edu/>.

Industry Organizations Involved in Power Electronic Systems

While the CPES Universities, and others like them, deal with cutting-edge research and development in Power Electronics, it takes the engineering and manufacturing skills of many commercial companies to put their research into practice. One organization that is at the forefront of representing these manufacturers is the Power Sources Manufacturers Association, or PSMA. The PSMA represents over 125 manufacturers within the Power Electronics area and co-hosts the annual APEC (Applied Power Electronics Conference and Exhibition) trade show where vendors can highlight their latest commercial offerings to interested companies and engineers.

The PSMA also hosts many student-oriented programs with an emphasis on learning about Power Electronics. In 2002 the PSMA sponsored its first annual Power Education Award to recognize organizations that promote Power Electronics in Kindergarten through Undergraduate levels. LearnOnLine, Inc. and the principal author of this book were honored to receive this prestigious award for REEL Power – a K-12 Renewable Energy Education Lab project. It is heartening to know that organizations like the PSMA are reaching out to students at every grade level to make them aware of the opportunities and challenges involved in the Power Electronics field.

You can learn more about the PSMA by visiting their web site at www.psma.com.

You can also learn more about REEL Power by going to Appendix G - LearnOnLine's REEL Power Project.

More Information

You can find more information on Power Electronic Systems at the following web site:
www.learnonline.com/ewre.html.

Chapter 6: Experiment #5, Three-Phase AC Alternator

This final experiment in *Experiments with Renewable Energy* demonstrates the ability of permanent magnets and coils of wire to create electricity. In 1831, Michael Faraday demonstrated that by moving a magnet across a coil of wire, an electrical current was produced. Known as Faraday's Law of Magnetic Induction, this discovery made it possible to develop motors, generators and alternators. Our wind turbine works by rotating permanent magnets across fixed coils of wires, which induces electric currents in the wires. The type of electricity generated is known as three-phase.

6

UNDERSTANDING THREE-PHASE POWER

You may be most familiar with the type of power which comes from the wall socket in your house. This is single-phase, 110 VAC, 60 Hz power. That's 110 Volts, Alternating Current, at a frequency of 60 Hz, if you live in North America. For those in Europe, 50 Hz is common. One cycle of the single phase waveform goes through 360 angular degrees, from start to finish, as shown in Figure 6-1.

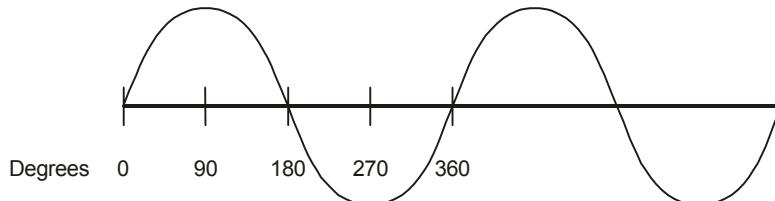


Figure 6-1: Single – Phase AC Output

Even though you may be most familiar with single phase 110 VAC, in fact, most of the electric power in the world is three-phase. The diagram in Figure 6-2 below shows three-phase power.

If you are thinking the diagram looks like 3 single-phase waveforms, each slightly shifted from one another, you're exactly right. This shift is the "phase". In this context, the term "phase" refers to the time displacement of electrical energy. Each separate sine wave is spaced, or phased, 120 angular degrees apart from one another. Phase 2 starts 120 degrees after Phase 1, and Phase 3 starts 120 degrees after Phase 2.

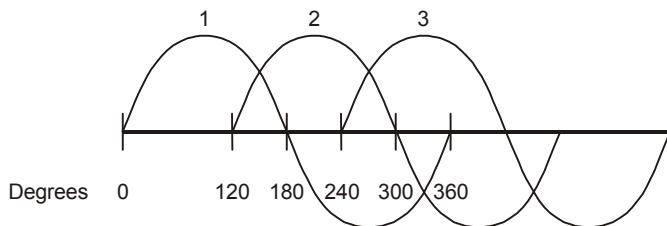


Figure 6-2: Three – Phase AC Output

Phase 2 starts 120 degrees after Phase 1, and Phase 3 starts 120 degrees after Phase 2. If you are wondering how all those phases travel on a single conductor, they don't. Typically, three-phase power is delivered to buildings through 4 wires. There is one wire for each of the three phases, plus a ground wire.

The concept of three-phase power was originally conceived by Nikola Tesla. Tesla proved that three-phase power was far superior to single-phase power. In a single-phase unit, the power falls to zero during each cycle. In a three-phase unit, however, it *never* drops to zero. No matter where you are in the cycle, one of the three phases is nearing its peak. As a result, the power delivered to the load is nearly the same at any instant. Furthermore, three-phase is typically 150% more efficient as compared with single-phase within the same power range, and in a three-phase unit the conductors need only be 75% the size of conductors for single-phase for the same power output. All these advantages make three-phase power efficient to produce and distribute. Let's find out why.

Creating Three-Phase Power

It turns out that creating three-phase power occurs quite naturally from geometry. With that in mind, take a look at the physical layout of the coils and magnets in our wind turbine, shown again in Figure 6-3. The turbine's black rotor disk has six magnets that are physically arranged 60 degrees apart from one another, and the north-south orientation of the poles alternates from one magnet to the next.

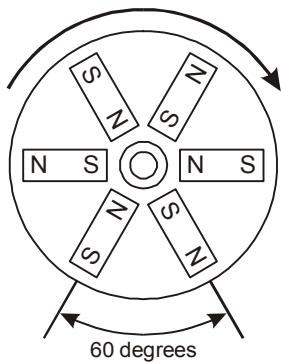


Figure 6-3
Wind Turbine Magnet
Arrangement

6

Inside the base, or stator, of the turbine are three coils of wire. In the schematic in Figure 6-4, the coils are wired such that they look like an upside-down letter "Y". This arrangement is called a "Wye" coil arrangement. It's also called a "Star" arrangement, as well, because it looks like a three-pointed star (but you'll have to use your imagination, somewhat). Notice that the coils are separated by 120 degrees, like the numbers on a clock face at the hours of 12, 4 and 8.

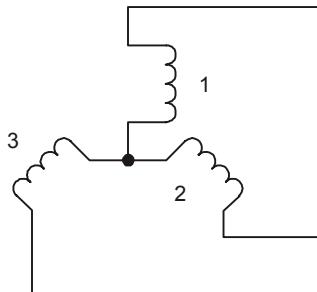


Figure 6-4
Wye Coil Arrangement

As the wind turbine spins, the magnets pass over each coil in turn. From the point of view of one coil, a changing magnetic field is seen. This changing magnetic field induces a changing current in the coil of wire. With three coils and six magnets, the resulting output waveform is three-phase AC.

Wye vs. Delta Windings

One can also wire the coils in another arrangement, known as a "Delta" configuration. This arrangement, and the resultant output waveform, is shown in the diagram below.

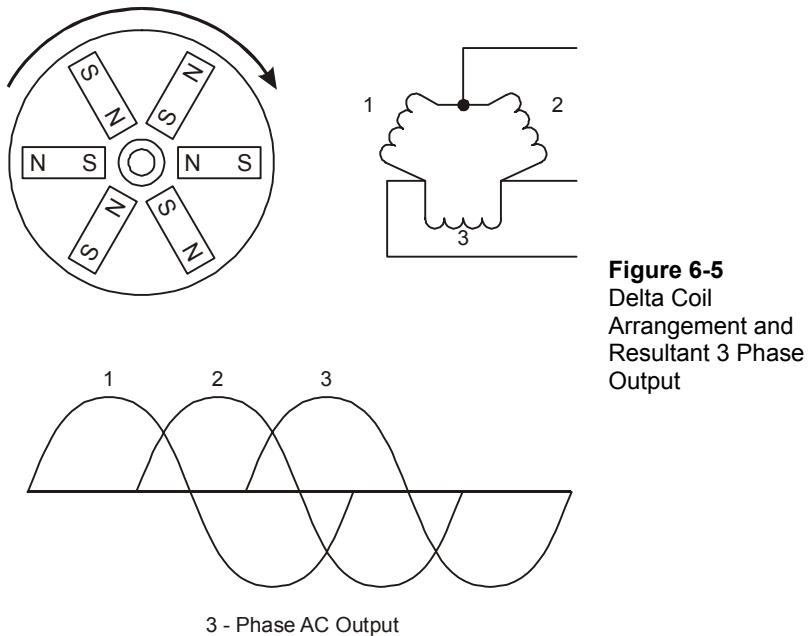


Figure 6-5
Delta Coil
Arrangement and
Resultant 3 Phase
Output

Comparing Wye and Delta

Surprisingly, the Delta waveforms are the same as the Wye waveforms, and the power output from both wiring arrangements are also the same. The major difference lies in the fact that the Wye arrangement generates more voltage while the Delta arrangement generates more current for a given rotational speed of the magnets.

	For a given rotational speed of the magnets:
	Wye coil arrangement: = More Voltage, Less Current
	Delta coil arrangement: = More Current, Less Voltage

Calculating Power Output

Power output is calculated from the product of the voltage and current. Or, in mathematical terms:

$$\text{power} = \text{voltage} * \text{current}$$



In electronics, voltage, current and power are usually denoted as:

$$\text{Voltage} = E \quad \text{Current} = I \quad \text{Power} = P \quad \text{Resistance} = R$$

Therefore, our equation becomes:

6

$$P = E * I$$

Calculating No-Load Power Output

For either coil arrangement, the voltage and current produced by a single winding can be measured. Once the voltage or current of one "phase" (coil) of the alternator is determined, the "sum of the three phases" is computed by multiplying by the square root of 3.

Wye Arrangement:

$$\text{volts of 1 phase} * \sqrt{3} = \text{volts of three phases} \rightarrow E_{\text{total}} = E_{\text{coil}} * \sqrt{3}$$

$$\text{current of 1 phase} = \text{current of three phases} \rightarrow I_{\text{total}} = I_{\text{coil}}$$

Delta Arrangement:

$$\text{current of 1 phase} * \sqrt{3} = \text{current of three phases} \rightarrow I_{\text{total}} = I_{\text{coil}} * \sqrt{3}$$

$$\text{volts of 1 phase} = \text{volts of three phases} \rightarrow E_{\text{total}} = E_{\text{coil}}$$

And for both arrangements:

$$P_{\text{total}} = E_{\text{total}} * I_{\text{total}}$$

Example 1: a single coil, turning at 600 RPM, generates 1 volt at 0.1 amp. Let's compute the total voltage, total current, and total power, for both Wye and Delta arrangements.

Solution:

Wye Arrangement:

$$E_{\text{total}} = E_{\text{coil}} * \sqrt{3} = E_{\text{coil}} * 1.732 = 1 * 1.732 = 1.732 \text{ volts}$$

$$I_{\text{total}} = I_{\text{coil}} = 0.1 \text{ amp}$$

$$P_{\text{total}} = E_{\text{total}} * I_{\text{total}} = 1.732 * 0.1 = 0.1732 \text{ watts}$$

Delta Arrangement:

$$E_{\text{total}} = E_{\text{coil}} = 1 \text{ volt}$$

$$I_{\text{total}} = I_{\text{coil}} * \sqrt{3} = I_{\text{coil}} * 1.732 = 0.1 * 1.732 \text{ amps} = 0.1732 \text{ amps}$$

$$P_{\text{total}} = E_{\text{total}} * I_{\text{total}} = 1 * 0.1732 = 0.1732 \text{ watts}$$

The Wye arrangement produces more voltage, 1.732 volts for the Wye versus 1.0 volts for the Delta. Conversely, the Delta produces more current, 0.1732 amps versus only 0.1 amp for the Wye arrangement.

However, the power output, 0.1732 watts, is exactly the same for both coil arrangements. This power output is called the "no load" output. A "load" is the electrical device to which electrical power is delivered. The load may be a light fixture, an electrical appliance, a battery, or something of this type. In the next section, we'll calculate the power output with a load.

Calculating Power Output with a Load

The internal resistance of the source (our alternator) must be taken into effect to calculate power output with a load. The key difference between the Wye and the Delta arrangements lies in the total resistance of the coil arrangement. ***The total resistance of the Wye arrangement is three times as large as the total resistance of the Delta arrangement!***

$$R_{\text{totalWye}} = 3 * R_{\text{totalDelta}}$$



Ohm's Law: $I = E/R$

Example 2:

Given:

Rotational speed of magnets = 1200 RPM
 voltage of one phase = E_{coil} = 3 volts
 Load = a 1.2 volt rechargeable battery
 $R_{totalWye}$ = 15 ohms
 $R_{totalDelta}$ = 5 ohms

Solve:

Calculate the total power when charging the battery for both the Wye and Delta coil arrangements.

6

Solution:

Wye Arrangement:

$$E_{total} = E_{coil} * \sqrt{3} = E_{coil} * 1.732 = 3.0 * 1.732 = 5.20 \text{ volts}$$

The total voltage generated is 5.2 volts, and 1.2 volts are consumed by the load, the 1.2 volt rechargeable battery. For current computations we have 5.20 volts being generated minus 1.2 volts being consumed by the battery load, leaving 4.00 volts.

$$\begin{aligned} I &= E / R = (5.2 - 1.2) \text{ volts} / 15 \text{ ohms} = 4.0 \text{ volts} / 15 \text{ ohms} = 0.27 \text{ amps} \\ P &= E * I = 4.0 * 0.27 = 1.07 \text{ watts} \end{aligned}$$

The total power for the Wye arrangement is 1.07 watts.

Delta Arrangement:

$$E_{total} = E_{coil} = 3.0 \text{ volts}$$

The total voltage generated is 3.0 volts, and 1.2 volts is consumed by the load, the 1.2 volt rechargeable battery. For current computations we have 3.0 volts being generated minus 1.2 volts being consumed by the battery load, leaving 1.8 volts.

$$\begin{aligned} I &= E / R = (3.0 - 1.2) \text{ volts} / 5 \text{ ohms} = 1.8 \text{ volts} / 5 \text{ ohms} = 0.36 \text{ amps} \\ P &= E * I = 1.8 * 0.36 = 0.65 \text{ watts} \end{aligned}$$

The total power for the Delta arrangement is 0.65 watts.

What a difference! The Wye arrangement produces almost twice the power of the Delta arrangement when charging a 1.2 volt battery.

Wye versus Delta Selection in the "Real World"

At this point you may be asking “Why use a Delta arrangement at all when the Star arrangement produces more power?” This is a very good question and here’s one answer.

In commercial wind-driven AC alternators (such as the ones described in the end of this chapter), the three-phase coils are automatically switched between Wye and Delta based mainly on the speed of the blades. When the wind turbine is just starting up, the coils are configured as a Wye, which allows the wind turbine to generate the same power at lower rpm’s. However, when the wind turbine comes up to speed, the extra voltage generated with the Wye arrangement is generally too high for the rated load, so the coils are automatically switched to a Delta arrangement to generate more current at a lower, but standardized voltage. Another reason for starting a wind turbine with a Wye winding setting is because it allows the turbine to come up to speed faster. Therefore both Wye and Delta coil arrangement have their individual applications in real-world power producing systems.

Wiring the Wye Coils

For our wind turbine, we will use the Wye coil arrangement. Each set of coils is color-coded by two wires of similar coloring, as shown in the table below.

Table 6-1: Color Code of Coils		
Coil 1	Blue	Blue-white
Coil 2	Green	Green-white
Coil 3	Orange	Orange-white

The colors are shown in the schematic in Figure 6-6. Pay particular attention to the *blue* and *blue-white* wire placement. This is not a mistake. It is necessary to “flip” the source-return orientation of this coil relative to the other two coils to achieve the proper three-phase operation.

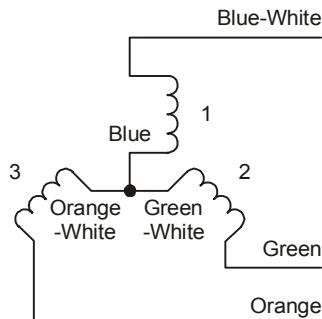


Figure 6-6
Wye Coil Arrangement Schematic

6

Rectifying Three Phase Power

Our wind turbine produces three phase alternating current (AC), which we'd like to rectify to direct current (DC). Recall the bridge rectifier circuit of Chapter 4, which used 4 diodes. In the bridge circuit, two diodes were active during the positive part of the waveform, and the other two diodes during the negative part.

By adding two more diodes, this circuit can be expanded to rectify all three phases from our wind turbine. With 6 diodes, the three-phase rectifier can rectify all three phases. Each phase uses 3 of the 6 diodes at any given time. These 3 diodes work just like the diodes in the bridge rectifier, one for the positive part of the waveform, the others for the negative part of the waveform. At different times, different diodes are used by different coils.

The diagrams below trace the current flow through the three-phase rectifier. Each frame freezes a moment during an interesting event (phase max positive/negative values) and analyzes how current is steered and rectified within the three-phase rectifier. Figure 6-7 shows the moment of maximum voltage generated at each phase, and Figure 6-8 shows the moment of minimum voltage generated at each phase.

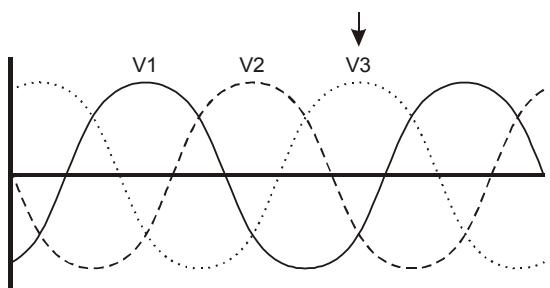
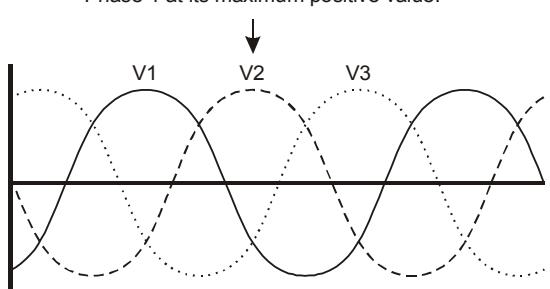
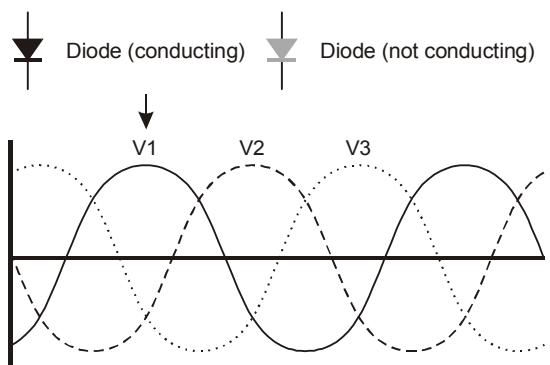
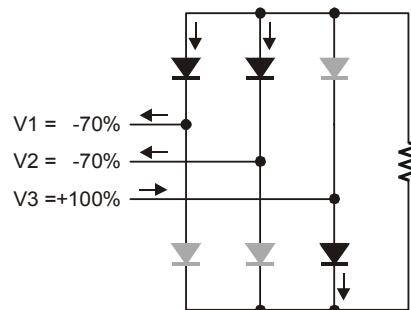
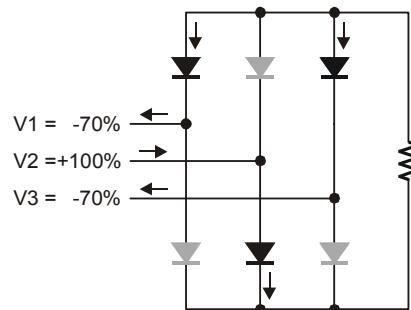
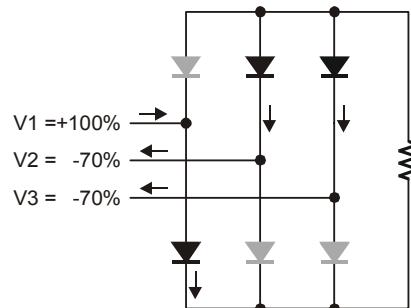


Figure 6-7

Wye Coil Arrangement Schematic
voltages are percentages of
maximum values.



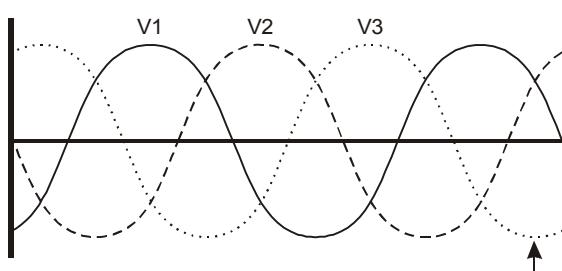
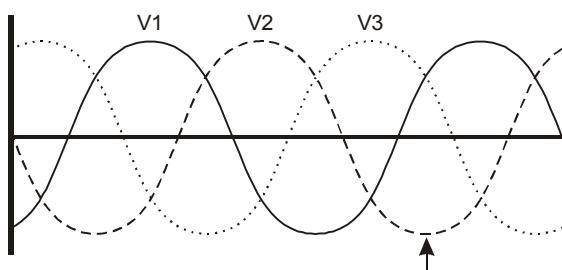
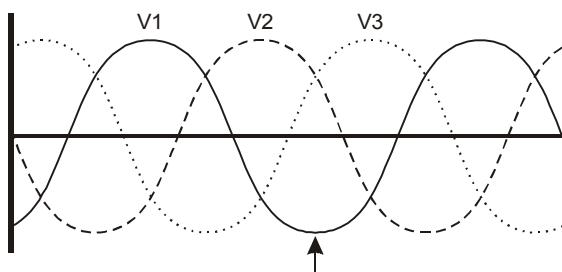
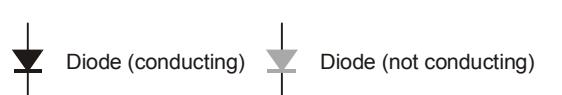
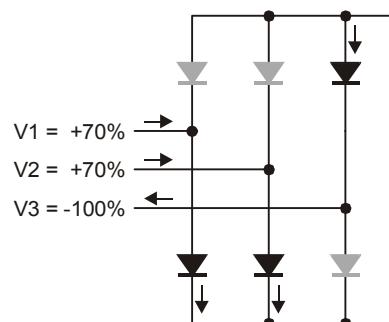
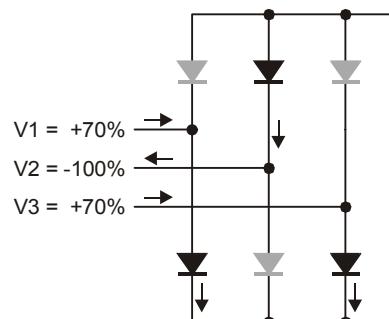
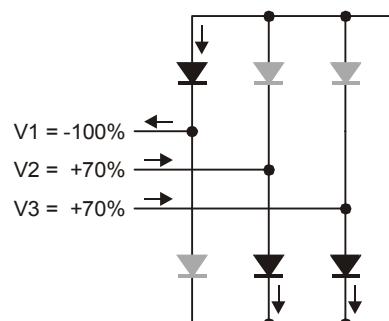


Figure 6-8
Wye Coil Arrangement Schematic
voltages are percentages of
maximum values.



6

As you can see from the images, regardless of the polarity and intensity of any phase's voltage, the steering abilities of the three-phase bridge rectifier allow current to flow in only one direction through the load.

ACTIVITY #1: BUILDING THE THREE-PHASE WYE CIRCUIT

Now to the fun part; that is, building our three-phase alternator. As before, we will be removing all the parts from the previous experiment except for the A/D converter circuit.

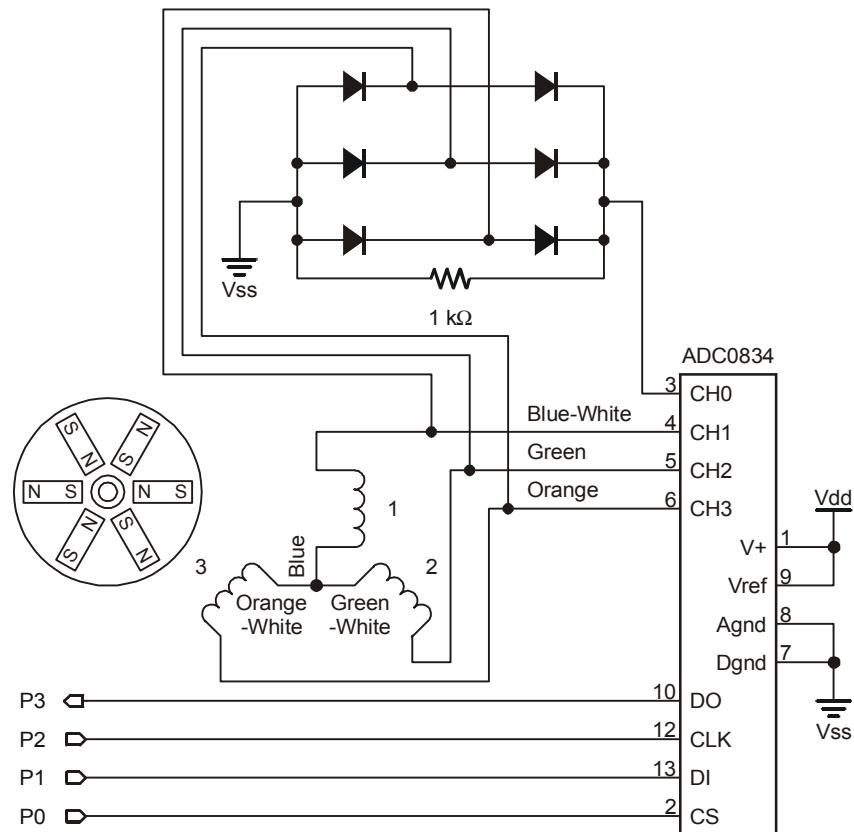
Parts Required

A/D Converter circuit plus...

- (6) 1N4148 diodes
- (1) 1 k Ω resistors
- (1) Fully constructed wind turbine
- (1) Table fan (optional, not supplied)

The complete circuit is shown in Figure 6-9 and Figure 6-10. The 3 coils are wired in a Wye arrangement. Each phase is directly connected to a channel on the A/D converter. Each phase is also connected to the three phase full wave rectifier circuit. The rectified output is wired to CH0 on the A/D converter. We've also placed a 1 k Ω load resistor across the rectified output to better stabilize the generated voltages.

- ✓ Remove all but the A/D converter circuit from your board.
- ✓ Build the circuit shown in Figure 6-9 and Figure 6-10.
- ✓ Double-check your wiring before moving on – this is a complex circuit!

**Figure 6-9:** Three-Phase Wye Schematic

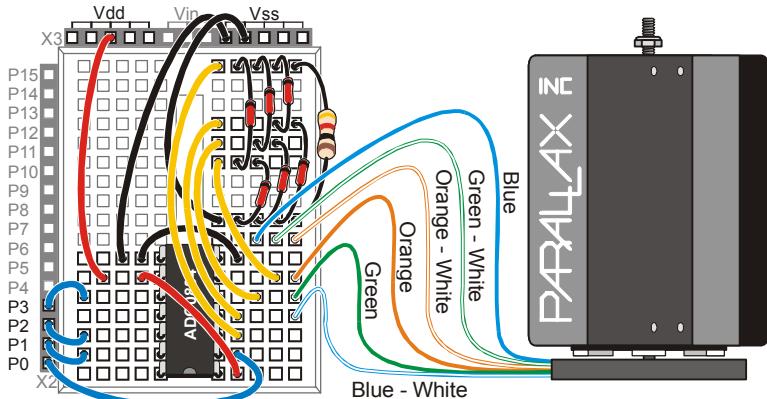


Figure 6-10: Three-Phase Wye Wiring Diagram

Sampling

We will be sampling all three phases of the output, plus the rectified sum of all three voltages, thus using all four channels of the A/D converter. We'll only sample the positive-going part of the Phase 1, 2, and 3 waveforms. Channel 4 of the A/D converter will sample the rectified output.

The sampling challenge arises because at the speed the wind turbine is rotating, sampling each of the four waveforms in real-time and sending them to StampPlot does not result in a good real-time display. What our program needs, then, is a way to sample each of the four A/D converter input signals for later display, similar to what we did in Experiment 4. The big difference, however, is that we now need to save four sample groups instead of just one – AND – we need a mechanism to keep the samples “in sync” with the actual real-time voltages that are being generated by the wind turbine.

To solve this problem, we'll take 10 samples of the waveform's voltage, as we did in Experiment 4, except now we'll do this for each of the four waveforms -- phase 3, phase 2, phase 1, and rectified output. This means we'll need to store a total of 40 samples, each sample taking one byte, for a total of 40 bytes. That's a large amount of data. Fortunately, the BASIC Stamp module provides a large memory bank, called EEPROM, for data storage, as well as built-in commands which make it easy to store and retrieve data from EEPROM memory.

RAM vs EEPROM

The BASIC Stamp's EEPROM can hold 2048 bytes, or 2 KB, of information. Part of the storage space, starting at address 2047 and growing toward address 0, is used to hold the PBASIC program. The unused space which remains can be used to store data, which builds from address 0 toward address 2047.

EEPROM is different from RAM (random access memory) variable storage in several respects:

- RAM is volatile memory, meaning that the memory contents are lost when power is removed
- EEPROM is non-volatile. The contents are maintained when the power is turned off.
- EEPROM takes more time to store a value, sometimes up to several milliseconds
- EEPROM can accept a finite number of write cycles, around 10 million writes. RAM has unlimited read/write capabilities
- The primary function of the EEPROM is to store programs; data is stored in the leftover space

For the BASIC Stamp 2 module, the maximum number of writes to its EEPROM is 10 million. That sounds like a lot, but when you are using EEPROM for data storage, it's important to make some rough calculations to see if the limit will be exceeded.

For example, consider a program that writes to a location in EEPROM once per second. How many days would it take to exceed 10 million writes?

$$1 \text{ write/sec} * 60 \text{ sec/min} * 60 \text{ min/hour} * 24 \text{ hours/day} = 86,400 \text{ writes/day}$$

$$10,000,000 \text{ writes} / 86,400 \text{ writes/day} = 116 \text{ days}$$

At one write per second, it would take nearly 116 days of continuous operation to exceed 10 million writes. Our Experiment 5 program writes to EEPROM once every 2.5 seconds, so we will not exceed the 10 million writes for over 290 days, or nearly ten months. Even though this is quite a long time, we can extend the functional life of the EEPROM even more by utilizing the following techniques. First, write to different parts of the EEPROM, instead of using the same locations over and over again. Second, only write samples when the wind turbine is actually turning.

So, here's the plan for sampling the four waveforms:

- Loop through each A/D channel, from CH3 to CH0
 - CH3 = phase 3, CH2 = phase 2, CH1 = phase 1, and CH0 = rectified output
- Wait and synch up with the positive cycle of phase 3 (arbitrarily chosen)
- Quickly take 10 samples of the waveform (`ch3`, `ch2`, `ch1`, or `ch0`)
 - Store each sample in variable storage (RAM) as it is obtained
- Transfer the 10 samples to EEPROM
- Repeat for the next A/D channel

This results in four individual sample groups that, taken together, we'll call "banks", stored in EEPROM. The next section shows how the banks are arranged.

Storing Samples in EEPROM

The diagram below shows each bank in EEPROM memory. The first ten bytes of EEPROM contain samples of the phase 3 voltage output, and the next six EEPROM bytes are unused. Although six EEPROM bytes per group are "wasted" by not writing to them, addressing the EEPROM space this way is easier to understand and handle programmatically.

The Phase 2, Phase1, and Rectified Output Samples, ten each, follow similarly. Each of the four sample groups within each bank is separated by sixteen bytes, or \$10 in hexadecimal. Each bank is separated from one another by 64 bytes, or \$40 in hexadecimal. Each time a new set of samples is taken, the data is written to a new bank. This spreads the writes to the EEPROM out over a larger area, thus conserving the writeable lifecycle of each EEPROM byte.

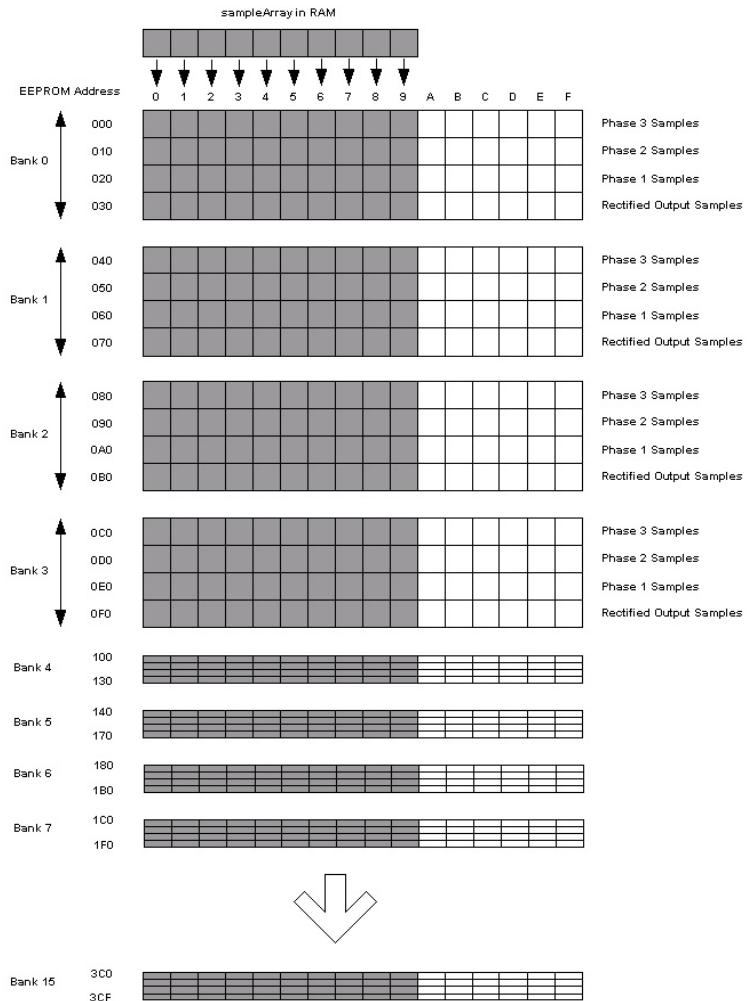


Figure 6-11: EEPROM Memory Allocation For Variables

All addresses are in hexadecimal. Shaded areas indicate sample storage, and unshaded areas are unused.

Displaying Samples using StampPlot

Even though we took 10 samples of Phase 3, followed by 10 samples of Phase 2, and so on, we want to display the samples as if we captured all four A/D channels at once. So we'll read one sample from each sample group (Phase 3, Phase 2, Phase 1, and Rectified Output) out of EEPROM. Then we'll plot those four samples. The resultant plot will look like a four-channel oscilloscope, displaying four traces at once. Here's the plan:

- Loop through each of the 10 samples, from 0 to 9
- Extract 4 pieces of data from EEPROM using 4 separate reads
 - Read one sample from Phase 3, one from Phase 2, one from Phase 1, and one from Rectified Output
- Store the values into the variables `ch3`, `ch2`, `ch1`, `ch0`
- Call the `Plot_it` subroutine, which will plot the values of `ch0` through `ch4`
- Repeat for the next sample

Now that we have an idea of how the program will work, let's see how it looks in PBASIC.

ACTIVITY #2: PROGRAMMING THE THREE-PHASE AC ALTERNATOR

As usual, we will be adding to our previous program, and commenting out the routines not currently needed. (Alternatively, you can download the source code from www.parallax.com)

- ✓ In the BASIC Stamp Editor, open Half-FullWaveRectifier.bs2
- ✓ Rename and save as AlternatorPhase1.bs2.
- ✓ Update the title section as follows:

```
' Experiments with Renewable Energy v1.0 - AlternatorPhase1.bs2
' Generate 3-Phase power from wind-powered alternator, then plot
' only Phase 1.
' {$STAMP BS2}
' {$PBASIC 2.5}
```

Now add this to the Declarations section

```
' ----- For Experiment 5: Three-Phase AC Alternator -----
Plotting    CON      1          ' Program is plotting
Sampling    CON      0          ' Program is taking samples
epromAddr  VAR      dataPtr 'Word   ' EEPROM address pointer
```

```

eepromAddr = $0000           ' Initial eepromAddr value
plotCount    VAR      codePtr 'Nib
mode         VAR      Bit      ' Number plotted so far
mode = Sampling               ' Plot mode or sample mode
                                ' Initially will be sampling

```

In the Main Routine section, comment out the first 4 **GOSUB** commands, like this:

```

' -----
' Main Routine
' -----
Main:
DO
'   GOSUB Exp_1          ' Programmable Battery Charger
'   GOSUB Exp_2          ' Dueling Solar Cells
'   GOSUB Exp_3          ' Solar Cell Sun Tracker
'   GOSUB Exp_4          ' Half&Full Wave Rectification
'   GOSUB Exp_5          ' Three-Phase AC Alternator
'   GOSUB Plot_It         ' Plot Data w/ StampPlot Pro
LOOP

```

6

In the Experiment 5 section, add the explanatory comments and program code

```

' -----
' Experiment 5: Three-Phase AC Alternator
' -----
'
' Enter from power-on or reset with eepromAddr = $000,
' mode = Sampling, and plotCount = 0
'
' This routine will be called once each time through Main, and right
' after this routine, Plot_It will be called.
'
' If mode = Sampling, then take a set of 40 samples from all
' 4 A/D converter inputs. 10 samples per channel.
'
' If mode = Plotting, then retrieve one set of 4 samples from
' EEPROM. Jump out of Exp_5... the one set will get plotted when
' Plot_It is called. This will happen 10 times to get all 10 sets
' of 4 samples plotted.
'
' Sampling:
'
' For each of the three phases and the rectified output,
' take samples as follows:
'
' - Increment the eepromAddr to the next sub-block (except for phase 3)
' - Set the A/D converter to sample phase 3

```

```

- Sync up to the positive peak of phase 3 (except for phase 3, itself)
- Switch the A/D to sample the next phase or the rectified output
  (again, except for phase 3)
- Gather 10 samples of the converted values in sampleArray (RAM)
- Transfer the 10 samples from sampleArray to EEPROM

Plotting:
For each of the three phases and the rectified output,
plot samples as follows:

- Increment plotCount for the next time around
- Read each sample from EEPROM into the appropriate
  variable (ch3, ch2, ch2, ch0)
- Exit the program and let the Plot_It routine graph the four samples
- When we've plotted all the samples, increment the eepromAddr
  to the next sub-block, change mode to Sample the next time through,
  and reset plotCount to zero

Note: If the eepromAddr is greater than address $0300, reset it to zero -
we don't want to clobber the program memory

-----
Exp_5:
SELECT mode
CASE Plotting
  plotCount = plotCount + 1
  GOSUB Exp_5_Retrieve_For_Plottting
  IF (plotCount = 9) THEN
    mode = Sampling
    plotCount = 0
    GOSUB Exp_5_Inc_EepromAddr
  ENDIF
CASE Sampling
  GOSUB Exp_5_Sample_Phase_3
  GOSUB Exp_5_Sample_Phase_2
  GOSUB Exp_5_Sample_Phase_1
  GOSUB Exp_5_Sample_Rectified_Output
  GOSUB Exp_5_Retrieve_For_Plottting
  mode = Plotting
ENDSELECT

Exp_5_End:
RETURN

-----
' Subroutines for Experiment 5
'-----

Exp_5_Sample_Phase_3:
a2dMuxId = A2dMuxId3
' Set the A/D to convert Channel 3
```

```

GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak
GOSUB Exp_5_Take_Samples            ' Take samples of phase 3
GOSUB Exp_5_Write_Samples           ' Write samples to EEPROM

Exp_5_Sample_Phase_3_End:
RETURN

'-----

Exp_5_Sample_Phase_2:
GOSUB Exp_5_Inc_EepromAddr          ' Increment pointer to EEPROM
a2dMuxId = A2dMuxId3               ' Set the A/D to convert Channel 3
                                    ' sync up to phase 3
GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak

a2dMuxId = A2dMuxId2               ' Switch the A/D to convert Channel 2

GOSUB Exp_5_Take_Samples            ' Take samples of phase 2
GOSUB Exp_5_Write_Samples           ' Write samples to EEPROM

Exp_5_Sample_Phase_2_End:
RETURN

'-----

Exp_5_Sample_Phase_1:
GOSUB Exp_5_Inc_EepromAddr          ' Increment pointer to EEPROM
a2dMuxId = A2dMuxId3               ' Set the A/D to convert Channel 3
                                    ' sync up to phase 3
GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak

a2dMuxId = A2dMuxId1               ' Switch the A/D to convert Channel 1

GOSUB Exp_5_Take_Samples            ' Take samples of phase 1
GOSUB Exp_5_Write_Samples           ' Write samples to EEPROM

Exp_5_Sample_Phase_1_End:
RETURN

'-----

Exp_5_Sample_Rectified_Output:
GOSUB Exp_5_Inc_EepromAddr          ' Increment pointer to EEPROM
a2dMuxId = A2dMuxId3               ' Set the A/D to convert Channel 3
                                    ' sync up to phase 3
GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak

```

```

a2dMuxId = A2dMuxId0                                ' Switch the A/D to convert Channel 0

GOSUB Exp_5_Take_Samples                            ' Take samples of the rectified wave
GOSUB Exp_5_Write_Samples                          ' Write samples to EEPROM

Exp_5_Sample_Rectified_OutputEnd:
    RETURN

'-----
' Retrieve 4 samples, one for each channel.  This will be called 10 times,
' to get the total "bank", which is 4 sets of 10 samples per channel.
Exp_5_Retrieve_For_Plottting:
    ' READ (eepromAddr - $30 + plotCount), ch3 ' Read phase 3 into ch3
    ' READ (eepromAddr - $20 + plotCount), ch2 ' Read phase 2 into ch2
    READ (eepromAddr - $10 + plotCount), ch1 ' Read phase 1 into ch1
    ' READ (eepromAddr - $00 + plotCount), ch0 ' Read rectified val into ch0

Exp_5_Retrieve_For_Plottting_End:
    RETURN

'-----

Exp_5_Wait_For_Zero:                                ' Loop until the rectified wave
    DO                                              ' is below 0.10 volts
        GOSUB A2D
        LOOP UNTIL (a2dResult <= 5)                 ' (5 x 0.02 volts = 0.10 volts)

Exp_5_Wait_For_Zero_End:
    RETURN

'-----

Exp_5_Wait_For_Next_Peak:                           ' Loop until the rectified wave
    DO                                              ' is non-zero
        GOSUB A2D
        LOOP UNTIL (a2dResult => 5)

Exp_5_Wait_For_Next_Peak_End:
    RETURN

'-----

Exp_5_Take_Samples:                                 ' Enter with A2D channel set
    FOR sampleCount = 0 TO 9                         ' Acquire 10 samples of the
        GOSUB A2D                                     ' wave as quickly as possible
        sampleArray(sampleCount) = a2dResult
    NEXT

Exp_5_Take_Samples_End:
    RETURN

```

```

' -----
Exp_5_Write_Samples:                                ' Enter with eepromAddr set
    FOR sampleCount = 0 TO 9                      ' Store the wave samples to
    EEPROM
        WRITE (eepromAddr + sampleCount), sampleArray(sampleCount)
    NEXT

Exp_5_Write_Samples_End:
    RETURN

' -----
Exp_5_EepromAddr_To_Zero:                          ' Initialize the eepromAddr
    eepromAddr = $0000                            ' to the bottom of EEPROM
    RETURN

' -----
Exp_5_eepromAddr_To_Zero_End:
    RETURN

' -----
Exp_5_Inc_EepromAddr:
    eepromAddr = eepromAddr + 16
    IF (eepromAddr >= $0300) THEN
        GOSUB Exp_5_eepromAddr_To_Zero
    ENDIF
    ' Add 16 to move to next bank
    ' Make sure not to overwrite
    ' our PBASIC program which
    ' starts around $0300

Exp_5_Inc_EepromAddr_End:
    RETURN

'----- End Exp_5 Subroutines -----

```

6

✓ If you have hand-entered all these changes, save your work!

ACTIVITY #3: HOW THE THREE-PHASE AC ALTERNATOR PROGRAM WORKS

This experiment samples, then plots, then increments the EEPROM address pointer to the next bank. The sampling can be done in one fell swoop, but the plotting is done by 10 separate calls to the `Plot_It` routine. We'll sample 10 data points for each channel, for a total of 40 data points in each bank. Then, we'll retrieve 4 samples at a time, one for each channel, and plot them, 10 times in all. For this reason, we need to keep track of whether Experiment 5 is currently taking samples, or plotting. A variable, `mode`, will help keep track of this state. Two constants, `Plotting` and `Sampling`, equal to 1 and 0, respectively, are introduced to abstract the state of the program. If `mode` is set to

Sampling, the program is in the middle of plotting our 10 samples. If mode is equal to **Plotting**, then the current bank of samples has been plotted, and it is time to take some more samples. Our program initializes the value of **mode** to **Sampling**, to force the program to take samples the first time through.

```
' ----- For Experiment 5: Three-Phase AC Alternator -----
Plotting      CON      1
Sampling      CON      0
EEPROMAddr   VAR      dataPtr 'Word
EEPROMAddr = $0000
plotCount    VAR      codePtr 'Nib
mode         VAR      Bit
mode = Sampling
```

A **SELECT...CASE** statement examines the **mode** variable. The first time through, it is equal to **Sampling**, so the program will take samples. Each phase gets its own subroutine to perform the sampling.

```
Exp_5:
SELECT mode
CASE Plotting
    plotCount = plotCount + 1
    GOSUB Exp_5_Retrieve_For_Plottting
    IF (plotCount = 9) THEN
        mode = Sampling
        plotCount = 0
        GOSUB Exp_5_Inc_EEPROMAddr
    ENDIF
CASE Sampling
    GOSUB Exp_5_Sample_Phase_3
    GOSUB Exp_5_Sample_Phase_2
    GOSUB Exp_5_Sample_Phase_1
    GOSUB Exp_5_Sample_Rectified_Output
    GOSUB Exp_5_Retrieve_For_Plottting
    mode = Plotting
ENDSELECT

Exp_5_End:
RETURN
```

Exp_5_Sample_Phase_3 is where we start taking samples of Phase 3 and store them into EEPROM. The first instruction activates CH3 of the A/D converter. Then, through a

series of subroutine calls, we'll wait for the zero point, then the next peak, as shown in Figure 6-5.

```
Exp_5_Sample_Phase_3:
    a2dMuxId = A2dMuxId3
    GOSUB Exp_5_Wait_For_Zero
    GOSUB Exp_5_Wait_For_Next_Peak
    GOSUB Exp_5_Take_Samples
    GOSUB Exp_5_Write_Samples
    RETURN
```

The purpose of the `Exp_5_Wait_For_Zero` subroutine is to “sync up” to the next positive cycle of phase 3 (as we have arbitrarily named it). We did much the same in Experiment 4 by first waiting for the half wave rectified voltage output to go to zero or just below 5 ($5 = 5 * 0.02\text{volts/sample} = 0.10 \text{ volts}$). This routine does exactly the same thing, however we have converted it into a subroutine since other parts of our program will also make calls to it. Once it finds that the phase 3 wave is at zero volts, it exits (returns) to the next instruction, which is a call to `Exp_Wait_For_Next_Peak`.

```
Exp_5_Wait_For_Zero:
    DO
        GOSUB A2D
    LOOP UNTIL (a2dResult <= 5)
    RETURN
```

As its name implies, `Exp_Wait_For_Next_Peak` waits until the phase 3 voltage output just begins to start its peak voltage cycle. With subsequent calls to the A/D converter the `a2dResult` is tested against the value of 5, which is really 0.10 volts as explained earlier. Once past this level we can be assured that the wave is beginning to peak. It is at this point that we want to begin taking voltage samples.

```
Exp_5_Wait_For_Next_Peak:
    DO
        GOSUB A2D
    LOOP UNTIL (a2dResult => 5)
    RETURN
```

The `Exp_5_Take_Samples` routine quickly takes ten converted voltage samples and stores them in the `sampleArray` in RAM, just as we did in Experiment 4.

```
Exp_5_Take_Samples:
    FOR sampleCount = 0 TO 9
```

```

        GOSUB A2D
        sampleArray(sampleCount) = a2dResult
NEXT

Exp_5_Take_Samples_End:
RETURN

```

Exp_5_Write_Samples is where the program transfers the ten samples in **sampleArray** to EEPROM. Notice how it does this. Within the **FOR...NEXT** loop the **sampleCount** starts at zero and increments to nine. Using the **WRITE** instruction we store the sample in **sampleArray** indexed by **sampleCount** to EEPROM at the address that is the sum of **eepromAddr** plus **sampleCount**. In this one **WRITE** instruction **sampleCount** is used twice as an index; once to acquire the sample in **sampleArray** and again to write the sample into EEPROM.

```

Exp_5_Write_Samples:
FOR sampleCount = 0 TO 9
    WRITE (eepromAddr + sampleCount), sampleArray(sampleCount)
NEXT

Exp_5_Write_Samples_End:
RETURN

```

Exp_5_Sample_Phase_2 is quite similar to the previous Phase 3 sample code with two important exceptions. The first difference is a call is to **Exp_5_Inc_eepromAddr**.

```

Exp_5_Sample_Phase_2:
GOSUB Exp_5_Inc_EepromAddr

```

Recall from Figure 6-11 that each of the four groups within each bank is separated by sixteen bytes, or \$10 hexadecimal. The **Exp_5_Inc_eepromAddr** subroutine effectively moves the EEPROM address from the last entry of the Phase 3 samples to the first entry of Phase 2 samples.

```

Exp_5_Inc_eepromAddr:
eepromAddr = eepromAddr + 16
IF (eepromAddr >= $0300) THEN
    GOSUB Exp_5_EepromAddr_To_Zero
ENDIF

Exp_5_Inc_EepromAddr_End:
RETURN

```



We don't want to overwrite our PBASIC code! Our program code occupies from about address 960 (\$3C0) to 2047 (\$7FF). Therefore, the subroutine tests the absolute address of the `eepromAddr` pointer to determine if it will encroach into the area of EEPROM where our program code is stored. We test it at 768 (\$300), which is sufficiently away from the last program byte. If we've just incremented it at or above this value, we must reset the `eepromAddr` pointer back to 0000. In effect, we will be using EEPROM from 0000 to 768 (\$000 → \$300) for storing data samples.

Once this is done, set the A/D converter to sample phase 3 (CH3) and then make the two calls to “sync up” to its next positive-going part.

6

```
a2dMuxId = A2dMuxId3

GOSUB Exp_5_Wait_For_Zero
GOSUB Exp_5_Wait_For_Next_Peak
```

Now here is where things get a little different. The next instruction switches the A/D converter to sample phase 2 (CH2) rather than phase 3. This is so that we can sample the phase 2 waveform “in sync with” the phase 3 waveform, which we are arbitrarily using as the reference phase. Since each phase is equally separated from the other two, we could have chosen phase 1 or phase 2 as the reference. However, we simply chose to use phase 3, instead. The next two instructions again take ten samples of phase 2 and write them to EEPROM as the next group in this bank.

```
a2dMuxId = A2dMuxId2

GOSUB Exp_5_Take_Samples
GOSUB Exp_5_Write_Samples
```

The code that samples phase 1 and the rectified output is a carbon copy of the code for phase 2, with the exception that the phase 1 code sets the A/D converter to sample CH1 and the rectified output code samples CH0.

```
Exp_5_Sample_Phase_1:
GOSUB Exp_5_Inc_EepromAddr
a2dMuxId = A2dMuxId3

GOSUB Exp_5_Wait_For_Zero
GOSUB Exp_5_Wait_For_Next_Peak
```

```

a2dMuxId = A2dMuxId1

GOSUB Exp_5_Take_Samples
GOSUB Exp_5_Write_Samples
RETURN

' -----
Exp_5_Sample_Rectified_Output:
    GOSUB Exp_5_Inc_EepromAddr
    a2dMuxId = A2dMuxId3

    GOSUB Exp_5_Wait_For_Zero
    GOSUB Exp_5_Wait_For_Next_Peak

    a2dMuxId = A2dMuxId0

    GOSUB Exp_5_Take_Samples
    GOSUB Exp_5_Write_Samples
    RETURN

```

Well, we're done with the sampling tasks, at least for the first set of samples. Our program has successfully captured four sets of A/D converter samples from each channel, stored them in RAM (`sampleArray`), then transferred them to EEPROM in bank 0 on a one-at-a-time basis.

Since we are done with sampling, the last statement in our `CASE` statement switched `mode` back to `Plotting`. This completes the `SELECT...CASE` statement, and control returns to the `Main` program.

When `Main` jumps back to `Exp_5`, the `SELECT...CASE` will drop into the `Plotting` case. A variable named `PlotCount` will keep track of how many sets of samples have been plotted. We increment the `plotCount` so that it points to the next set of four samples.

```
plotCount = plotCount + 1
```

Execution now jumps to the `Exp_5_Retrieve_For_Plottting` subroutine.

Now comes the part of the program that extracts four channels of data from EEPROM for eventual display by the `Plot_It` routine. By commenting out individual `READ` commands, we can choose which channels to plot individually, or together. (Here we have only channel 1 plotting).

```

Exp_5_Retrieve_For_Plottting:
  'READ (eepromAddr - $30 + plotCount), ch3
  'READ (eepromAddr - $20 + plotCount), ch2
  READ (eepromAddr - $10 + plotCount), ch1
  'READ (eepromAddr - $00 + plotCount), ch0

```

The program code can execute up to four separate **READ** commands to EEPROM to extract four bytes of sample data stored there. The key to doing this is knowing what address to use. We started at the top of the program with **eepromAddr = \$000**, and each pass through the code groups **Exp_5_Sample_Phase_2**, **Exp_5_Sample_Phase1** and **Exp_5_Sample_Rectified_Output** incremented **eepromAddr** by 16 or \$10 hexadecimal. Therefore, at this point, **eepromAddr** is at address \$030 hexadecimal. Therefore, to read back the data we stored in EEPROM we first need to “subtract” the group address from **eepromAddr**. Then to get to the proper byte of data in each group, we need to “add” the value of **plotCount**.

Look at the first **READ** instruction. Here we subtract \$30 from **eepromAddr** thus making the effective address \$000 (\$030 - \$030=\$000). Then we add the current value of **plotCount**, which is currently at 0. So our effective address for the first **READ** instruction is \$000. The effective addresses of the other **READ** instructions are \$010, \$020 and \$030, respectively. Refer, again, to Figure 6-11 to see what we’ve done.

In reading all four samples into their respective channel variables (**ch3 - ch0**) the **Plot_It** routine will plot the four values the next time through.

The next time **Main** calls **Exp_5** the **SELECT...CASE** will increment **plotCount**, then branch to **Exp_5_Retrieve_For_Plottting** again. Only this time **plotCount** is at 1 and our effective addresses for **ch3** through **ch0** are as follows:

ch3	\$001	=	$(\$030 - \$030 + 1)$
ch2	\$011	=	$(\$030 - \$020 + 1)$
ch1	\$021	=	$(\$030 - \$010 + 1)$
ch0	\$031	=	$(\$030 - \$000 + 1)$

As **plotCount** is incremented with each call to **Exp_5_Retrieve_For_Plottting** it eventually reaches the value of 9. This is where the **IF...THEN** statement in the **CASE** statement will be true. After switching **mode** back to **sampling**, we’ll jump to the

Exp_5_Inc_eepromAddr subroutine and increment **eepromAddr** to the next bank of EEPROM data storage area. In this case it will increment from \$030 to \$040.

```
IF (plotCount = 9) THEN
    mode = Sampling
    plotCount = 0
    GOSUB Exp_5_Inc_EepromAddr
ENDIF
```

Then we'll take ten more sets of samples per channel, storing them into Bank 1, which starts at \$0040. This process of sampling, plotting, and incrementing to the next bank repeats over and over.

Of course, we don't have unlimited storage, and we don't want to clobber our stored program, so we have to reset the **eepromAddress** back to Bank 0 at some point. This is taken care of in **Exp_5_Inc_eepromAddr**.

```
Exp_5_Inc_EepromAddr:
    eepromAddr = eepromAddr + $10
    IF (eepromAddr >= $0300) THEN
        GOSUB Exp_5_EepromAddr_To_Zero
    ENDIF
```

The **IF...THEN** statement tests for our EEPROM address limit, which we have chosen as \$300 or 768. If the program increments **eepromAddr** over \$300 a call to the **Exp_5_EEPROM_To_Zero** subroutine is executed, where the **eepromAddr** is reset to \$000 as follows:

```
Exp_5_EepromAddr_To_Zero:
    eepromAddr = $0000

Exp_5_EepromAddr_To_Zero_End:
    RETURN
```

What all of this address manipulation has accomplished has been to preserve the life of our EEPROM. The program has spread out the writes to EEPROM over as wide an area as possible. If you left the power to the Board of Education on twenty-four hours a day – AND – if the wind turbine were spinning all of that time, the life of our EEPROM would be 10 years.

$$10 \text{ months/bank} * 12 \text{ banks} = 120 \text{ months / 12 months / year} = 10 \text{ years}$$

Remember at the very beginning of our program explanation we mentioned that EEPROM would only be written when the wind turbine was spinning? Well, take a look at the code under `Exp_5_Sample_Phase_3` again.

```
Exp_5_Sample_Phase_3:
    a2dMuxId = A2dMuxId3
    GOSUB Exp_5_Wait_For_Zero
    GOSUB Exp_5_Wait_For_Next_Peak
    GOSUB Exp_5_Take_Samples
    GOSUB Exp_5_Write_Samples
```

6

Whether the wind turbine is spinning, or not, the first two instructions will be executed. However the third instruction, the call to `Exp_5_Wait_For_Next_Peak`, will never return and will be caught in an endless loop waiting for the voltage to go from zero to non-zero; something that usually won't happen if the wind turbine is NOT spinning. Therefore, our program will be caught in this subroutine until the wind turbine begins spinning once again. The result is that while the program is stuck in this loop it will not write to EEPROM, thus extending its functional life even further.

Your Turn: Sampling and Displaying the Three-Phase Voltage Outputs

If you look at the code under the label `Exp_5_Retrieve_For_Plottting`, you'll notice that only one of the four `READ` statements are not "patched out", and only the `READ` of phase 1 is active. We will build up to the entire four-channel plot on an additive basis by taking out the comments ('') from each `READ` statement one-at-a-time. If we showed you the entire four-channel plot at this time, you would become very confused (as we were when we first saw it). So we'll lead up to the four-channel display slowly beginning with just phase 1.

- ✓ Anchor your wind turbine and arrange your table fan so that the turbine blades are turning steadily and briskly.
- ✓ Run the program AlternatorPhase1.bs2.
- ✓ Close the Debug Terminal after noting which COM port it is using.
- ✓ Open StampPlot Pro by clicking on Start → Programs → Parallax Inc → StampPlot → Experiments with Renewable Energy → sic_ewre_exp_45.spm.
- ✓ Set the COM port in StampPlot to the same one that was being used by the Debug Terminal.

- ✓ Click on the Connect button, and make sure the Enable Plotting box is checked.

Figure 6-12 shows what you should be seeing on your computer, a plot of only phase 1.

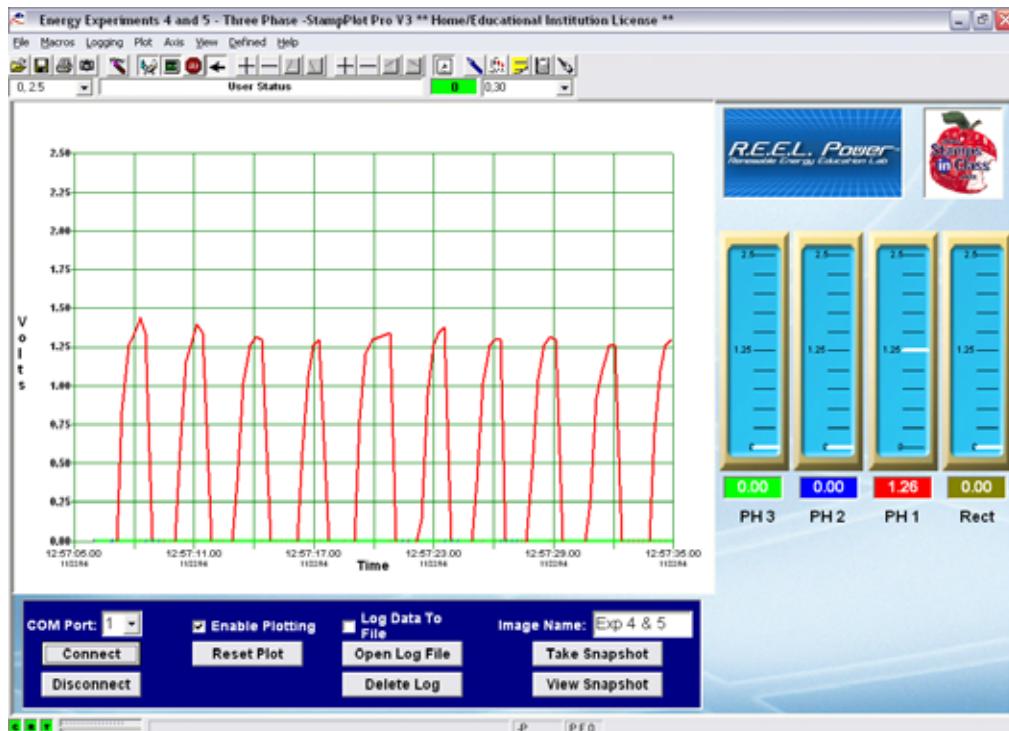


Figure 6-12: StampPlot Plot of Phase 1

By comparison, Figure 6-13 shows what one full cycle of phase 1 looks like as displayed on an oscilloscope. Notice how the entire waveform is shown. You don't see the negative-going portion on StampPlot since our program only samples above zero volts.

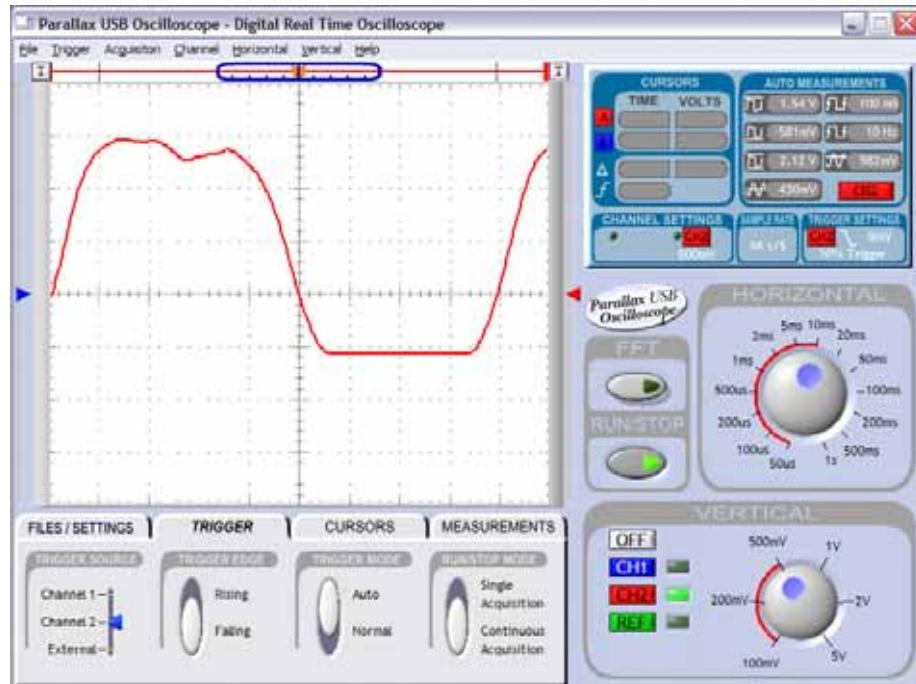


Figure 6-13: Oscilloscope Plot of Phase 1

To display both phase 1 and phase 2 together you will need to uncomment the `READ` instructions for phase 2. (Or, you can download `AlternatorPhase1-2.bs2`.)

- ✓ Disable StampPlot by clicking on the Disconnect button, then bring up the BASIC Stamp Editor.
- ✓ In the `Exp_5_Retrieve_For_Plottting` subroutine, delete the apostrophe in front of the second `READ` command:

```
Exp_5_Retrieve_For_Plottting:
  'READ (epromAddr - $30 + plotCount), ch3
  READ (epromAddr - $20 + plotCount), ch2
  READ (epromAddr - $10 + plotCount), ch1
  'READ (epromAddr - $00 + plotCount), ch0
```

- ✓ Re-run the program.

- ✓ Now close the Debug Terminal and bring up the StampPlot screen back up.
- ✓ Re-enable it by clicking on the Connect button, at the same time making sure that the Enable Plotting box is also checked.

You should see something like that shown in Figure 6-14.

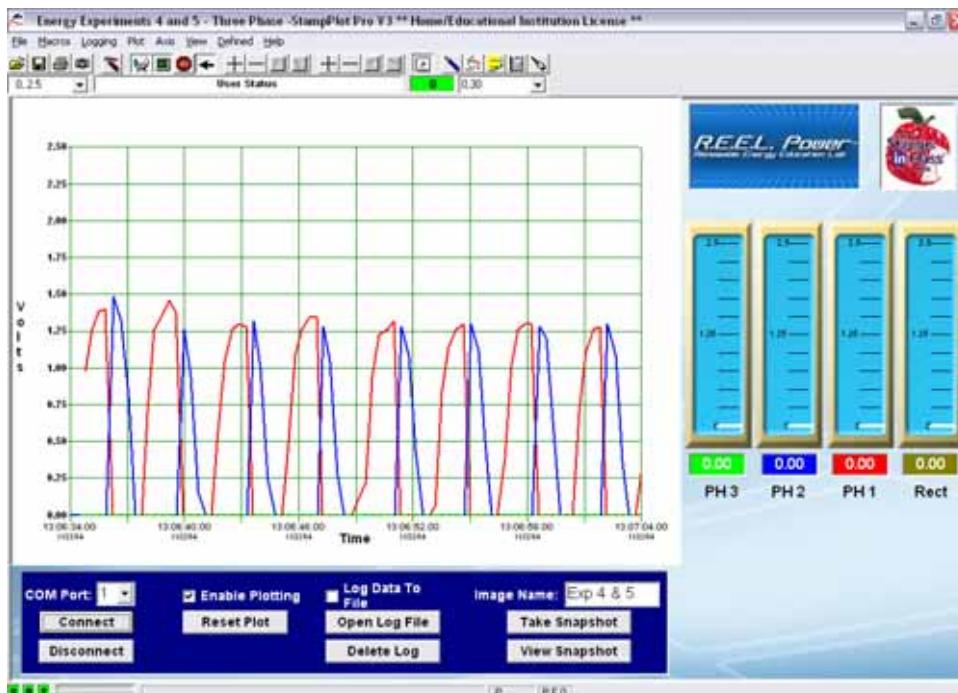


Figure 6-14: StampPlot Plot of Phase 1 and Phase 2

Once again, by comparison Figure 6-15 shows what one cycle of each phase looks like as displayed on an oscilloscope.

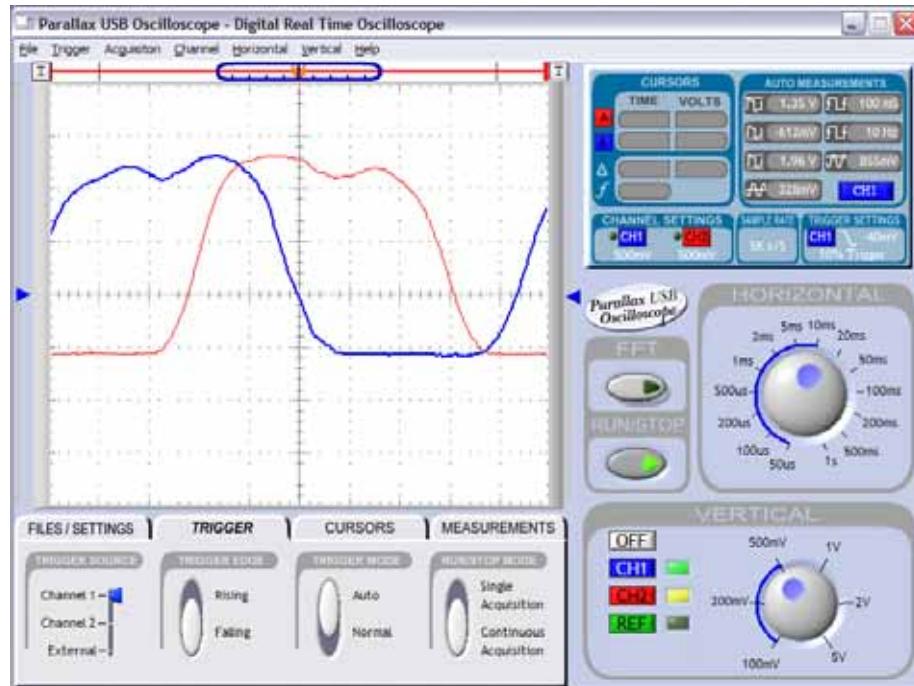


Figure 6-15: StampPlot Plot of Phase 1 and Phase 2

To display phase 1 and phase 3 together you will need to uncomment the `READ` instructions for phase 3, and comment out phase 2 again. (Or, you can download `AlternatorPhase1-3.bs2`.)

- ✓ Disable StampPlot by clicking on the Disconnect button, then bring up the BASIC Stamp Editor.
- ✓ In the `Exp_5_Retrieve_For_Plottin`g subroutine, replace the apostrophe in front of the second `READ` command, and delete it from the first one:

```
Exp_5_Retrieve_For_Plottting:  
    READ (eepromAddr - $30 + plotCount), ch3  
    'READ (eepromAddr - $20 + plotCount), ch2  
    READ (eepromAddr - $10 + plotCount), ch1  
    'READ (eepromAddr - $00 + plotCount), ch0
```

- ✓ Re-run the program.
- ✓ Now close the Debug Terminal and bring up the StampPlot screen back up.
- ✓ Re-enable it by clicking on the Connect button, at the same time making sure that the Enable Plotting box is also checked.

You should see something like that shown in Figure 6-16.

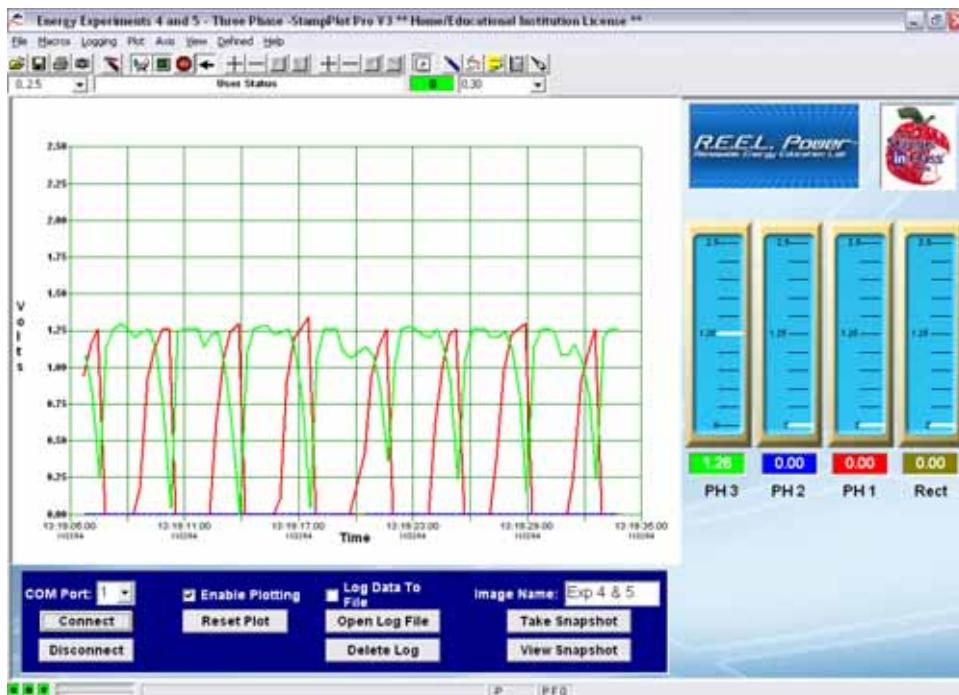


Figure 6-16: StampPlot Plot of Phase 1 and Phase 3

Note the tiny dip on the positive part of the wave for Phase 3. This is the “signature” of our particular three-phase AC alternator. Yours will certainly be different due to the particular way your (hand-wound) coils were arranged in the stator and the way your magnets were (hand) placed on the rotor.



Phase 3 not dropping below 0.5 volts? If your plot of Phase 3 doesn't drop down below 0.5 or 0.25 volts, try making your turbine turn faster. This was experienced by our testing staff (:-) and a faster turbine gave the better-looking plot shown above.

6

Figure 6-23 shows what one cycle of each phase looks like as displayed on an oscilloscope.

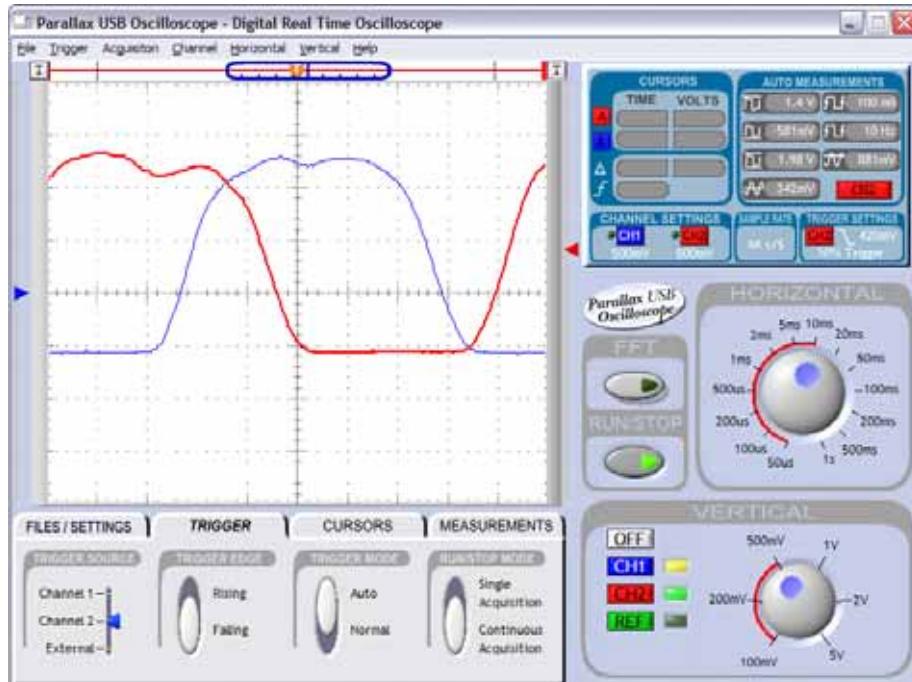


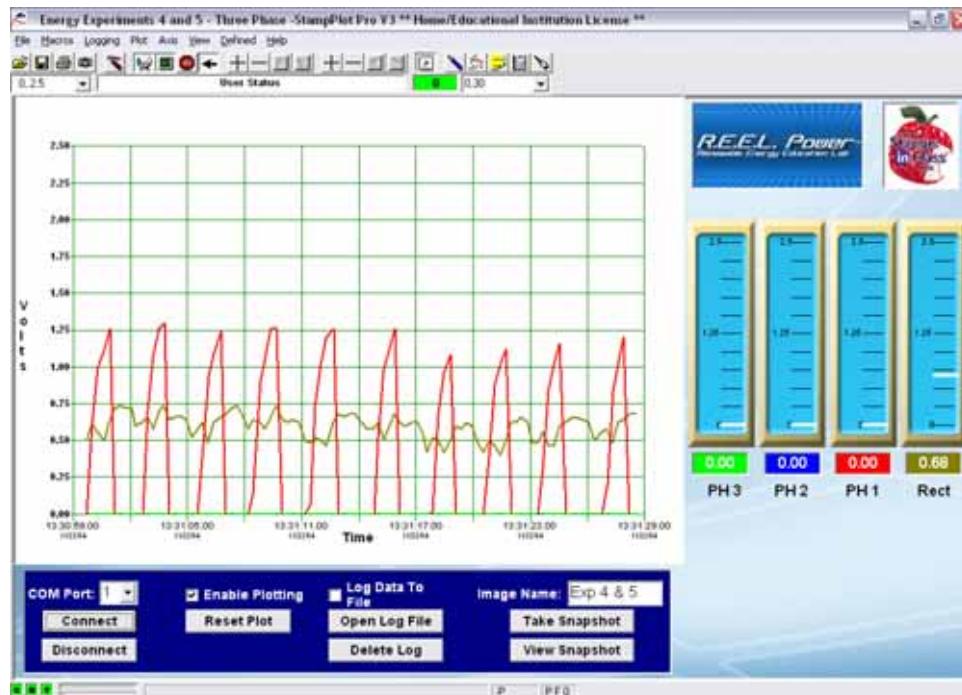
Figure 6-17: Oscilloscope Plot of Phase 1 and Phase 3

Before displaying all three phases and the rectified output, let's first look at phase 1 and the rectified output by themselves. You will need to uncomment the last **READ** instruction, and comment out phase 3 again. (Or, you can download AlternatorPhase1-R.bs2.)

- ✓ Disconnect in StampPlot, then bring up the BASIC Stamp Editor.
- ✓ In the **Exp_5_Retrieve_For_Plottting** subroutine, place the apostrophe in front of the first and second **READ** commands, and uncomment the third and fourth as shown:

```
Exp_5_Retrieve_For_Plottting:  
    'READ (eepromAddr - $30 + plotCount), ch3  
    'READ (eepromAddr - $20 + plotCount), ch2  
    READ (eepromAddr - $10 + plotCount), ch1  
    READ (eepromAddr - $00 + plotCount), ch0
```

- ✓ Re-run the program, and close the Debug Terminal.
- ✓ Bring up StampPlot again.
- ✓ Click on Connect, making sure that the Enable Plotting box is also checked.



6

Figure 6-18: StampPlot Plot of Phase 1 and Rectified Output

You should see something like that shown in Figure 6-18. Now compare this to the oscilloscope plot of the same thing in Figure 6-19.

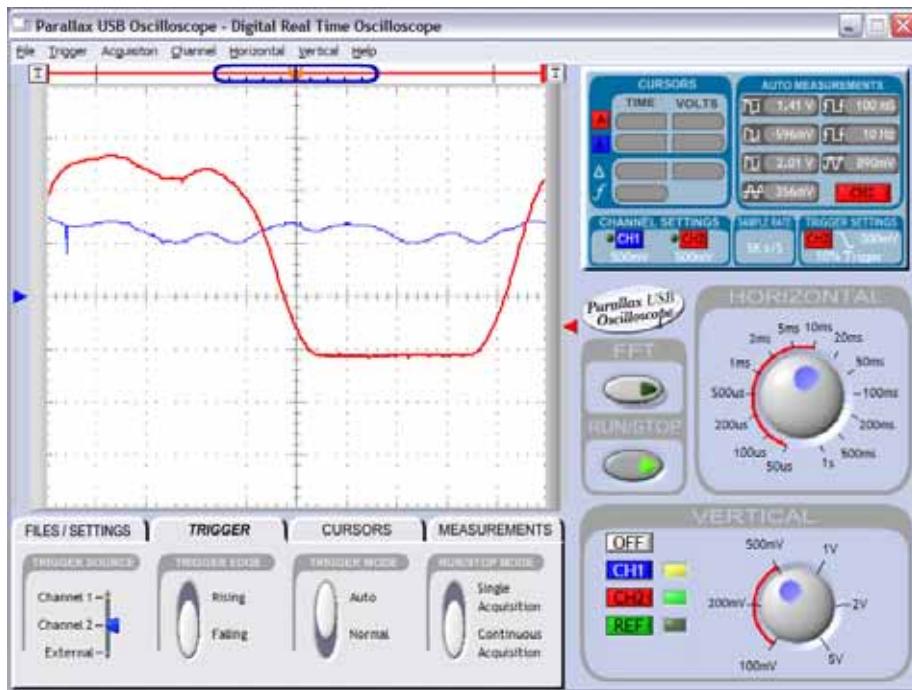
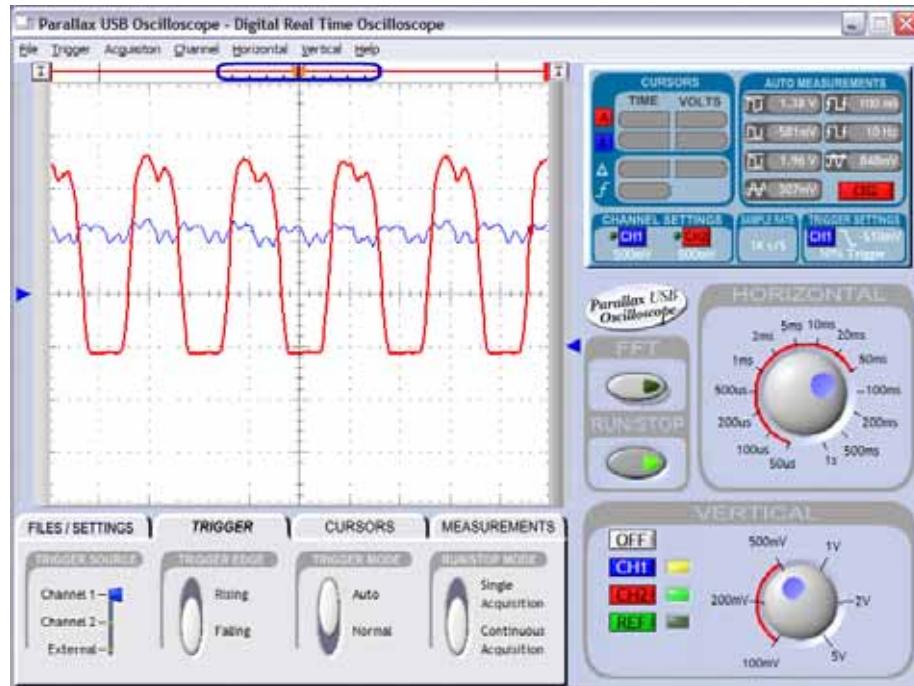


Figure 6-19: Oscilloscope Plot of Phase 1 and Rectified Output



6

Figure 6-20: Oscilloscope Plot of Phase 1 and Rectified Output (different time scale)

Now for the grand finale – displaying all three phases and the rectified output in one plot.

- ✓ Disable StampPlot by clicking on the Disconnect button, then bring up the BASIC Stamp Editor.
- ✓ In the `Exp_5_Retrieve_For_Plottin`g subroutine, delete all apostrophes in front of the `READ` commands, so they will all be executed. (Or, download AlternatorPhase1-2-3-R.bs2.)

```
Exp_5_Retrieve_For_Plottin:
  READ (eepromAddr - $30 + plotCount), ch3
  READ (eepromAddr - $20 + plotCount), ch2
  READ (eepromAddr - $10 + plotCount), ch1
  READ (eepromAddr - $00 + plotCount), ch0
```

- ✓ Re-run the program.

- ✓ Now close the Debug Terminal and bring up the StampPlot screen back up.
- ✓ Re-enable it by clicking on the Connect button, at the same time making sure that the Enable Plotting box is also checked.

You should see something like that shown in Figure 6-18.

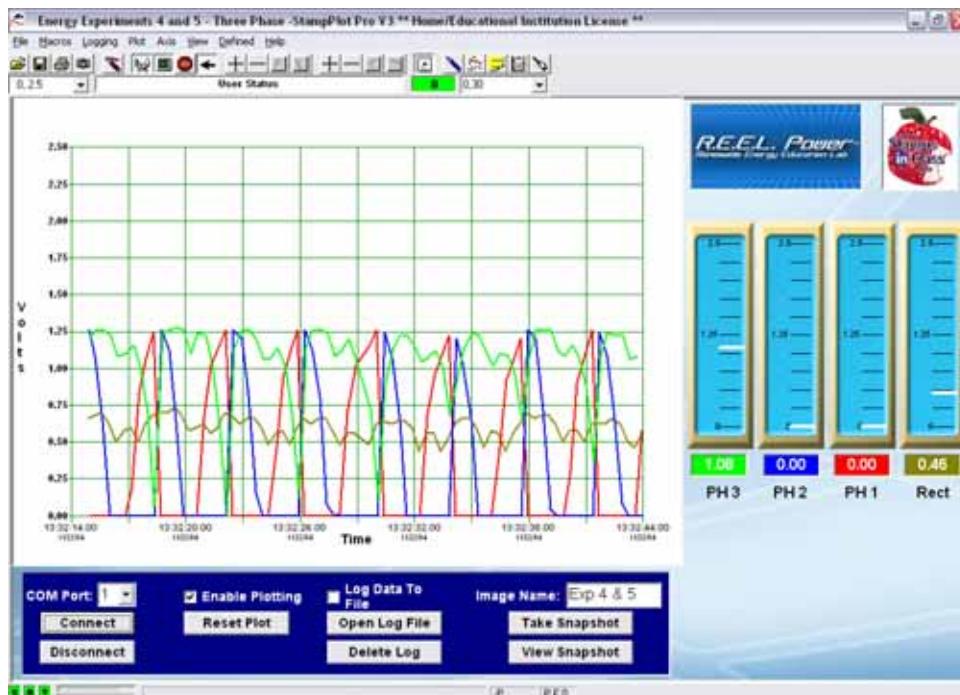


Figure 6-21: StampPlot Plot of All Three Phases and Rectified Output

Since we don't have a 4-channel oscilloscope we can't show you a comparable display. However, we hope you now understand why we lead up to the four-channel display by taking you there one step at a time. You can see the confusion that would have resulted if we would not have done so.



In terms of comparing the StampPlot plots with the oscilloscope plots, you may have noticed that the bottom, zero voltage, part of the waveforms is shown on the oscilloscope plots. The reason you don't see the negative-going portion on StampPlot is because our program only samples above zero volts. Nevertheless, the StampPlot plots are still quite representative of the actual waveforms.

6

SUMMARY AND APPLICATIONS

The application of three-phase power began in the late 19th Century with power plants being one of the first adopters of this new technology.

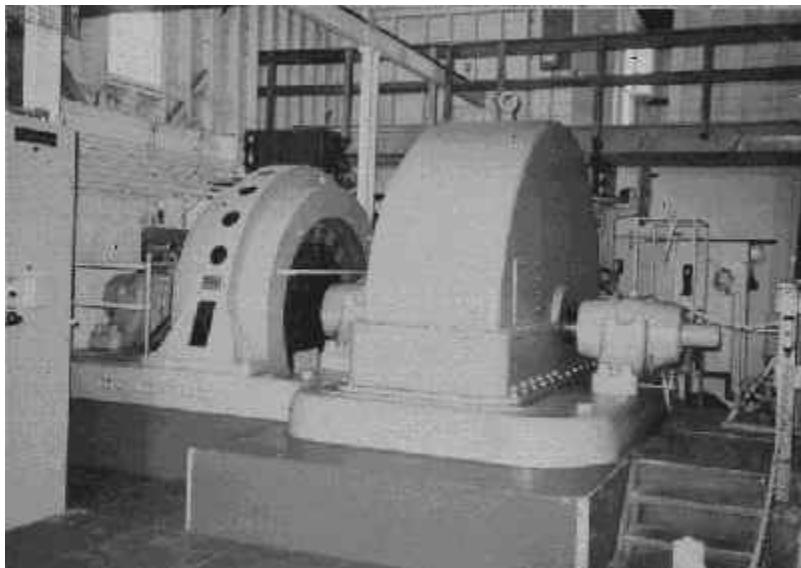


Figure 6-22: The Mill Creek Hydroelectric Plant

Photo courtesy Southern California Edison

Built by the Redlands Electric Light and Power Company located in Redlands, California, the Mill Creek hydroelectric generating plant began operating on 7 September 1893. It was the first commercial use of three-phase alternating current generators in the United States. The power from this 250 kW generator was transmitted 7.5 miles to the city of Redlands at 2400 Volts. The electricity was used for both lighting for the city of Redlands and a nearby icehouse. The people of Redlands enthusiastically welcomed the

new energy source and within 2½ years an additional generator had to be installed to accommodate the growing demand.

Before the Mill Creek No.1 Power House was built, the generation of electricity had generally been by DC (direct current) generators, which meant that there were limitations on how far electricity could be delivered. On the other hand, the San Antonio Canyon Powerhouse used 1-phase AC (alternating current) sources. Single-phase motors of this era required elaborate means to start and synchronize the motors. The success of the three-phase generators at the Mill Creek No. 1 was apparent, for these original generators were used until 1934. Although the original units have been replaced, this plant is still in operation to this day. Today, more than 100 years after Mill Creek's completion, three-phase generators are still the primary form of power generation around the world.

THREE-PHASE POWER IN THE REAL WORLD

Three-phase power has many uses from appliances such as refrigerators, to welders, transformers, ovens, computers, CNC mills, oil rig pumps and just about anything else you can imagine being powered by electricity. Three-phase by its very nature is a much more smooth form of electricity as compared with single-phase power. It is this more consistent electrical power that allows machines that use it to run more efficiently and last many years longer than their relative machines running on the other phases. Some applications are able to work with three-phase power in ways that would not work on single-phase at all.

The alternator in your family car is a three-phase device with a rectifier circuit nearly the same as the one you built in this experiment, except its diodes are far more powerful. Commercial versions of wind driven alternators also use three-phase power exclusively including electronic circuitry to automatically switch between Wye and Delta windings depending upon the wind conditions. And, of course, the high-voltage power lines you see marching across the landscape are three-phase as well.

Then there are wide varieties of power electronic circuits that are used to control three-phase motors like the one here made by Anacon Systems, Inc. Electronic modules like their EagleDrive 3 motor controller have taken over many of the tasks that mechanical systems performed in the past to keep motors running smoothly and at top efficiencies.



Figure 6-23
The EagleDrive 3 motor by Anacon Systems, Inc.

Photo © Anacon Systems Inc., used with permission.

There are also applications that demand three-phase power from single-phase, which is what this unit from Isomatic does. Isomatic manufactures single phase to three phase converters using electronic switching and an internal motor-generator with 600% overload capability. These units are suitable for farms, workshops, etc, where the site lacks three-phase supply and within large buildings where all 3 phases are not close to the equipment location.



Figure 6-24: An Isomatic Single-phase to three-phase Converter

Photo © Isomatic; used with permission.

With all the three-phase power generating equipment available, how about something to measure it? That's exactly what the 355 Harmonics Analyzer, a hand-held three-phase

power and harmonics analyzer manufactured by Dranetz-BMI does. This instrument measures true three-phase harmonics and power. An ideal tool for industrial plant engineers and utility power quality engineers, it lets you make quick, accurate judgments about overheated neutral conductors, transformer derating, and equipment specification.



Figure 6-25
The Dranetz-BMI 355
Harmonics Analyzer

*Photo © Dranetz-BMI;
used with permission.*

GENERATING ELECTRICITY USING WIND POWER

Experiment 5 examined the fundamentals of three-phase power. The information presented here deals with a principal application of this technology, namely wind power.



Figure 6-26: Power-generating Windmills

Photo courtesy of Paul Gipe

Using the wind to generate clean, efficient and “cheap” electricity has been the dream of many people and industries for at least 100 years. However, until very recently the ability to achieve these three goals was elusive, mainly because fossil fuels were so plentiful and relatively inexpensive. Now with the threat of “global warming” along with the attendant pollution caused by the burning of fossil fuels, renewable energy technologies are making inroads into providing commercially attractive power sources.

While solar power is a great choice for localized electricity generation, wind power is certainly the choice for grid-based power generation. Modern wind generators, and the wind farms that host them, can provide large cities like San Francisco and even entire rural states with sufficient power to operate homes and businesses, alike. It has been correctly claimed that if our Midwest states like North and South Dakota were to build mega-wind farms, this flat, barren and constantly windy territory could become the “Saudi Arabia of the United States” in terms of grid-based wind energy generation.

For sometime now counties such as Germany, Spain and Denmark (in order of the percentage of use of wind power) have supplemented their existing fossil and nuclear power generation by the use of wind power. The United States is behind in similar programs, however it is on a course to catch up quickly. We will briefly explore these and other examples of wind power usage beginning with the primary types of wind turbines in use today.

Wind Energy Technologies

While old-fashioned windmills are still seen in many rural areas for pumping water, modern wind turbines are divided into two major categories: horizontal axis turbines and vertical axis turbines.



Figure 6-27
Horizontal Axis Turbine

The New Mexico Wind Energy Center has 136 turbines, 320 feet high. Producing 204 megawatts, this is the third largest wind power plant in the world.

Photo by Michael McDiarmid, Courtesy of New Mexico Energy Conservation and Management Division

Horizontal axis turbines like the one pictured in Figure 6-27 are the most common turbine configuration used today. They consist of a tall tower, atop which sits a fan-like rotor that faces into or away from the wind. Most horizontal axis turbines built today have two or three blades, although some have fewer or more blades. The newer, larger and more powerful horizontal axis turbines have blades that are longer than those of a 747-jet airliner! Plus, they are the most efficient in terms of energy production, to date.

A vertical axis wind turbine (VAWT) is likely to fall into one of two major categories: Savonius and Darrieus, however neither turbine type is in wide use today.

The Darrieus turbine was invented in France in the 1920s. Often described as looking like an eggbeater, this vertical axis turbine has vertical blades that rotate into and out of the wind. Using aerodynamic lift, these turbines can capture more energy than drag devices. The Giromill and cyclo-turbine are variants on the Darrieus turbine.



Figure 6-28
A Darrieus-type Turbine

Photo Courtesy of Darrel Dodge

6

The basic “theoretical” advantages of a vertical axis machine are that the generator and gearbox can be placed on the ground and do not require a tower. Plus, you do not need a mechanism to turn the blades into the wind as you do with a horizontal axis machine. That said, the disadvantages of a Darrieus turbine far outweigh its advantages. First of all, wind speeds are much lower close to the ground so the overall power generating efficiency is not very impressive. Plus it needs a push to get started and must also have a wide network of guy wires to hold it in place, which occupy valuable farm land that can’t be used for grazing or planting. Finally, when the main bearings or other parts need maintenance the whole machine must be torn down. That’s why the Darrieus turbine picture here has been out of service for many years, rusting away on a hill above the St. Lawrence Seaway in Canada.

First invented in Finland, the Savonius turbine is S-shaped if viewed from above. This drag-type VAWT turns relatively slowly, but yields a high torque. It is useful for grinding grain, pumping water, and many other tasks, but its slow rotational speeds are not good for generating electricity.



Figure 6-29
A Savonius-type Wind
Turbine

*Photo courtesy of Lance
Turner*

Some practical electrical applications for Savonius wind turbines still exist, however, like the one pictured here. When there is a need for a small amount of electricity and solar panels are not practical due to climate or lots of trees, etc. a home-built Savonius wind turbine will do nicely. This one in particular provides the user with enough electrical power to open and close a gate to the driveway entrance as well as power safety lights.

Wind Farms

There are many today who think of harvesting the wind as if it were a crop. In reality, wind is a like a crop since it has value to the wind farmer who cultivates it as well as to the consumer this is serviced he services with the electricity the wind farm generates. Unlike editable crops, however, the wind crop does not need to be tilled, fertilized or spayed with pesticide to grow healthy and strong. At the same time there is inherent risk in establishing and maintaining an economically healthy wind farm. Here are some examples of new and potentially successful wind farms.

Native American Wind Farms

The Rosebud Sioux Tribe Wind Turbine Project is the first large-scale Native American-owned and operated wind turbine in the country.



Figure 6-30
The Rosebud Sioux
Tribe Wind Turbine
Project

*Photo courtesy of the
Intertribal Council On
Utility Policy*

6

Located on the Rosebud Sioux reservation in south-central South Dakota, the project had been stalled for over a year as funding and energy sales issues were addressed. With a long term contract with NativeEnergy, complementing a short term sales option retained by the Tribe for a portion of the wind turbine, the Rosebud Sioux Tribe proceeded with final construction financing and placed the turbine order. The project was completed in February 2003. Since it has proved to be successful, many more turbines will be constructed and put into operation, and funds are being raised for Phase 2 (<http://www.nativeenergy.com>). Our Native Americans may well begin to enjoy the status and economic rewards of becoming our nation's largest energy producer, competing with the likes of nuclear and coal fired power plants that now dot the same landscape.

Wind Farms as a Tourist Attraction

“Windmill Tours” of Palm Springs, California has hit upon a unique and fun way to learn about wind power. As the ad says ... “Travel through a forest of towering windmills on electric-powered vehicles. Feel the energy as the giant blades WHOOSH overhead. Your skilled guide will take you inside this working wind farm comprised of turbines modernized to efficiently contribute to a cleaner and safer environment. As you travel along the 90-minute adventure, you realize the environmentally friendly power propelling experience was created by the air you are breathing”.



Figure 6-31
A Stop at Windmill Tours in Palm Springs, California

*Photo courtesy of Windmill Tours
Palm Springs*

Wind Farms At Sea – United States

One of the more unique and controversial new planned wind farms could be located on our East Coast off Nantucket Island, which is near Cape Cod, Massachusetts.

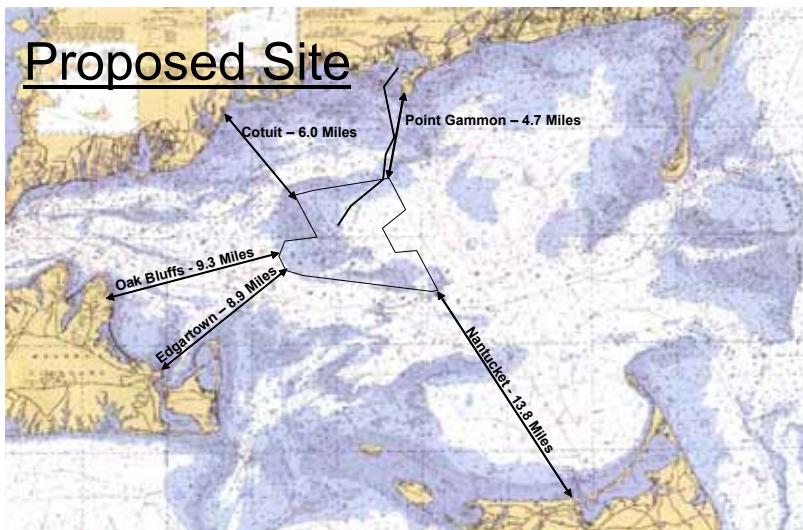


Figure 6-32
Proposed Location of Offshore Wind Farm

*Photo
©courtesy of Cape Wind Associates LLC*

Cape Wind Associates of Boston, has proposed building the country's first offshore wind farm in Nantucket Sound. The 130 wind turbines would be placed in a 24-square-mile area on Horseshoe Shoal, and some of the 417-foot tall turbines would be visible from Nantucket, Martha's Vineyard and various points along the Cape's south coast from Mashpee to Dennis. As popular as these areas are to the summer vacation tourist trade, let alone the local year-round residents, it will be interesting to see if this plan succeeds.

Wind Farms At Sea – Denmark

In the summer months of 2002 the world's largest offshore wind farm on the Danish west coast was built and put into operation. The sea-based wind farm is sited 14 to 20 kilometers into the North Sea, west of Blåvands Huk, and represents the first phase of a large-scale Danish effort to produce non-polluting electricity from these offshore wind turbines. The "Horns Rev" project, as it is called, has a total capacity of 4000 megawatts and must be established in full before 2030.



Figure 6-33: The Horns Rev Danish Wind Farm

Photo © Elsam A/S, used with permission

Historically, wind power capacity has been developed on land, but it has become increasingly difficult to obtain the required permits for turbine sites. With its available coastline, interest has been directed toward coastal areas with shallow water depths between 15 and 50 feet that have the possibility of locating the turbines far enough away from the coast that they are visually neutral, something the Nantucket Sound project is criticized for ignoring.

ENECO's Spar-WARP™ Wind Machine

The Spar-WARP™ wind machine developed by ENECO can produce *and store* clean, safe electricity offshore anywhere that there is sufficient wind of between 13 to 17 mph.

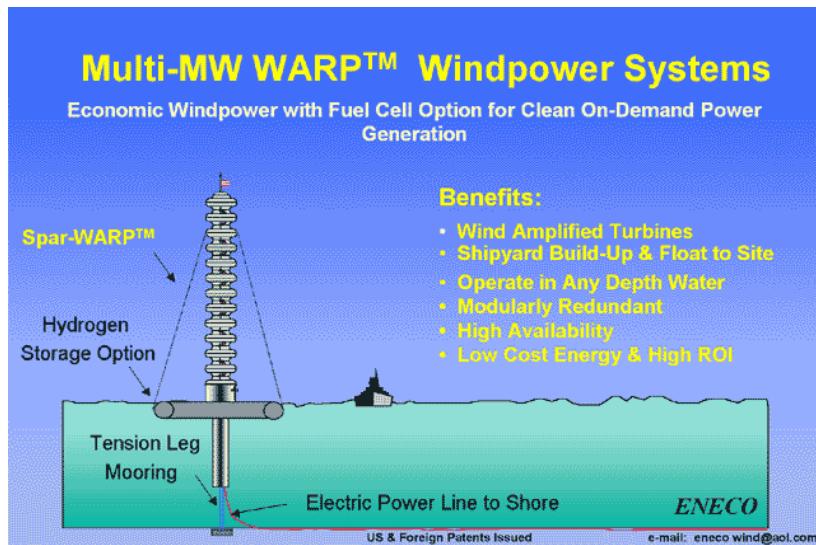


Figure 6-34: The Spar-WARP™ Wind Machine

Graphic © ENECO, used with permission

This outstanding example of a commercially viable source of renewable energy has the ability to store the power it generates as hydrogen gas. The WARP, which stands for Wind Amplified Rotor Platform, is designed to further amplify wind that it receives by a factor of 50 to 80 percent! Without the conventional propeller blades the multi-tiered donut-like rotors create a spinning effect that produces this amplification. There are many potential applications for ENECO's patented Spar-WARP™ technology, including powering offshore platforms directly as well as providing onshore power with a cable between it and the shoreline.

The Future of Wind Power

As we said at the beginning of this section, wind power has a great potential future for large grid-powered wind farm applications. Even though you can find many examples of

small and medium sized wind machines that turn three-phase alternators, the real future for wind power is in large installations. The economics of wind power versus fossil fuel are growing narrower thus making wind power a likely candidate for future power plants. Only politics and resistance to change still keep this technology from becoming commonplace. However, this is today's situation. Tomorrow is a different story, entirely.

Wind power will require both power electronics and engineers skilled in the art of wind-driven turbines to continue on its successful path to acceptance and reality. Hopefully, what you've learned here and in Experiment 5 will help to influence your interest in this amazing technology.

6

Credits

The following individuals and organizations are recognized here for their contributions to this section, whose material was reprinted with permission:

Paul Gipe - <http://www.wind-works.org>

ENECO - <http://www.warp-eneco.com>

Lance Turner - Making a simple Savonius Wind Turbine

<http://www.ata.org.au/articles/70byosav.htm>

U.S. Department of Energy - <http://www.eere.energy.gov>

Wind Mill Tours - <http://www.windmilltours.com>

More Information

You can find more information on Wind Power at the following web site:

www.learnonline.com/ewre.html.

Appendix A: LearnOnLine's REEL Power Project



Figure A-1: A 21st Century Learning and Teaching Project sponsored by LearnOnLine, Inc.

LearnOnLine's REEL Power Project, which stands for "Renewable Energy Education Lab," is the inspiration for this course on energy. REEL Power is designed to help students understand and apply the math and science they are studying by means of a motivationally enhancing (make that "fun filled") renewable energy project. REEL Power is both classroom-based and Internet-based and makes excellent use of both these local and worldwide resources to help students with their studies.

So what exactly is REEL Power?

The REEL Power project is designed to create a "simulated", nationwide, Internet-connected power grid where each REEL Power System becomes a "virtual power plant" on the grid. To become a virtual power plant, schools, educational organizations, hobbyists and experimenters can either use the solar panels and three-phase AC alternator used in these experiments or acquire more powerful models, which are available from LearnOnLine. These become the power sources for the virtual power plants.

However, the most exciting feature of the REEL Power project is that these renewable energy devices can be connected to the Internet from your computer where the

“measure” of power is automatically sent to the REEL Power web site for display on a digitized map. As the measure of power reaches it, the REEL Power web site displays each virtual power plant as an illuminated graphic icon on the computerized map.

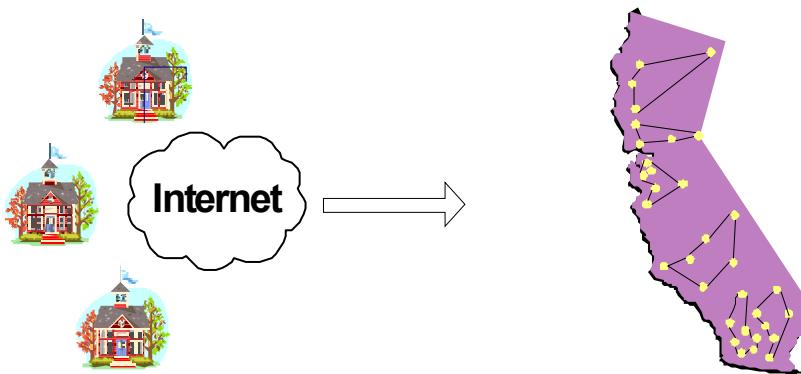


Figure A-2: Schools can form a virtual power grid over the Internet.

Schools, organizations and individuals that have the renewable energy equipment are given the label of “Power Generators”. Power Generators, like real-world power plants, generate wind and solar power for consumption by their customers. To extend the similarity of a real-world power grid, schools without the renewable energy equipment can also participate in the REEL Power project as “Power Consumers”. Power Consumer schools act as “loads” on the virtual power grid much like homes and businesses do on a real-world power grid. Power Generator schools are directed to seek out other local schools that can act as Power Consumers for their generated power, and each category of Power Generator and Power Consumer has its own separate graphic icon on the computerized map.

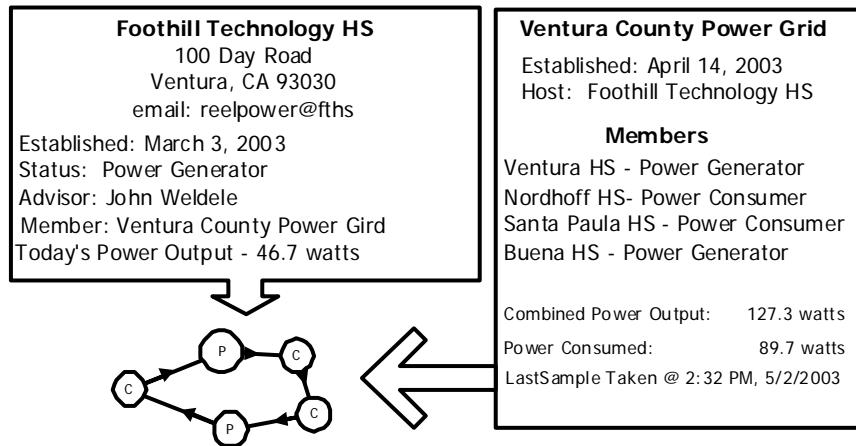


Figure A-3: REEL Power participants on the Internet

By clicking on any Power Generator or Power Consumer icon on the digital map the name of the school, teacher and students responsible for the REEL Power setup are displayed along with accompanying photos and text, as well as the amount of power generated or consumed that day.

REEL Power gives students relevant reasons for studying math and science, since it is likened to the same real-world power generation systems and elements that everyone experiences on a daily basis. Teachers always hear the same “What good is studying math and science?” and “When will I ever get to use it?” lines from their students. REEL Power gives teachers viable answers to these recurring questions and also provides the incentive for students to excel in these subjects.

With REEL Power’s understandable real-world renewable energy simulations it is anticipated that parents and caregivers will become more aware of the relevance of a math and science education in terms of future career choices for their children. This should lead to greater parental interest and participation in their children’s daily studies of these subjects. This is especially true for parents of “at risk” children who may otherwise not have the incentive to remain in school, let alone be inspired to pursue a technical career after high school.

Technically the REEL Power project makes good use of computers and the Internet; items which are in place in many schools but are rarely used to their best capabilities. And socially the project promotes collaboration among teachers and students, both in class and with other schools via the Internet, bringing together individuals from a wide variety of social and economic backgrounds to work together on a common activity.

More Information

Visit <http://www.learnonline.com> for more information on LearnOnLine's REEL Power project.

Appendix B: Source Code Listing

```

' Experiments with Renewable Energy v1.0 - AlternatorPhase1-2-3-R.bs2
' Generate 3-Phase power from wind-powered alternator, then plot all three
' phases plus the rectified sum of all three voltages.
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----
' Declarations
' -----

' ----- For Plot_It Subroutine -----
ch0      VAR     Byte          'Voltage reading from A/D CH0
ch1      VAR     Byte          'Voltage reading from A/D CH1
ch2      VAR     Byte          'Voltage reading from A/D CH2
ch3      VAR     Byte          'Voltage reading from A/D CH3
checkSum VAR     Byte          'Sum of all of above readings

' ----- For ADC0834 4-Channel Multiplexed A/D Converter -----
A2dChipSel PIN     0           ' A/D Converter Chip Select(P0)
A2dDataIn  PIN     1           ' A/D Converter Data Input(P1)
A2dClk    PIN     2           ' A/D Converter Clock(P2)
A2dDataOut PIN     3           ' A/D Converter Data Output(P3)

A2dMuxId0 CON    %1100        ' A/D MUX IDs for Ch 0..3
A2dMuxId1 CON    %1110        ' Bit 3 = 1, start bit
A2dMuxId2 CON    %1101        ' Bit 2 = 1, single (1)/diff (0)
A2dMuxId3 CON    %1111        ' Bit 1 odd / sign, Bit 0 select

a2dMuxId  VAR     Nib          ' A/D Channel MUX ID to shift out
a2dResult VAR     Byte          ' 8-bit result of A/D conversion

' ----- For Experiment 1: Programmable Battery Charger -----
ChargeLed  PIN    13          ' Charge LED (green)
DrainLed   PIN    11          ' Drain LED (red)
ReplayLed  PIN    9           ' Replay LED (yellow)

ChargeBatt PIN    15          ' to charge transistor base
DrainBatt  PIN    4           ' to drain transistor base

codePtr    VAR     Nib          ' pointer to code block to execute
codePtr = 0 VAR     Word         ' starting at 0 (Exp_1_Init)
dataPtr    VAR     Word         ' eeProm pointer for data logging
DataPtrMax CON    400          ' maximum eeProm pointer value
BattFullChg CON    150          ' ~3.00 volts for both batteries in series
True       CON    1           ' Boolean true/false

counter   VAR     Byte          ' counter for averaging every 64 seconds
avgVolts  VAR     Word          ' average battery voltage storage
avgCurrent VAR     Word          ' average battery current storage
MinVolts  CON    100          ' ~2.00 volts for both batteries in series
maxVolts  VAR     Byte          ' maximum measured battery voltage
doneDraining VAR    Bit          ' tell whether battery drained
doneCharging VAR    Bit          ' tell whether battery charged

```

```

' ----- For Experiment 2: Dueling Solar Cells -----
LeftSpLed    PIN      9          ' Left Solar Panel LED on P9
RightSpLed   PIN      8          ' Right Solar Panel LED on P8

offSet        VAR      codePtr 'Nib          ' Value proportional to A/D voltage
offset = 0           ' resolution of 0.02 volts/bit.
                      ' Example: offSet=10x0.02V = 0.20V

' ----- For Experiment 3: Solar Cell Sun Tracker -----
Servo         PIN      12         ' Sun tracker servo on P12
waitCount     VAR      Nib        ' Dampens servo's "hunting"
WaitVal       CON      1          ' initial value of waitCount

' ----- For Experiment 4: Half and Full Wave Rectification -----
sampleCount   VAR      waitCount 'Nib        ' Number of A/D samples per period
sampleArray    VAR      Byte(10)   ' 10-element array for A/D samples

' ----- For Experiment 5: Three-Phase AC Alternator -----
Plotting      CON      1          ' Program is plotting
Sampling      CON      0          ' Program is taking samples
EEPROMAddr   VAR      dataPtr 'Word        ' EEPROM address pointer
EEPROMAddr = $0000  ' Initial EEPROMAddr value
plotCount     VAR      codePtr 'Nib        ' Number plotted so far
mode          VAR      Bit        ' Plot mode or sample mode
mode = Sampling ' Initially will be sampling

'----- Main Routine -----
Main:
DO
' GOSUB Exp_1          ' Programmable Battery Charger
' GOSUB Exp_2          ' Dueling Solar Cells
' GOSUB Exp_3          ' Solar Cell Sun Tracker
' GOSUB Exp_4          ' Half&Full Wave Rectification
' GOSUB Exp_5          ' Three-Phase AC Alternator
' GOSUB Plot_It         ' Plot Data w/ StampPlot Pro
LOOP

'----- Subroutines -----
'

'----- Experiment 1: Programmable Battery Charger -----
Based on codePtr, branch to the appropriate subroutine
Exp_1_Init      codePtr = 0
Exp_1_Charge    codePtr = 1
Exp_1_Drain     codePtr = 2
Exp_1_Replay    codePtr = 3

Exp_1_Init
  Halt charging and discharging
  Extinguish all LEDs
  Zero relevant variables
  Read last codePtr value from eeProm,
  increment it

```



```

Exp_1_Charge:           ' codePtr = 1 : charge batteries
' Jump out if battery is charged already
  IF (doneCharging = True) THEN GOTO Exp_1_End

  HIGH ChargeBatt          ' activate battery charging transistor
  LOW DrainBatt            ' deactivate battery discharging transistor
  TOGGLE ChargeLed         ' flash the charge LED
  LOW DrainLED              ' extinguish the others
  LOW ReplayLED

  a2dMuxId = a2dMuxId3      ' sample the battery charge current (BCI) and
  GOSUB A2D                  ' add it to avgCurrent
  ch3 = (255-a2dResult)      ' I = (Vdd - Vcollector)/10 ohms
  avgCurrent = avgCurrent+ch3

  a2dMuxId = a2dMuxId2      ' sample the battery voltage (BV) and
  GOSUB A2D                  ' add it to aveBattVolt
  ch2 = a2dResult
  avgVolts = avgVolts+ch2

' abort charge cycle - battery is charged
  IF (ch2 > BattFullChg) AND (dataPtr <10 ) THEN
    doneCharging = True        ' set flag
    HIGH ChargeLed            ' turn charge LED on steadily
    LOW ChargeBatt            ' deactivate battery charging transistor
  ENDIF
  counter = counter - 1       ' decrement the averaging counter
  IF counter <> 0 THEN Exp_1_End ' exit if not zero, else

  WRITE dataPtr,avgVolts.HIGHBYTE ' store the average charging voltage to EEPROM
  dataPtr = dataPtr+1           ' increment the EEPROM pointer

  WRITE dataPtr,avgCurrent.HIGHBYTE ' store the average charging current to EEPROM
  dataPtr = dataPtr+1           ' increment the EEPROM pointer

  '' IF avgVolts.HIGHBYTE > maxVolts THEN maxVolts = avgVolts.HIGHBYTE
  '' IF avgVolts.HIGHBYTE <= maxVolts - 10 THEN codePtr=2:GOTO Exp_1_End

  avgVolts = 0                 ' reset for the next set of data
  avgCurrent = 0

  IF (dataPtr = DataPtrMax) THEN   ' charge cycle complete
    doneCharging = True          ' set flag so we don't charge any more
    HIGH ChargeLed              ' turn charge LED on steadily
    LOW ChargeBatt              ' deactivate battery charging transistor
  ENDIF

  GOTO Exp_1_End                ' beginning with the next time thru

Exp_1_Drain:             ' codePtr = 2 : discharge batteries
' Jump out if battery is drained already
  IF (doneDraining = True) THEN GOTO Exp_1_End

  LOW ChargeBatt              ' deactivate battery charging transistor
  HIGH DrainBatt               ' activate battery discharging transistor
  TOGGLE DrainLED              ' flash the DrainLED
  LOW ChargeLed                ' extinguish the others
  LOW ReplayLed

```

```

a2dMuxId = a2dMuxId0      ' sample the load voltage (BLV)
GOSUB A2D
ch0 = a2dResult

a2dMuxId = a2dMuxId2      ' sample the battery voltage (BV)
GOSUB A2D
ch2 = a2dResult

a2dMuxId = a2dMuxId1      ' sample the load current (BLI)
GOSUB A2D
ch1 = ch2 - a2dResult
' ch1 = I = (Vbatt-Vcollector)/10 ohms
' test battery voltage for below min
IF (ch2 < MinVolts) THEN
  doneDraining = True
  HIGH DrainLed
  LOW DrainBatt
  deactivate battery discharging transistor
ENDIF

GOTO Exp_1_End

Exp_1_Replay:   ' codePtr = 3 : replay charge cycle

LOW ChargeBatt          ' deactivate battery charging transistor
LOW DrainBatt           ' deactivate battery discharging transistor
TOGGLE ReplayLed        ' flash the replayLed
LOW ChargeLed            ' extinguish the others
LOW DrainLed

READ dataPtr,avgVolts.LOWBYTE
ch2 = avgVolts.LOWBYTE    ' display the voltage and ...
dataPtr = dataPtr+1

READ dataPtr,avgCurrent.LOWBYTE
ch3 = avgCurrent.LOWBYTE  ' current from the last charge
dataPtr = dataPtr+1

IF (dataPtr => DataPtrMax) THEN
  dataPtr = 0
  HIGH ReplayLed          ' turn replay LED on solid
ENDIF

Exp_1_End:
RETURN

' -----
' Experiment 2: Dueling Solar Cells
' -----
'

' Convert the left solar panel voltage to an 8-bit value
' Convert the right solar panel voltage to an 8-bit value
' Compare the two voltage values:
'   IF both are equal plus or minus offSet
'   Then illuminate both LEDs
'
'   IF the left solar panel voltage > the right solar panel voltage
'     Then illuminate the left LED and extinguish the right LED
'

```

```

'      IF the right solar panel voltage > the left solar panel voltage
'          Then illuminate the right LED and extinguish the left LED
'      Return
'-----

Exp_2:
    a2dMuxId = A2dMuxId1                                ' convert A/D Channel 1
    GOSUB A2D                                            ' which is the left solar panel
    ch1 = a2dResult                                       ' set CH1 = converted value

    a2dMuxId = A2dMuxId2                                ' convert A/D Channel 2
    GOSUB A2D                                            ' which is the right solar panel
    ch2 = a2dResult                                       ' set CH2 = converted value

Exp_2_Compare_CH1_CH2:
    IF (ch1 > ch2) THEN                                ' test for CH1 > CH2
        GOTO Exp_2_CH1_Is_Greater
    ELSEIF (ch2 > ch1) THEN                            ' test for CH2 > CH1
        GOTO Exp_2_CH2_Is_Greater
    ELSE
        GOTO Exp_2_CH1_Equals_CH2
    ENDIF

Exp_2_CH1_Equals_CH2:
    HIGH LeftSpLed                                      ' both CH1 and CH2 are equal, so
    HIGH RightSpLed                                     ' illuminate the left LED and
    GOTO Exp_2_End                                       ' illuminate the right LED
                                                       ' and exit

Exp_2_CH1_Is_Greater:
    ch1 = ch1 - offSet                                 ' CH1 is > CH2, so
    IF (ch1 = ch2) THEN                                ' subtract the offSet from CH1
        GOTO Exp_2_CH1_Equals_CH2
    ELSE
        HIGH LeftSpLed                                  ' branch if CH1 = CH2, else
        LOW RightSpLed                                 ' illuminate the left LED and
        GOTO Exp_2_End                                   ' extinguish the right LED
                                                       ' and exit

Exp_2_CH2_Is_Greater:
    ch2 = ch2 - offSet                                 ' CH2 is > CH1, so
    IF (ch2 = ch1) THEN                                ' branch if CH2 = CH1, else
        GOTO Exp_2_CH1_Equals_CH2
    ELSE
        LOW LeftSpLed                                 ' extinguish the left LED and
        HIGH RightSpLed                               ' illuminate the right LED
        GOTO Exp_2_End                                 ' and exit

Exp_2_End:
    RETURN

'-----
' Experiment 3: Solar Cell Sun Tracker
'-----


' Based on the values for CH1 and CH2 from Experiment 2
' Determine if the servo should stop, move left or move right
'

```

```

' By moving the servo left or right, the individual solar cell voltages
' will eventually become equal and the program will cause the servo to stop.
' That is, until the sun moves and the voltages become unequal again thus
' causing the servo to reposition the solar panels.

' Return

'-----


Exp_3:
IF (ch1 = ch2) THEN Exp_3_Stop
IF (ch1 < ch2) THEN Exp_3_Move_Left
IF (ch1 > ch2) THEN Exp_3_Move_Right

Exp_3_Stop:
PULSOUT Servo, 750
GOTO Exp_3_Set_Wait_Count

                                ' CH1 = CH2 so
                                ' stop the servo
                                ' and re-initialize the waitCount

Exp_3_Move_Left:
waitCount = waitCount - 1
IF (waitCount <> 0) THEN
    GOTO Exp_3_End
ELSE
    PULSOUT Servo, 755
    GOTO Exp_3_Set_Wait_Count
ENDIF

                                ' CH1 < CH2 so
                                ' decrement the waitCount
                                ' and re-initialize the waitCount

Exp_3_Move_Right:
waitCount = waitCount - 1
IF (waitCount <> 0) THEN
    GOTO Exp_3_End
ELSE
    PULSOUT Servo, 745
    GOTO Exp_3_Set_Wait_Count
ENDIF

                                ' CH1 > CH2
                                ' Decrement the waitCount
                                ' and re-initialize the waitCount

Exp_3_Set_Wait_Count:
waitCount = WaitVal

                                ' Re-initialize the waitCount
                                ' and exit

Exp_3_End:
RETURN

'-----


' Experiment 4: Half and Full Wave Rectification
'-----


' Set the A/D converter to convert only on ch3

' If sampleCount is not equal to 10
' Then plot existing samples
' Else prepare to plot new samples

' To plot new samples first wait for the rectified wave to equal zero
' Once zero, wait for the beginning of the first peak
' Then take ten (10) samples as rapidly as possible
' Zero sampleCount to begin plotting next

' Plot each sample in the order taken using Plot_It
' Increment sampleCount and

```

```

' Exit
'

'-----
' Exp_4:
a2dMuxId = A2dMuxId3                                ' Set the A/D to convert Channel 3
IF (sampleCount <> 10) THEN
    GOTO Exp_4_Plot
ENDIF

Exp_4_Wait_For_Zero:                                  ' Loop until the rectified wave is zero
    GOSUB A2D
    IF (a2dResult <= 5) THEN
        GOTO Exp_4_Wait_For_First_Peak
    ENDIF
    sampleCount = sampleCount + 1
    IF (sampleCount = 10) THEN
        GOTO Exp_4_Take_Samples
    ENDIF
    GOTO Exp_4_Wait_For_Zero

Exp_4_Wait_For_First_Peak:                            ' Loop until the first peak is detected
    GOSUB A2D
    IF (a2dResult > 5) THEN Exp_4_Take_Samples
    GOTO Exp_4_Wait_For_First_Peak

Exp_4_Take_Samples:                                 ' As rapidly as possible take 10
    FOR sampleCount = 0 TO 9                          ' A/D samples and save them in the
        GOSUB A2D                                    ' sampleArray
        sampleArray(sampleCount) = a2dResult
    NEXT
    sampleCount = 0                                   ' Re-initialize sampleCount for plotting

Exp_4_Plot:
    ch3 = sampleArray(sampleCount)                  ' Get the next A/D sample into CH3
    sampleCount = sampleCount + 1                   ' and increment sampleCount

Exp_4_End:
    RETURN

'-----
' Experiment 5: Three-Phase AC Alternator
'-----

' Enter from power-on or reset with eepromAddr = $000,
' mode = Sampling, and plotCount = 0

' This routine will be called once each time through Main, and right
' after this routine, Plot_It will be called.

' If mode = Sampling, then take a set of 40 samples from all
' 4 A/D converter inputs. 10 samples per channel.

' If mode = Plotting, then retrieve one set of 4 samples from
' EEPROM. Jump out of Exp_5... the one set will get plotted when
' Plot_It is called. This will happen 10 times to get all 10 sets
' of 4 samples plotted.
'
```

```

' Sampling:
'   For each of the three phases and the rectified output,
'   take samples as follows:
'
'     - Increment the eepromAddr to the next sub-block (except for phase 3)
'     - Set the A/D converter to sample phase 3
'     - Sync up to the positive peak of phase 3 (except for phase 3, itself)
'     - Switch the A/D to sample the next phase or the rectified output
'       (again, except for phase 3)
'     - Gather 10 samples of the converted values in sampleArray (RAM)
'     - Transfer the 10 samples from sampleArray to EEPROM
'
' Plotting:
'   For each of the three phases and the rectified output,
'   plot samples as follows:
'
'     - Increment plotCount for the next time around
'     - Read each sample from EEPROM into the appropriate
'       variable (ch3, ch2, ch2, ch0)
'     - Exit the program and let the Plot_It routine graph the four samples
'     - When we've plotted all the samples, increment the eepromAddr
'       to the next sub-block, change mode to Sample the next time through,
'       and reset plotCount to zero
'
' Note: If the eepromAddr is greater than address $300, reset it to zero -
'       we don't want to clobber the program memory
'

'-----
' Exp_5:
SELECT mode
CASE Plotting
    plotCount = plotCount + 1          ' Will plot 10 sets of data
    ' numbered 0..9
    GOSUB Exp_5_Retrieve_For_Plottin
    IF (plotCount = 9) THEN
        mode = Sampling             ' Done plotting, switch state
        plotCount = 0                ' Reset plotCount
        GOSUB Exp_5_Inc_EepromAddr  ' Inc. addr to next bank
    ENDIF
CASE Sampling
    GOSUB Exp_5_Sample_Phase_3      ' Take 10 samples each of Phases
    GOSUB Exp_5_Sample_Phase_2      ' 3, 2, 1, Rectified Out
    GOSUB Exp_5_Sample_Phase_1
    GOSUB Exp_5_Sample_Rectified_Output
    GOSUB Exp_5_Retrieve_For_Plottin
    mode = Plotting               ' Queue up 1st set of samples
                                    ' Change state back to plotting
ENDSELECT

Exp_5_End:
RETURN

'-----
' Subroutines for Experiment 5
'-----

Exp_5_Sample_Phase_3:
a2dMuxId = A2dMuxId3           ' Set the A/D to convert Channel 3
GOSUB Exp_5_Wait_For_Zero      ' Loop until the wave is zero

```

```

GOSUB Exp_5_Wait_For_Next_Peak           ' Wait for wave to begin to peak
GOSUB Exp_5_Take_Samples                 ' Take samples of phase 3
GOSUB Exp_5_Write_Samples                ' Write samples to EEPROM

Exp_5_Sample_Phase_3_End:
    RETURN

'-----

Exp_5_Sample_Phase_2:
    GOSUB Exp_5_Inc_EepromAddr           ' Increment pointer to EEPROM
    a2dMuxId = A2dMuxId3
    GOSUB Exp_5_Wait_For_Zero           ' Set the A/D to convert Channel 3
    GOSUB Exp_5_Wait_For_Next_Peak     ' sync up to phase 3
    GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
    GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak

    a2dMuxId = A2dMuxId2
    GOSUB Exp_5_Take_Samples            ' Switch the A/D to convert Channel 2
    GOSUB Exp_5_Write_Samples           ' Take samples of phase 2
    GOSUB Exp_5_Write_Samples           ' Write samples to EEPROM

Exp_5_Sample_Phase_2_End:
    RETURN

'-----

Exp_5_Sample_Phase_1:
    GOSUB Exp_5_Inc_EepromAddr           ' Increment pointer to EEPROM
    a2dMuxId = A2dMuxId3
    GOSUB Exp_5_Wait_For_Zero           ' Set the A/D to convert Channel 3
    GOSUB Exp_5_Wait_For_Next_Peak     ' sync up to phase 3
    GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
    GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak

    a2dMuxId = A2dMuxId1
    GOSUB Exp_5_Take_Samples            ' Switch the A/D to convert Channel 1
    GOSUB Exp_5_Write_Samples           ' Take samples of phase 1
    GOSUB Exp_5_Write_Samples           ' Write samples to EEPROM

Exp_5_Sample_Phase_1_End:
    RETURN

'-----

Exp_5_Sample_Rectified_Output:
    GOSUB Exp_5_Inc_EepromAddr           ' Increment pointer to EEPROM
    a2dMuxId = A2dMuxId3
    GOSUB Exp_5_Wait_For_Zero           ' Set the A/D to convert Channel 3
    GOSUB Exp_5_Wait_For_Next_Peak     ' sync up to phase 3
    GOSUB Exp_5_Wait_For_Zero           ' Loop until the wave is zero
    GOSUB Exp_5_Wait_For_Next_Peak     ' Wait for wave to begin to peak

    a2dMuxId = A2dMuxId0
    GOSUB Exp_5_Take_Samples            ' Switch the A/D to convert Channel 0
    GOSUB Exp_5_Write_Samples           ' Take samples of the rectified wave
    GOSUB Exp_5_Write_Samples           ' Write samples to EEPROM

Exp_5_Sample_Rectified_OutputEnd:
    RETURN

```

```

' Retrieve 4 samples, one for each channel. This will be called 10 times,
' to get the total "bank", which is 4 sets of 10 samples per channel.
Exp_5_Retrieve_For_Plottting:
    READ (epromAddr - $30 + plotCount), ch3 ' Read phase 3 into ch3
    READ (epromAddr - $20 + plotCount), ch2 ' Read phase 2 into ch2
    READ (epromAddr - $10 + plotCount), ch1 ' Read phase 1 into ch1
    READ (epromAddr - $00 + plotCount), ch0 ' Read rectified val into ch0

Exp_5_Retrieve_For_Plottting_End:
    RETURN

' -----
Exp_5_Wait_For_Zero:
    DO                                ' Loop until the rectified wave
        GOSUB A2D                      ' is below 0.10 volts
    LOOP UNTIL (a2dResult <= 5)          ' (5 x 0.02 volts = 0.10 volts)

Exp_5_Wait_For_Zero_End:
    RETURN

' -----
Exp_5_Wait_For_Next_Peak:
    DO                                ' Loop until the rectified wave
        GOSUB A2D                      ' is non-zero
    LOOP UNTIL (a2dResult => 5)

Exp_5_Wait_For_Next_Peak_End:
    RETURN

' -----
Exp_5_Take_Samples:
    FOR sampleCount = 0 TO 9           ' Enter with A2D channel set
        GOSUB A2D                      ' Acquire 10 samples of the
        sampleArray(sampleCount) = a2dResult ' wave as quickly as possible
    NEXT

Exp_5_Take_Samples_End:
    RETURN

' -----
Exp_5_Write_Samples:
    FOR sampleCount = 0 TO 9           ' Enter with eepromAddr set
        WRITE (epromAddr + sampleCount), sampleArray(sampleCount) ' Store the wave samples to EEPROM
    NEXT

Exp_5_Write_Samples_End:
    RETURN

' -----
Exp_5_EepromAddr_To_Zero:
    eepromAddr = $0000                 ' Initialize the eepromAddr
                                         ' to the bottom of EEPROM

Exp_5_EepromAddr_To_Zero_End:
    RETURN

```

```

'-----  

Exp_5_Inc_EepromAddr:  

    eepromAddr = eepromAddr + 16           ' Add 16 to move to next bank  

    IF (eepromAddr >= $0300) THEN        ' Make sure not to overwrite  

        GOSUB Exp_5_EepromAddr_To_Zero   ' our PBASIC program which  

    ENDIF                                ' starts around $0300  

Exp_5_Inc_EepromAddr_End:  

    RETURN  

'----- End Exp_5 Subroutines -----  

'-----  

' A/D Converter Routine  

'  

'  

' Enter with a2dMuxId as the channel to convert  

' Return with 8-bit result in a2dResult (msb,6,5,4,3,2,1,lsb)  

'  

'-----  

A2D:  

    HIGH A2dChipSel                   ' Initialize signals  

    LOW A2dDataIn                     ' Disable A/D Chip Select  

    LOW A2dClk                        ' Initial state of data in  

    a2dResult = 0                      ' Initial state of clock  

                                      ' Clear the 8-bit result  

A2D_Start_Conversion:  

    LOW A2dChipSel                   ' Start the conversion process  

                                      ' Enable A/D chip select  

A2D_Shift_Out_Channel_ID:  

    SHIFTOUT A2dDataIn,A2dClk,MSBFIRST,[a2dMuxId\4]          ' Shift out the Channel ID value  

A2D_Shift_In_Result:  

    PULSOUT A2dClk,10                ' Shift in the result  

    SHIFTIN A2dDataOut,A2dClk,MSBPRE,[a2dResult\8]            ' Disable A/D chip select  

    HIGH A2dChipSel  

A2D_End:  

    RETURN                            ' Return to calling routine  

'-----  

' Plot Acquired Data Using StampPlot Software  

'-----  

Plot_It:  

    checkSum = ch0 + ch1 + ch2 + ch3      ' Compute checksum for StampPlot  

    DEBUG ch0, ch1, ch2, ch3, checkSum    ' Transmit data to StampPlot  

    PAUSE 250                           ' Give StampPlot time to plot data  

Plot_It_End:  

    RETURN

```



Appendix C: Programming StampPlot Pro

StampPlot Pro is a serial computer interface program for data acquisition and control for the Parallax, Inc, series of BASIC Stamp® microcontroller modules. StampPlot is a very versatile program with many great features. With the large number of choices available, the program can be a little intimidating to the new user. This guide will discuss many of the most popular features of StampPlot, though only brief examples are provided to help you get started.

Here is a partial list of StampPlot features covered in this document:

- Basic control of the plot through the button bar
- Plotting analog or digital data easily
- Control virtually all StampPlot settings with code
- Accept data as strings or as binary values
- Save data and images of the plots
- Use image, drawing and sound files for effects
- Place controls on the interface for monitoring and control
- Perform math operations
- Configure from macros (text scripts)
- Use an interface for interactive control of your BASIC Stamp module
- Perform acquisition and control over the Internet

Registering Your Software

StampPlot Pro has two types of registration licenses:

Developer's License for users who wish to use the development features of StampPlot to create configurations, which include drag-and-drop design using the Object Editor and Macro editor to quickly design Graphical User Interfaces (GUIs).

Standard Users License for those not interested in development. As discussed in this documents, development work may still be done, but not as simply as with the Developer's License. StampPlot may be used with the Standard license for free by educational institutions and home user for personal use.

StampPlot will run for 10 minutes without a registration license for testing. After you have obtained registration, or qualify for the free standard license, you may register by

selecting the appropriate license from the Register Menu and entering your provided User Name and Code.

Section 1: Installation and Basic Control

Installation

You may find StampPlot Pro on the Parallax CD November 2003 or newer. From the Welcome page click on Software → BASIC Stamps → StampPlot Pro, then click Install.

- ✓ Install StampPlot as you would any software application by running the setup.exe program.

Once installed, the program is available from:

Start menu → Programs → StampPlot Pro Version 3 → StampPlotV3, or from the icon on your desktop if available.

The first time StampPlot is ran it will finish installing the required files and then will ask you to will close and restart StampPlot. The distribution notes will also be displayed.

- ✓ Open StampPlot and follow any remaining installation instructions that appear.

Registration

- ✓ Select Register from the menu bar.
- ✓ Select Free Home/Educ Standard License.
- ✓ A box will appear informing you that some features will be disabled. Click Yes.
- ✓ Another box will appear thanking you for registering. Click OK.

Configuring and Testing

Before going any further let's make sure you can plot some simple data using StampPlot.

- ✓ Open your BASIC Stamp Editor software.
- ✓ Open StampPlot, if it is not still open.

The default screen, shown in Figure C-1, provides a variety of ready to run plots and a BASIC Stamp module test program.

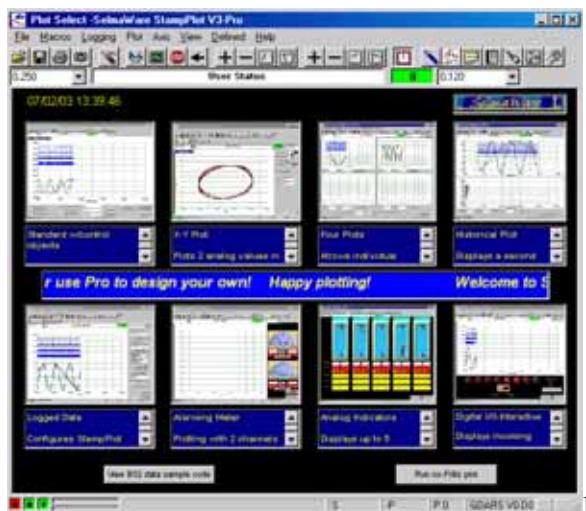


Figure C-1
StampPlot Default Screen

- ✓ Click on the button "View BS2 Data Code Sample."
- ✓ Answer Yes to the security message.
- ✓ A text window will open with PBASIC code. Copy and paste the code into your BASIC Stamp Editor.
- ✓ In the BASIC Stamp Editor, select the model of BASIC Stamp module you are using from the toolbar to place the appropriate compiler directive in your program.
- ✓ Download the program to your BASIC Stamp module.
- ✓ View the BASIC Stamp Editor Debug Terminal briefly. You will see data being sent from the BASIC Stamp module.
- ✓ Make note of the Com Port in use; this can be found in the top right box of the Debug Terminal.
- ✓ Close the Debug Terminal. Only one program can access the communications port receiving this data at any one time. Before StampPlot can accept and use this data the BASIC Stamp Editor's Debug Terminal must be closed.
- ✓ Bring up StampPlot Pro.
- ✓ In StampPlot, click the "Run No-Frills Plot" button. This will give you the simple plotting configuration.
- ✓ Click the Connect button on the StampPlot button bar. 

If you get an error message stating there was a problem opening the port:

- ✓ Ensure that the BASIC Stamp Editor Debug Terminal is closed.
- ✓ Open the Configuration Window. 
- ✓ Select the Port tab.
- ✓ Select the correct COM Port, which must be the same one that was in use by the BASIC Stamp Editor's Debug Terminal.
- ✓ Click "Set as Default" to lock it in.
- ✓ Try connecting again.

If you do not get a connection error message, but do not see an indication of data arriving (R flashing red in lower left corner and no data being plotted):

- ✓ Select the Port tab.
- ✓ Select the correct COM Port to see if you had the wrong one.
- ✓ If that doesn't help, un-check "DTR Enabled". For certain BASIC Stamp hardware configurations having this checked will 'lock-up' your BASIC Stamp module. All carrier boards from Parallax will work with this enabled.
- ✓ Reconnect.

Once you are connected, and you see the flashing red "R" in the lower left hand corner of your screen, continue on as follows:

- ✓ Click the Plot button to enable plotting on StampPlot. 
- ✓ Click the Reset button to reset the plot back to time 0. 

If all has gone well you should see 4 analog channels and 2 digital values being plotted! The data is plotted with value for the Y-axis and time of plotting on the X-axis.

Data Points

To be able to redraw and save the plot, the incoming digital and analog data is stored in what are termed 'Data Points'. By default the number of Data Points stored is 500. This may be adjusted under the Configuration Window's Data tab. Changing the number of points will reset your plot. The amount of data points used is shown graphically in the lower left corner Figure C-2.

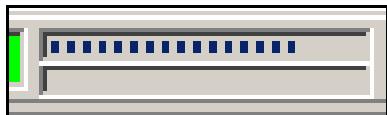


Figure C-2
Data Points Bar Graph



Once the data points are full, the plot will reset. Very large amounts of data points are NOT recommended (over 5000) as they will considerably slow the plot when it is refreshed.

To perform continuous plotting the option to ‘Flush Old Data’ may be enabled pressing F11. This will remove a percentage of the oldest data from the Data Points storage each time it fills to maximum.

Data Queue

As data arrives it is placed in a queue to wait for processing. If data arrives too quickly, the queue will begin to fill and there will be a delay from the time data was sent by the BASIC Stamp module to the time it is plotted. The bottom bar indicates the amount of data awaiting processing. The speed with which StampPlot can process data is dependent on the speed of your computer, the nature of the data, and other factors, but a rule of thumb is one piece of data every 10milliseconds (.01 seconds).

Other Plot Control Buttons

Stop and Shift:

Using Stop: When the plot reaches the maximum time on the plot, the Plot button will go-up stopping the plotting of new data.

Using Shift: When the plot reaches the maximum time the plot will shift to the left and display the new data being plotted.

Adjusting the Plot Scale:

The first set of four buttons is used to adjust the Y-axis (analog value) scales:

- + Double scale.
- Half scale.
- ▲ Shift scale up.
- ▼ Shift scale down.

Pressing CTRL-A will auto adjust the scales for the plotted minimum and maximum scales. Pressing CTRL-L will shift into/out-of logarithmic Y scales.

The second set of arrows is used to adjust the X-axis (time) in a similar manner.

Pressing CNTL-R will change the X-axis to show time in Real-Time – time of day and date.

Testing out the Installed Macro Configurations

While the data sent is only 4 analog and 2 digital values, StampPlot can perform many functions without changing any of the code in the BASIC Stamp module.

- ✓ Press CTRL-F1 or select the menu option Macro → Run Startup Macro to return to the initial loading screen,
- ✓ Select the 1st plot style choice by clicking the image. You may read about it in the scrolling text box below it prior to clicking the image.
- ✓ Connect to the BASIC Stamp module and observe how the data is used.
- ✓ Some of the control choices may be a little confusing at first, but feel free to experiment! This document will help clarify much of what you see.

Section 2: Plotting Analog, Digital and Messages

StampPlot analyzes the format of the arriving data and uses it. General rules for data are:
Analog data begins with a value, and can have up to 10 values separated by commas.

Digital data begins with % and contains only 1's or 0's.

Messages are those not meeting the above (more rules on this later).

ALL lines must end in a carriage return (CR).

Plotting Analog Data- ASCII Format

Analog data can be sent from the controller to be plotted by sending the values as text or ASCII strings. For example, the following lines of code will send data as text. The **DEBUG** command is used to send data from the controller back to the computer.

Analog data is formatted as a text value followed by a carriage-return (CR).

- ✓ In StampPlot, disconnect from the Com Port if you have not already done so. This will allow you to send a new program to your BASIC Stamp module.

- ✓ Enter the program below into your BASIC Stamp Editor.
- ✓ Run the program.
- ✓ Monitor the BASIC Stamp Editor's Debug Terminal to observe the format of data being sent before closing it.
- ✓ Use the No-Frills choice of StampPlot for testing: Connect, Plot, Enable Shifting and Data Flushing (F11).



```
'{$STAMP BS2}
'{$PBASIC 2.5}
'Plotting 1 analog value

x VAR Byte

DO
  FOR x = 0 TO 255
    DEBUG DEC x, CR
    PAUSE 100
  NEXT
LOOP
```

As can be seen in your Debug Terminal, the data arrives as a 0, then 1 and so on. Each value is represented by 1, 2 or 3 characters (bytes) forming each number. This is due to the **DEC** modifier formatting the values as text.

When StampPlot is connected and plotting you will see the values plotted. The **PAUSE** command is used to prevent the data queue from filling from high-speed data.

Multiple analog values may be plotted by separating each value with a comma. Notice the form is a **DEC** followed by a variable, a comma, a comma in quotes, a comma and another value. PBASIC requires a comma between each value or string sent, and StampPlot requires commas as part of the text.

- ✓ Disconnect the Com Port from StampPlot.
- ✓ Enter and run the following program in the BASIC Stamp Editor.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
'Plotting 3 analog values

val    VAR Byte
val2   VAR Byte
val3   VAR Byte
```

```
DO
  FOR Val = 0 TO 255
    val2 = val / 2
    val3 = val * 2
    DEBUG DEC val, ",", DEC val2, ",",
    DEC val3, CR
    PAUSE 100
  NEXT
LOOP
```

- ✓ Look at the Debug Terminal to ensure your data is properly formatted. It should look like the example seen below:

100,50,200

- ✓ Close the Debug Terminal
- ✓ Open StampPlot and connect.
- ✓ Click on the Plot button. Your graph should look like the one below.

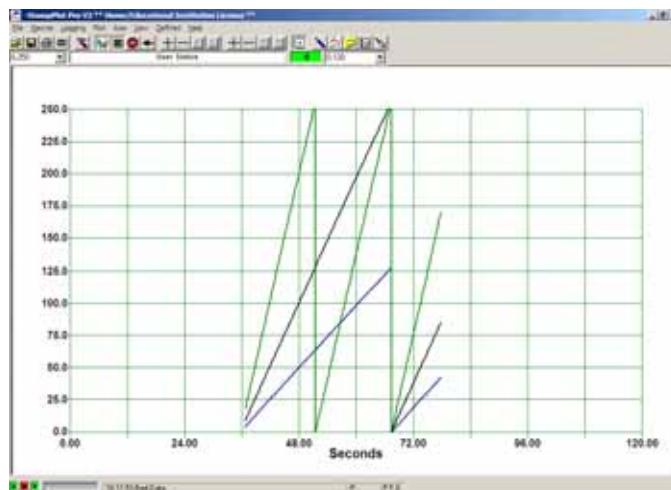


Figure C-3
Plotting Three
Analog Values

Up to 10 values may be plotted simultaneously by separating them with commas.



Plotting Digital Values

By beginning a line with % StampPlot will plot the binary digits – bits (1's and 0's) as digital traces. The **IBIN DEBUG** modifier is typically used since this will send the values as binary values starting with %. For example

```
DEBUG IBIN8 129, CR
```

will display

%10000001

- ✓ Enter and run the following program.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
'Plotting 8 binary values

val VAR Byte

DO
    FOR val = 0 TO 255
        DEBUG IBIN8 val, CR
        PAUSE 100
    NEXT
LOOP
```

- ✓ Close the Debug Terminal in the BASIC Stamp Editor.
- ✓ Connect in StampPlot and view the signals.

The **IBIN8** modifier caused the BASIC Stamp module to send the data with 8 positions. It is very important that the same number of bits is sent consistently and ends with a carriage return.

Sending Messages

When a string is sent that does not start with a value or a percent sign (and a few other restrictions later on), StampPlot will treat it as a message and list it in the message window. As always, the line must end with a carriage return.

- ✓ Enter and run the following program.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
```

```
'Sending messages
val VAR byte
DO
    FOR val = 0 TO 255
        DEBUG "the value is", DEC val, CR
        PAUSE 100
    NEXT
LOOP
```

Of course analog, binary and messages may be mixed in the same program, but each requires their own line ending with a **CR**.

Section 3: Debug/Immediate Window and Control Instructions

The Debug/Immediate Window is used to monitor data as it is processed or to directly enter data and instructions.

- ✓ Re-run the “Plotting 3 Analog Values” program.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
'Plotting 3 analog values

val      VAR Byte
val2     VAR Byte
val3     VAR Byte

DO
    FOR Val = 0 TO 255
        val2 = val / 2
        val3 = val * 2
        DEBUG DEC val, ",", DEC val2, ",", DEC val3, CR
        PAUSE 100
    NEXT
LOOP
```

- ✓ Close the Debug Terminal
- ✓ In StampPlot, connect and plot.
- ✓ Open the Logs Debug/Immediate Window with this button: 
- ✓ In the Logs Debug/Immediate Window check the Anlg (analog) check box to view analog data as it is processed.
- ✓ Note that the analog values are displayed.

- ✓ Disconnect on StampPlot.
- ✓ Keep plotting enabled.
- ✓ Reset the plot area.
- ✓ In the text box at the bottom of the Logs Debug/Immediate Window, type in three values, pressing Enter on your keyboard after each value.
- ✓ Note that these values were plotted.
- ✓ Use the keyboard UP arrow to display your last entry, then press Enter again.
- ✓ Plot several other values for the fun of it.
- ✓ Try some binary values, such as %1001, and a text message such as Hello World!



NOTE: The plot will not shift automatically when the connection is not open.

Control Instructions

The box you were using is the Command Line Interface (CLI) text box, where you can enter test values or instructions. Virtually every facet of StampPlot can be controlled using control instructions. These instructions may come from:

- The user manually entering them in the Debug/Immediate CLI.
- Serially from the BASIC Stamp module as strings.
- From a PC based text file (macro).
- Over the Internet using the TCP-Serial Gateway program.

All control instructions are 4-lettered mnemonics starting with an exclamation point (!), and of course, must end with a carriage return.

Try the following instructions one at a time using the CLI, and watch the effect of each. Explanations to the right are not to be entered or used. Instructions to enter will be *italics* for emphasis.

<i>!POBJ Clear</i>	Removes all controls form the plot screen
<i>!NEWP</i>	Starts a new plot – default configuration
<i>!SPAN -100,100</i>	Sets the analog (Y-axis) scales
<i>!TMAX 600</i>	Sets maximum time for 600 minutes
<i>!RTIM ON</i>	Enables Real-Time on X-axis
<i>!SHFT ON</i>	Enables plot shifting at maximum
<i>!FLSH ON</i>	Enables data flushing

<i>!TITL Practice</i>	Titles the plot window
<i>!PLOT ON</i>	Enables plotting
<i>!RSET</i>	Resets the plot
<i>!STAT This is a message</i>	Places a message in the User Status text box in the plot.
<i>!DEBUG Hello!</i>	Displays data in the Debug Window.

Instructions that use ON/OFF may also use 1/0:

!CONN 1 Connect on COM port

These control instructions may also be part of the PBASIC code as a **DEBUG** instruction.

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'Plotting 1 analog value & using control instructions
PAUSE 100

DEBUG CR,"!POBJ Clear", CR
DEBUG "!NEWP", CR
DEBUG "!PLOT ON", CR
DEBUG "!SHFT ON", CR
DEBUG "!RSET", CR

Val      VAR Byte

DO
  FOR Val = 0 TO 255
    ' Plot Value
    DEBUG DEC VAL, CR
    ' Show value in Status box
    DEBUG "!STAT Value = ", DEC Val, CR
    PAUSE 1000
  NEXT
LOOP
```

As always, watch the BASIC Stamp Editor Debug Terminal to verify the strings look well formed.

NOTE: When StampPlot connects it will cycle the DTR line causing the BASIC Stamp module to reset. This is important to ensure the configuration information at the beginning of the program is sent. If the DTR Enabled option is unchecked you may need to manually reset your BASIC Stamp module.

To insure there is not a portion of a previous line (from resetting your stamp in the middle of sending) in the StampPlot queue, always start your 1st **DEBUG** with a **CR** to end the sting portion.

The StampPlot Help files, Summaries, provides a full listing of available control instructions and use.



Section 4: Plotting Analog Values with Binary Values

Data is sent as 1's and 0's represented by different voltage levels from the BASIC Stamp module to the computer. Occasionally a glitch may occur causing perhaps the number 100 to be 300 just because 1 bit (1 or 0) out of 24 was in error (each character in the number 100 is represented by 1 byte which is 8 bits, giving 24 bits for 3 characters). This doesn't happen often (hopefully!) but when it does it can cause problems if you are performing important data collection.

StampPlot provides for the means to use each byte as a unique value and a process called Checksum to verify the integrity of the data. One byte may represent a value between 0 and 255.

We will test this with 3 analog values to be plotted with checksum verification. The first step is to configure StampPlot to use data in this format.

- √ Under the Configuration Window, Data Tab, select:
 - Use Binary Data
 - Number of bytes per data set: 3
 - Use Checksum must be checked.
- √ OR, use StampPlot control instructions in the CLI.
 - **USEB ON**
 - **!NUMB 3**
 - **!CSUM ON**

This configures StampPlot to expect 3 byte values plus a 4th for the checksum value. To understand what is occurring consider the example if our data were 10, 20 and 30 for the 3 values. Instead of sending a character for 1 then a character for 0 (2 bytes) for the value 10 it sends a single byte of the value 10 and similarly for 20 and 30. The largest value that may be sent is 255 since that is the maximum value for a byte.

Checksum means that the values of the individual bytes in each data set (or packet) are added up and that is sent as a byte. What would the checksum value be in this case? If you said 60 you'd be correct ($10+20+30$). When StampPlot sees that it has received 4 bytes it adds up the 1st 3 and compares what it calculated to the 4th. If there is a difference StampPlot will not use the data set and issue an error message in the Immediate/Debug window.

You may often get checksum error messages when first connecting due to connecting in the middle of a packet but StampPlot will quickly recover and find good packets.

✓ Let's write a program to send data for this configuration:

```
'{$STAMP BS2}
'{$PBASIC 2.5}

' Plotting 3 analog values from binary data

Val      VAR Byte
Val2     VAR Byte
Val3     VAR Byte

DO
  FOR Val = 0 TO 255
    Val2 = Val / 2
    Val3 = Val * 2
    DEBUG Val, Val2, Val3, Val+Val2+Val3
    PAUSE 100
  NEXT
LOOP
```

Note the difference from the previous line to plot 3 values:

```
DEBUG DEC VAL, "", DEC VAL2, "", DEC VAL3, CR
```

- It does not use the **DEC** modifier.
- It does not use the quoted comma-separators (PBASIC requires a comma between each value sent).
- The last value sent is the sum of the 3 bytes.
- No **CR** is sent at the end. This would be a 5th character sent (a value of 13).

Since a byte can only hold values up to 255, what happens if we send the values of 255,2,2? The checksum value will 'roll over' and start back at 0, so $255+2 = 1 + 2 = 3$.



Don't worry though, the BASIC Stamp module and the StampPlot software know this well and take care of it for you.

NOTE: Because data is sent in predefined packet sizes and the bytes represent values, you **CANNOT** send any other forms of data directly, such as data to be plotted as binary or control instructions once StampPlot is in binary mode.

Section 5: Logging and Saving

Logging Data and Messages

StampPlot allows data and messages to be saved to text files. The data is saved as comma-separated values and optionally time stamped.

The choices can be found under the Logging menu. Choices include saving data and messages to files and opening data and message files. Files are saved to the StampPlot Data directory. StampPlot control instructions for these are:

Files may be named with the instructions:

<i>!SAVD ON</i>	Enable saving of data file.
<i>!SAVM ON</i>	Enable saving of data file.
<i>!NAMD filename.txt</i>	Name of data file to save to.
<i>!NAMM filename.txt</i>	Name of message file to save to.

As a reminder these instructions may be sent by the BASIC Stamp module to ensure your data collection is on.

Saving Plots and Snapshots.

Saving a plot is saving the configuration of the plot and the current data points and messages in the message window. A plot may then be reopened and analyzed. If data flushing is on only current data in the data points will be saved for the plot. By clicking the File save button a directory and file save choice will open. By default saved files are to the data directory.

!SAVP filename

StampPlot can also save a jpg image of the plot, a snapshot, by clicking the camera. The image is saved to the data directory and appended with the date and time.

!SNAP filename

Plot files and snapshot saves can be configured to be automatically saved also.

!ASAV ON
!ASNP ON

There are choices to perform these when the data points are at maximum (**!MAXP**) or when the plot reaches the maximum time before shifting (**!MAXT**).

The date and time are automatically appended to the file names unless disabled
!APDT OFF

By default only the plot area information is saved as a file or a snapshot. The entire form, including controls, may be saved by enabling the choice under the File menu or using the instruction.

!FORM ON

Section 6: Use drawing, image and sound files for effects.

Drawing

StampPlot allows drawing of shapes and placing text on the plot. In general, the structure is: The drawing type + instruction, coordinates, parameters.

For example:

@TEXT 35a, 102a, 2, (BLUE), StampPlot Rules!

Drawing type

StampPlot supports 4 different types of drawing depending on what you want to produce.

Starts with @: Drawing is ‘constant’ and will survive a reset. Good if you want something on the plot permanently even if you reset.

Starts with ~: Drawing is ‘temporary’ and will be erased with resets, or anytime the plot shifts or is refreshed.

Starts with ^: Drawing is treated as a data points and will be plotted just as data is when the plot is shifted. As data, it is erased when the plot is reset. Plotting must be enabled.

Starts with !: Drawing is the same as using ^ but the drawing is restricted to the data plotting area. Plotting must be enabled.



- ✓ Test the following in the Immediate/DEBUG window's CLI:

```
!POBJ Clear
!NEWP
!PLOT ON
@TEXT 35a, 102a, 2, (BLUE), StampPlot Rules!
~FREC 10,25,30,150, (RED)
^FCIR 96,125,5, (GREEN)
!RECT 48,50,72,175, (BLACK)
```

- ✓ Now shift around the plot and compare what occurs as the plot is moved.

Why didn't the text move? The objects change position because their coordinates are based on the plot coordinates. Using absolute coordinates, you can plot using points independent of the plot scales, where lower left is 0a, 0a, and upper right is 100a, 100a.

Of course, these instructions can come from the BASIC Stamp module using **DEBUG**.

- ✓ Try this using this in a PBASIC program:

```
'{$$STAMP BS2}
'{$PBASIC 2.5}

'Plot and mark values of points

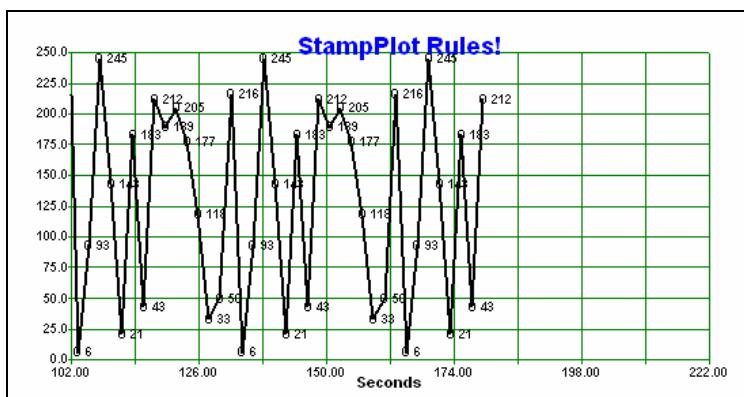
PAUSE 1000

DEBUG CR,"!POBJ Clear",CR
DEBUG "!NEWP",CR
DEBUG "!PLOT ON",CR
DEBUG "@TEXT 35a, 102a, 2, (BLUE), StampPlot Rules!", CR

x VAR Byte
x = 100

DO
  RANDOM x
  ' Plot value
  DEBUG DEC x,CR
  ' Mark data with value using text
  DEBUG "!TEXT (PTIME),", DEC x,",1,(black),O ", DEC x,CR
  PAUSE 2000
LOOP
```

(*PTIME*) in the text instruction substitutes the current plot time in seconds for the X coordinate. As always, monitor in the BASIC Stamp Editor's Debug Terminal ensure the format is correct.



Sound Files

StampPlot can play .WAV files to add sound effects to your program. There is an assortment in the StampPlot/Media directory. The sound files are treated just like the drawing instructions. By default StampPlot looks in the media directory for the sound file unless otherwise specified.

~PWAV clap

IWAV may also be used. This instruction will stop the prior wave file to play this one immediately. Compare the following two sets using the CLI:

```
~PWAV clap (CR) ~PWAV boing  
~PWAV clap (CR) ~IWAV boing
```

Can you modify the last program to 'boing' every time a point is plotted?

Image Files

StampPlot has a collection of jpeg (.jpg) image files that can be used to add images to your plot. These are again treated as drawing instructions using coordinates. The media directory is the default directory for images:

```
!POBJ Clear
!NEWP
!PLOT ON
^IMGP 80a,100a,90a,110a,comp\led_red_1.jpg
^IMGP 90a,100a,100a,110a,comp\led_grn_0.jpg
```



Again, the instructions may come from the BASIC Stamp module:

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'Plot digital and place 2 images

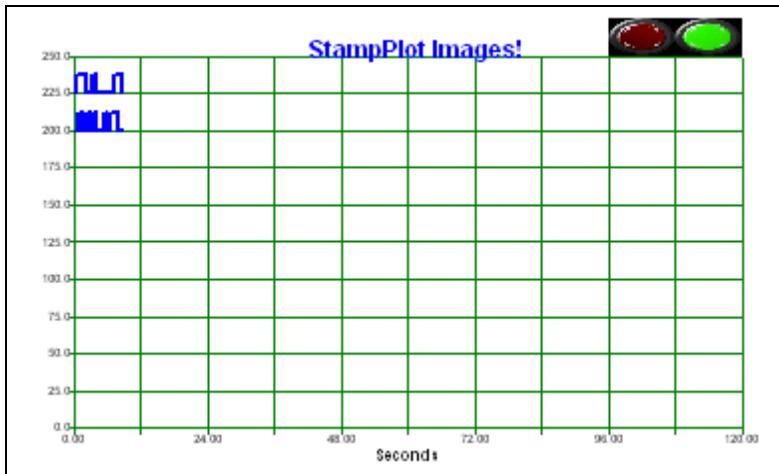
PAUSE 1000

DEBUG CR,"!POBJ Clear",CR
DEBUG "!NEWP",CR
DEBUG "!PLOT ON",CR
DEBUG "@TEXT 35a, 102a, 2, (BLUE), StampPlot Images!", CR

x VAR Byte

DO
  RANDOM x
  ' plot digital values
  DEBUG IBIN2 x,CR
  ' Place LEDs on plot
  DEBUG "^IMGP 80a,100a,90a,110a,comp\led_red_(BIT0).jpg", CR
  DEBUG "^IMGP 90a,100a,100a,110a,comp\led_grn_(BIT1).jpg", CR
  PAUSE 500
LOOP
```

(*BIT0*) and (*BIT1*) are replaced when used by StampPlot with 1 or 0 depending on the status of those bits plotted respectively.



Section 7: Place controls on the interface for monitoring and control.

Creating a Plot Object Control

Controls, such as text boxes, buttons, gauges and so on may be placed on the StampPlot background once the plot is resized.

```
!POBJ Clear  
' Size plot to 70% by 80% of window.  
!PPER 70,80  
!NEWP
```

A new plot object control is created by defining the type, naming it, setting the coordinates (the background is 0,0 to 100,100) and setting parameters.

To create a new meter the format is:

```
?POBJ oMeter.objName=L,T [,W,H,scale min, scale max,alarm min, alarm max]
```

Where *oMeter* means to use a meter control plot object.
ObjName is what you want to name it.

L = Left Coordinate of meter.



T = Top Coordinate of meter.
[] indicates these parameters are optional.
W = Width of the meter.
H = Height of the meter.
Scale min = The minimum value of the meter.
Scale Max = The maximum value of the meter.
Alarm Min = The lower alarm set point.
Alarm Max = The upper alarm set point.

So, to create a meter called meter1 at 75 left and 50 top using default width and height, and range from 0 to 255 with alarm set points at 25 and 200 (ensure you run previous code first):

```
!POBJ oMeter.Meter1=75,50,,,0,255,25,200
```

Meter1 may be updated by giving it a new value (notice how its name is used):

```
!POBJ Meter1= 100
```

BASIC Stamp Code to run the meter may be as follows:

```
'{$$STAMP BS2}
'{$PBASIC 2.5}

'Plot and use meter

PAUSE 1000

DEBUG CR,"!POBJ Clear",CR
DEBUG "!NEWP",CR
DEBUG "!PLOT ON",CR
DEBUG "@TEXT 35a, 102a, 2, (BLUE), StampPlot Meters!", CR

' Size the plot
DEBUG "!PPER 70,80",CR

' Create meter
DEBUG "!POBJ oMeter.Meter1=75,50,,,0,255,25,200",CR

x VAR Byte
x = 100

DO
  RANDOM x
  ' Plot value
  DEBUG DEC x,CR
```

```
' Update meter
DEBUG "!O Meter1=", DEC x,CR
PAUSE 500
LOOP
```

Notice the use of `!O`. This is short hand for `!POBJ` to save typing and code space. Also note the update string must be sent each time the value changes. This can be automated using an Update Value for the control object.

Update Values

StampPlot keeps track of all kinds of different values such as earlier when (`PTIME`) was used for the current plot time, and (`BIT0`) was used for the last digital bit 0 received. These are termed Macro Math Values.

Another is (`AINVAL0`) to (`AINVAL99`) for a set of analog values received. We can set the meter to automatically use the value when updated by setting an Update value:
`!O Meter1.U=(AINVAL0)`

Now, when the instruction to update is issued, the meter will be updated with this value.
`!O Update`

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'Plot with meter and update value

PAUSE 1000

DEBUG CR, "!POBJ Clear",CR
DEBUG "!NEWP",CR
DEBUG "!PLOT ON",CR
DEBUG "@TEXT 35a, 102a, 2, (BLUE), StampPlot Meters!", CR

' Size the plot
DEBUG "!PPER 70,80",CR

' Create meter
DEBUG "!POBJ oMeter.meter1=75,50,,,0,255,25,200",CR

' Set an update value
DEBUG "!O Meter1.U=(AINVAL0)",CR
```



```
x VAR Byte
x = 100

DO
    RANDOM x
    ' Plot value
    DEBUG DEC x,CR
    ' Update all plot object controls
    DEBUG "!O Update",CR
    PAUSE 500
LOOP
```

While this doesn't seem to save a lot, if you had 3 meters, text boxes, and other various controls, you would be able to update them all at once using *!O Update*.

ALARMS – Adding Event Code

Event code is StampPlot code that is ran when an event occurs. An event is a meter hitting an alarm level, a button being clicked, a text box having its text changed, or a variety of other actions that may take place with the plot object controls.

Event code is written by specifying the code to take place for the object, such as our meter:

!O Meter1.C=~PWAV boing

Multiple instructions can be issued by separating them with “**cr**”. If you are using multiple lines, a “**;**” may also be used (you’ll see this in macro text files).

!O Meter1.C=~PWAV boing(CR)!STAT ALARM AT (RTIME)

Where *(RTIME)* is the real time of the plot (the computer’s hours:minutes:seconds).

- ✓ Can you add the event code to the previous program? (hint: It belongs after the meter is created).
- ✓ Please look through the help files on other plot object controls and their use, and of course try some out!

Section 8: Performing Math Operations

StampPlot can perform math operations above and beyond what the BASIC Stamp module can handle, including full floating point math. Here are some rules for using math:

- Math operations are enclosed in square brackets [].

- Only one operation per bracket.
- Math is performed inner bracket to outer bracket.
- Commas separate values and operators.

Take for example:

```
!STAT [100,/,5]
```

This will calculate and display in the status text box the value of $100 / 5$ when entered.

Or, a little more complex:

```
!STAT [[100,/,5],-,20]
```

Will calculate $100/5 - 20$.

The BASIC Stamp module can send data formatted to have math operated on it prior to being plotted or used in other ways:

```
'{$STAMP BS2}
'{$PBASIC 2.5}

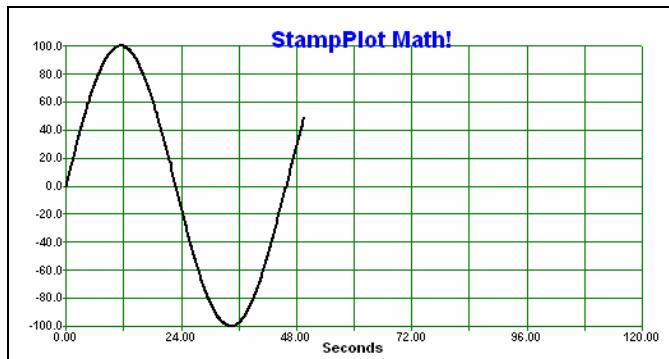
'Plot sine wave

PAUSE 1000

DEBUG CR,"!POBJ Clear",CR
DEBUG "!NEWP",CR
DEBUG "!PLOT ON",CR
DEBUG "!SPAN -100,100",CR
DEBUG "@TEXT 35a, 102a, 2, (BLUE), StampPlot Math!", CR

x VAR Word

DO
  FOR X = 0 TO 360
    ' Plot sine value * 100  [[value,SIN],*,100]
    DEBUG "[[",SDEC x,",",SIN],*,100]",CR
    PAUSE 100
    NEXT
LOOP
```



Section 9: Configure From Macros (Text Scripts)

Of course, all this great stuff of drawing, graphics, sounds, controls and math takes up room on your BASIC Stamp module, and in some cases you may not be able to modify the controller code to perform such operations.

Macros are simply text files that contain configuration information and can be used for the processing of data. Below is a simple macro.

- ✓ Use the Macro menu → Edit Macro with NotePad
- ✓ Provide a name, such as macroTest
- ✓ Answer yes to create (twice).
- ✓ Place text in file.
- ✓ Save file.
- ✓ Use the Macro menu → Run Macro, select the macro and open it.

```
'Macro to plot and show value in meter

INIT:
' Define this macro as the default macro for data
!DEFS (ME)

' Clear all plot objects
!POBJ Clear

'Start a new plot
!NEWP
```

```

' Span the Y-axis
!SPAN -100,100

' Enable plotting
!PLOT ON

' Place graphic text
@TEXT 35a, 102a, 2, (BLUE), StampPlot Meters!

' Size the plot
!PPER 70,80

' Create meter
!O oMeter.Meter1=75,50,,, -100,100
' set update value, format for 2 decimal places
!O Meter1.U=[(AINVAL0),FORMAT,0.00]

' Use default routine when data arrives
!USED ON
ENDMAC

'Routine ran when analog data arrives
DEFAULT:
' Update plot object controls
!O UPDATE
ENDMAC

```

This macro has two routines in it, *INIT* and *DEFAULT* and each ends with *ENDMAC*. *INIT* is run when the macro is opened. By specifying *!USED ON*, the *DEFAULT* routine is ran when analog data arrives. ALL comments must be on separate lines from the code in macros.

Here is a simple PBASIC code snippet test it.

```

x VAR WORD

DO
    FOR X = 0 TO 360
    DEBUG DEC x,CR
    PAUSE 100
    NEXT
LOOP

```

Now, the combination of the Macro and BASIC Stamp plot the values and display in the meter from 0 to 360.

But what if we want the SIN values plotted instead?



Manipulating Analog Data Before Plotting

We can stop StampPlot from automatically plotting the incoming analog data and manipulate the data before plotting it.

- ✓ At the end of the *INIT*: routine, before *ENDMAC*, add this:

```
' Use analog data for macro only - do not plot
!USEA ON
```

- ✓ Change the *DEFAULT* Routine to this:

```
'Routine ran when analog data arrives
DEFAULT:
  ' Plot analog channel 0 in red
  !ACHN 0, [[(AINVAL0), SIN], *, 100], (RED)
  ' Update plot object controls
  !O UPDATE
ENDMAC
```

!ACHN tells StampPlot to plot the value on a channel (0-9) in color specified.
!ACHN 3,100,(BLUE)

Other Analog Data Processing Methods

Besides use of the Default macro, object controls can be used to be trigger event code when analog data, digital data or message data is received. This is performed using a specially structured name for these object controls.

Name begins with:

- DA_ Event code will be processed when analog data arrives.
- DB_ Event code will be processed when digital data arrives.
- DM_ Event code will be processed when messages for the message list are received.

An example for analog data:

```
' DA_Hidden -- OBUTTON *****
!POBJ oButton.DA_Hidden=72.,73.,10.,5.,Obj10,8
!POBJ DA_Hidden.V=0
'-- Event Code
!POBJ DA_Hidden.C='Local Channels();'
();
'Plot on channel 0 1st byte();
^AchN 0, [(AINVAL0),*,0.02], (ORANGE) ();
();
'plot on channel 1 2nd byte();
^AchN 1, [(AINVAL1),*,0.02], (RED) ();
();
'plot on channel 2 3rd byte();
^AchN 2, [(AINVAL2),*,0.02], (BLUE) ();
!IFTH [(AINVAL1),==,(AINVAL2)],==,1,~pwav beep();
!POBJ UPDATE
```

A button is placed on the plot and named DA_Hidden.

The button is set to be hidden – not visible (*DA_Hidden.V=0*)

When analog data arrives the event code will be ran and 3 channels of analog data will be scaled by multiplying each value by 0.2 and plotted in Orange, Red and Blue.

If analog channel 1 is equal to analog channel 2 (the logical result of comparing them, 0 or 1, is equal to 1 the beep wav is played.

The plot objects are updated with current values.

One other way to trigger code on arrival of data is the use of the oAnalog object introduced in StampPlot Version 3 Release 2.

oAnalog.C = ~PWAV beep();

!STAT Value is (AINVAL0)

Macros (.spm files) may be opened directly by double-clicking them from Windows Explorer or from an Internet link. This will load StampPlot and bring up the macro.

Section 10: Use an interface for interactive control of your BASIC Stamp module.

The *!READ* instruction may be used to send data BACK to the BASIC Stamp module where it would accept and use the value. Treating their name like a macro math value

references object values. For example, a slider control named ‘Setpoint’ could have its value sent to the BASIC Stamp using:
!READ (Setpoint)

We will have the BASIC Stamp module create and read this interface with a slider with a range from 0 to 255 and adjust a value to match the slider’s.

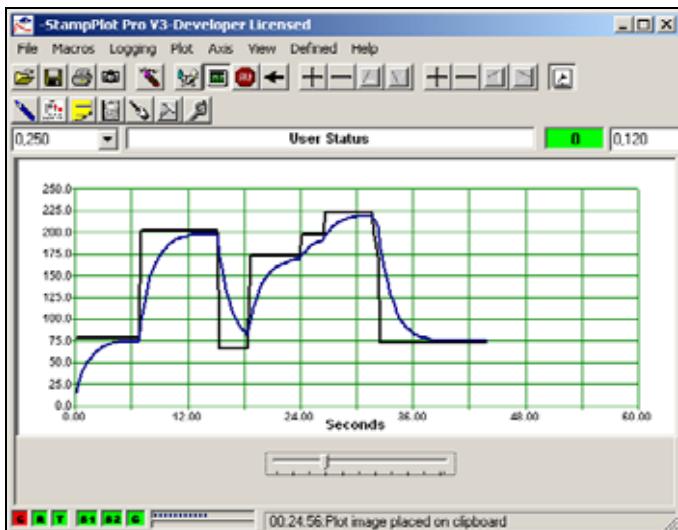


```
'{$STAMP BS2}
'{$PBASIC 2.5}

' Interactive control with a StampPlot slider
PAUSE 1000
DEBUG CR,"!POBJ Clear",CR
DEBUG "!NEWP",CR
DEBUG "!PPER 100,80",CR
' create slider on StampPlot named Setpoint
DEBUG "!POBJ oHSlider.Setpoint=38,15,29,7,0,255,78",CR

Actual    VAR Byte
Setp      VAR Byte

DO
  ' Request value from StampPlot
  DEBUG "!READ (Setpoint)",CR
  ' Accept value from StampPlot
  DEBUGIN DEC Setp
  ' Compare and adjust actual
  IF Actual < Setp THEN Actual = Actual + ((Setp-Actual)/5)
  IF Actual > Setp THEN Actual = Actual - ((Actual - Setp)/5)
  ' Plot setpoint and actual
  DEBUG DEC Setp, "", DEC Actual,CR
  PAUSE 250
LOOP
```



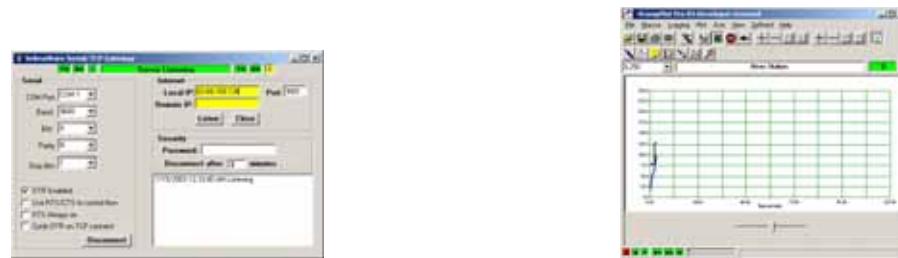
Note: It is recommended to use the PBASIC command **SERIN** with a timeout instead of **DEBUGIN** instruction. With **DEBUGIN**, if a response is not received (StampPlot not connected) the controller will ‘hang’ waiting for a return.

```
SERIN 16, 84,500,Timeout, [DEC Setp]  
Timeout:
```

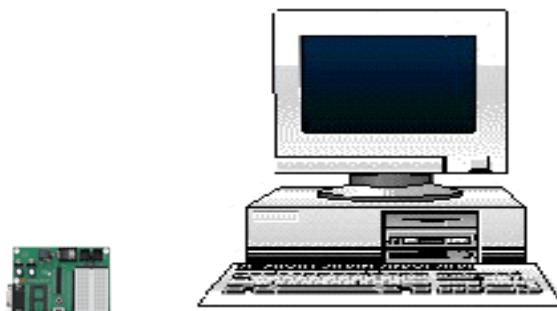
Another fun use is image buttons that display of one of 2 graphics when clicked and return a 1 or 0 when read and can be used as virtual switches. See the help files!

Section 11: Perform acquisition and control over the Internet.

StampPlot also installs a program called the TCP-Serial Gateway. This program allows monitoring and control over the Internet by creating a link (gateway) between your serial port and your Internet connection. The typical configuration is shown:



C

**Local – BASIC Stamp/Gateway****Remote – StampPlot**

Local Configuration:

On the local computer, the BASIC Stamp module is connected and programmed for use with StampPlot (most any program will do).

- ✓ Open the TCP-IP Gateway program.
- ✓ Connect to the serial port (select cycle DTR if you want your BASIC Stamp module to reset on a connection).
- ✓ Connect Listen on the Internet side.
- ✓ Record the Local IP address shown.

Remote Configuration:

- ✓ Open StampPlot
- ✓ Open the Configuration Window.
- ✓ On the Port Tab, select TCP.
- ✓ Enter the IP address from above.

- ✓ Click the Connect and Plot buttons.
- ✓ Watch it plot and say woooooooooooo.

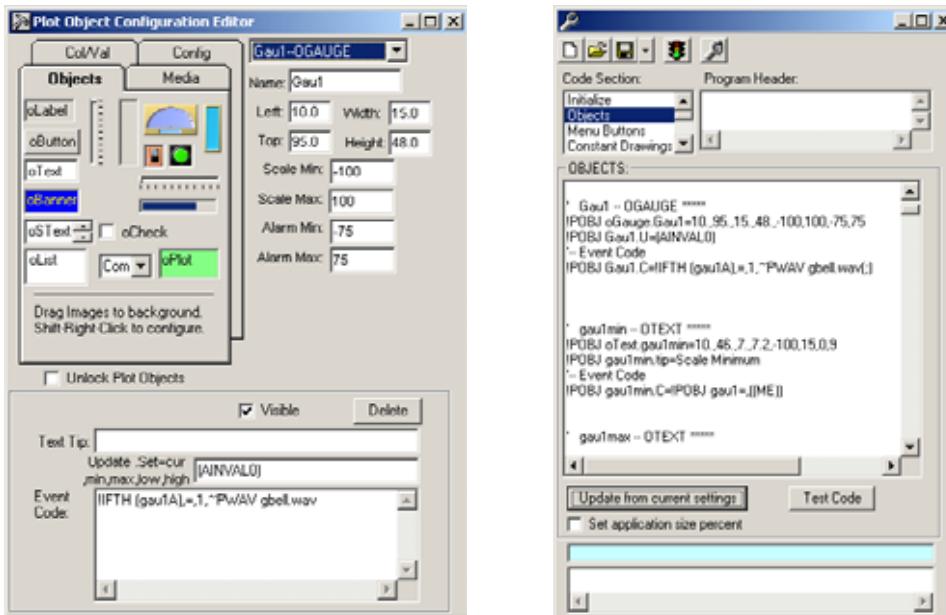
Interactive control works also!

If you have problems connecting, check with your network administrator, there may be a firewall. StampPlot TCP gateway does not support proxy-servers currently.

One hint is if you are trying to perform interactive control by sending data back to the BASIC Stamp module is to increase the timeout values of **SERIN** since the connection will not be as fast as straight serial. A good value is 500 to 1000 for timeout.

Developers License

A Developers license allows the use of the Plot Object and Macro Editor to Drag-and-Drop objects, program parameters and event code and automatically build macros from the current configuration and is available through Parallax.



This guide touched briefly on many aspects of StampPlot. The help files have plenty of more great information on StampPlot Pro.



Licenses available at:

www.parallax.com
www.stampplot.com
and many other distributors of BASIC Stamp microcontrollers.

For support needs concerning StampPlot please contact:

support@selmaware.com

Or join our Yahoo Group at:

<http://groups.yahoo.com/group/selmaware/>

Happy Plotting!
Martin Hebel
SelmaWare Solutions

Appendix D: Parts Listing

System and Software Requirements

- PC running Windows® 98 or higher operating system.
- Available serial port
- BASIC Stamp Editor for Windows v2.0 or higher
- StampPlot Pro v3 release 2.15 or higher, with a free Home/Educational Institution License.
- StampPlot Experiments with Renewable Energy macros
- Optional: BASIC Stamp source code for the experiments in this book.

E

The BASIC Stamp Editor, StampPlot Pro, and macros are on the Parallax CD included in your kit. You may check for the latest versions at www.parallax.com. The BASIC Stamp Editor software can be found on the Downloads menu. The StampPlot Pro software and macros, and the BASIC Stamp source code for this book, can be found on the Experiments with Renewable Energy product page under Education → Stamps in Class tutorials.

Parts Listing

Some components used in the Experiments with Renewable Energy Kit are readily available from common electronic suppliers. Customers who would like to purchase a complete kit may also do so through Parallax. To perform the experiments in this text, you need three hardware items: (1) a BASIC Stamp 2 module (available alone, or in the Board of Education - Full Kit); (2) a Board of Education (available alone or in a Board of Education Full Kit); and 3) the Experiments with Renewable Energy Parts Kit. The typical setup consists of the Board of Education Full Kit and the Experiments with Renewable Energy Parts Kit. In addition, several household items are necessary, listed on page 288.

Board of Education Kit

The BASIC Stamp 2 (BS2-IC) is available separately or in the Board of Education Full Kit. Individual pieces may also be ordered using the Parallax stock codes shown below.

Board of Education Full Kit (#28102)		
Parts and quantities subject to change without notice		
Parallax Code#	Description	Quantity
28150	Board of Education	1
800-00016	Pluggable wires	10
BS2-IC	BASIC Stamp 2 module	1
800-00003	Serial cable	1

The Experiments with Renewable Energy Parts and Text Kit (#28145)

Similar to most Stamps in Class Student Guides, you need a BASIC Stamp 2 module with a Board of Education carrier board and the Parts Kit. The content of the Experiments with Renewable Energy Parts Kit is listed below. These replacement parts are available from Parallax but may also be sourced from common electronic suppliers. In an effort to provide the best quality and most up-to-date products for our customers, you may find that some of the actual parts received with this kit may differ from those listed in the table.

Experiments with Renewable Energy Kit		
Parts and Text(#28145) Parts Only(#28144)		
Parts and quantities subject to change without notice		
Part #	Description	Quantity
150-01000	10 Ω resistors 1/8 W 5%	2
150-01020	1 kΩ resistors ¼ W 5%	2
150-02210	220 Ω resistors ¼ W 5%	3
150-04710	470 Ω resistors ¼ W 5%	3
201-01080	1000 µF capacitor	1
27000	Parallax CD	1
350-00001	Green LED	1
350-00006	Red LED	2
350-00007	Yellow LED	1
451-00303	3-pin male/male header	1
500-00001	Transistor 2N3904	2
501-00005	Small signal diode 1N4148	6
556-28144	Three-Phase Wind Turbine Kit	1
604-00027	4-channel A/D converter (ADC0834)	1
700-00003	4-40 nuts	2
700-00028	1/4" 4-40 machine screw	1
700-00059	Lock washer #4	1
700-00064	Parallax screwdriver	1
70011	Experiments with Renewable Energy Text	1
710-00007	7/8" 4-40 machine screw	2
712-00002	Metal washer #4	2
720-28144	1 ½ " hexagonal standoff, predrilled	1
727-27307	Solar Power Kit of Science	1
752-00001	Ni-Cad rechargeable AAA Batteries	2
753-00002	Battery holder w/ tinned leads	1
800-00016	Jumper wires, 3" (bag of 10)	2
800-28144	30" jumper wire w/ 1 tinned lead, set of 3	1
900-00008	Parallax Continuous Rotation Servo	1

E

Additional Items

You will also need several common tools and household items that are not included in your kit:

- Masking tape
 - Electrical tape
 - Cellophane tape
 - Cardboard or heavy paper
 - Scissors
 - 7/16" wrench
 - Wire stripper
 - Diagonal cutters (or nail clippers!)
 - WD-40 or similar lubricant
 - Light source such as a desk lamp
 - Electric desk fan
 - Hand drill with a #36 bit
- OR-
- Hobby knife (such as an X-acto® knife) with a slim, sharp tip

Appendix E: Resistor Color Code

Most common types of resistors have colored bands that indicate their value. The resistors that we're using in this series of experiments are typically "1/4 watt, carbon film, with a 5% tolerance". If you look closely at the sequence of bands you'll notice that one of the bands (on an end) is gold. This is band #4, & the gold color designates that it has a 5% tolerance.

The resistor color code is an industry standard in recognizing the value of resistance of a resistor. Each color band represents a number and the order of the color band will represent a number value. The first two color bands indicate a number. The third color band indicates the multiplier or in other words the number of zeros. The fourth band indicates the tolerance of the resistor +/- 5, 10 or 20 %.

Color	1st Digit	2nd Digit	Multiplier	Tolerance
black	0	0	1	
brown	1	1	10	
red	2	2	100	
orange	3	3	1,000	
yellow	4	4	10,000	
green	5	5	100,000	
blue	6	6	1,000,000	
violet	7	7	10,000,000	
gray	8	8	100,000,000	
white	9	9	1,000,000,000	
gold				5%
silver				10%
no color				20%

A resistor has the following color bands:

- Band #1. = Red
- Band #2. = Violet
- Band #3. = Yellow
- Band #4. = Gold

Looking at our chart above, we see that Red has a value of 2.

So we write: "2".
Violet has a value of 7.
So we write: "27"

Yellow has a value of 4.
So we write: "27 and four zeros" or "270000".

This resistor has a value of 270,000 Ω s (or 270K Ω) & a tolerance of 5%.

Index

- \$ -

\$PBASIC directive, 15
\$STAMP directive, 15

- A -

A/D converter, 27, 31
AC. *See* alternating current
AC/DC conversion, 7
alternating current, 5–8
 benefits of, 7
 definition of, 6
 rectification to, 139

ampere, 3
amplitude, 6
Anacon Systems, Inc, 223
anode, 44, 45

- B -

band-gap energy, 88
BASIC Stamp Editor, 10
BASIC Stamp[®] 2 microcontroller, 10
batteries
 alkaline, 81
 deep cycle, 80
 marine, 80
 memory effect, 47
 NiCad, 44
 overcharging, 46
 rechargeable battery care, 47, 53
Berkeley National Laboratory, 87

Board of Education Full Kit, 285
Board of Education[®] carrier board, 10
boron, 84

- C -

cadmium, 45
cadmium hydroxide, 46
calculating power output, 186
Canada, 227
Capacitor
 Polar – identifying terminals, 158
Cape Cod, 230
cathode, 44, 45
Center for Power Electronic Systems,
 177
charge, 3
chemical energy, 4
Clarke, Arthur C., 83
closed-loop control system, 134
Conservation of Energy principle, 1
CPES. *See* Center for Power Electronic
 Systems
CPS. *See* cycles per second
cycles per second, 6

- D -

Darrieus wind turbine, 226
DC. *See* direct current
declination formula, 136
delta coil, 186
Denmark, 225
diode
 forward-biasing, 139
germanium, 140

reversed-biased, 141
silicon, 140

direct current, 1–3
 benefits of, 7
 definition of, 5

dopants, 84

Dranetz-BMI 355 Harmonics Analyzer, 224

- E -

EagleDrive 3 motor, 223
Edison, Thomas, 5
EEPROM, 193
electrical energy, 4
electricity, methods of generating, 5
electrode, 44
electrolyte, 44
electromagnetic spectrum, 86
electron, 3
ENECO, 232
energy
 chemical, 4
 conversion to electricity, methods of, 5
 definition of, 1–3
 electrical, 4
 kinetic, 3
 mechanical, 4
 nonrenewable, 4
 nuclear, 4
 potential, 3
 radian, 4
 renewable, 4

sources, 4
thermal, 4

Equator, 138
Equinox, 136
Experiments with Renewable Energy Parts Kit, 286

- F -

Faraday, Michael, 179
Finland, 227
full-wave rectification, 163

- G -

gallium, 84
germanium, 140
Germany, 225

- H -

half-wave rectification, 141
hardware required
 Board of Education, 10
 three-phase AC alternator (wind turbine), 11

hardware requirements, 285
 additional household items, 288

hertz, 6
Hertz, Gustav, 6
horizontal axis, 225
Horns Rev Danish Wind Farm, 231
hydrogen, 46
hydrogen manufacturing, 114
Hz. See hertz

- I -

indium, 84
indium nitride, 87

infrared, 86
infrared spectrum, 86
International Space Station, 110
inverter, 8
ion, 44

- J -

joule, 2

- K -

kilowatt-hour, 2
kinetic energy, 3
kWh
see kilowatt-hour, 2

- L -

latitude, 138
LearnOnLine. *See* Appendix A
LearnOnLine, Inc, 178
linear power supplies, 172

- M -

magnet warning, 143
mechanical energy, 4
Mill Creek Hydroelectric Plant, 221

- N -

Nanosys, Inc, 87
National Semiconductor, 176
NativeEnergy, 229
neodymium magnets
warning, 143
newton, 2
nickelic hydroxide, 45
nickelous hydroxide, 45
nuclear energy, 4

- O -

ohm, 3
Ohm's Law, 3, 184
open-loop control system, 134
oxidation, 44
oxygen, 46

- P -

parabolic panels, 113
photovoltaic cell. *See* solar cells
potassium hydroxide, 45
potential energy, 3
Potts, Michael, 88
Poulek-Solar, 112
power electronic systems, 176
power output calculation, 183, 186
Power Sources Manufacturers Association, 178
power supply design, 176
switching power supplies, 175
power supply technologies, 8
PSMA. *See* Power Sources Manufacturers Association
power supply design
linear power supplies, 172

- R -

radiant energy, 4
rectification
definition, 139
full-wave, 139, 163
half-wave, 139, 141
three-phase power, 187–90
reduction, 44
REEL Power Project. *See* Appendix A

resistor color code, 289
ripple, 174
Rosebud Sioux Tribe Wind Turbine Project, 229

- S -

sampling, 192
Savonius wind turbine, 228
semiconductor, 84
SHEC-Labs Hydrogen Generator, 113
silicon, 84
Single-phase to three-phase Converter, 223
software required
 BASIC Stamp Editor, 10
 source code, 14
 StampPlot Pro, 20
software requirements, 285
solar azimuth and declination (diagram), 136
solar cells
 commercial applications, 106–8
 efficiency, 86
 future, 87
 history, 84
 structure (diagram), 85
Solstice, 137
Spain, 225
Spar-WARP™ Wind Machine, 232
StampPlot Pro
 installation, 24
 tutorial. See Appendix C
StampPlot Pro, 20

stator, 143
Stirling Energy Systems, 114
sunlight, 84
sun-tracking systems, 109–16

 detriments to, 115
 dual-axis, 112
 financial considerations, 115
 history, 109
 parabolic panels, 113
 passive trackers, 113
 single axis, 112

switching power supplies, 175
synchronous serial, 27

- T -

Tesla, Nikola, 5, 180
 polyphase patents, 7
thermal energy, 4
Thomson-Houston, 7
three-phase AC alternator
 magnet warning, 143
three-phase power, 179–90
 A/C output (diagram), 180
 commercial applications, 224
 creating, 180
 delta coil, 182
 history, 221
 rectification, 187–90
 sampling, 192
 wye coil arrangement, 181

transistors, 53, 54
- U -
ultraviolet, 86
- V -
vertical axis, 225
volt, 3
- W -
W. See watt
watt, 6
Watt, James, 6
WebBench, 175
Westinghouse, 7
wind farms, 228
wind turbine
commercial applications, 232
Darrieus, 226
eggbeater, 226
horizontal axis, 225
magnet layout, 181
Savonius, 228
vertical axis, 225
Windmill Tours, 230
work
definition of, 2
- Z -
Zomeworks Corp, 113

sensors!

Expand the capabilities of your next BASIC Stamp project with a sensor from Parallax. We stock an entire line of sensors that are compatible with the BASIC Stamp microcontroller.

Clockwise from top:

- TAOS TCS230 Color Sensor (#30054)
- Memsic 2125 Dual-Axis Accelerometer (#28017)
- Sensirion SHTX Humidity Sensor (#28018)
- FlexiForce Pressure Sensor (#30056)

For these and other sensors visit the Component Shop at www.parallax.com

PARALLAX



Measure and Interpret Signals

The Parallax Digital Oscilloscope with Understanding Signals text and components (order #28119). This new scope is available exclusively through Parallax and includes complete BASIC Stamp support. Learn how to view, measure, and generate signals used for servo control, sine waves, op-amp buffers, voltage amplifiers, infrared emitters, and synchronous and asynchronous serial communication.

Order #28119

Screen shot of software.

PARALLAX
www.parallax.com/sic

Go to the head of the class!

Explore microcontroller programming and interfacing
with any of our popular Stamps in Class Curricula.



What's a Microcontroller? introduces BASIC Stamp programming, circuit building, and electronics with fun, multi-sensory experiments. This is the gateway text to Stamps in Class.



Elements of Digital Logic will immerse students into a sight, sound, and hands-on logic environment. This curriculum utilizes both hardware and software simultaneously to teach logic.



The *Understanding Signals* kit shows students how to view and analyze signals with the Optriscope. A perfect companion kit, the example circuits are drawn from other Stamps in Class texts.



Basic Analog and Digital explores the principles of interfacing analog devices to the digital BASIC Stamp. Measure and control real-world conditions with A-to-D and D-to-A conversions.



Robotics with the Boe-Bot mounts a Board of Education onto a robot chassis. Learn programming as you teach your Boe-Bot to navigate autonomously with multiple sensor circuits.



Applied Sensors (formerly *Earth Measurements*) covers program structuring, sensor calibration, EEPROM data logging, conductivity of water, closed-loop feedback, and debugging in an earth science format.

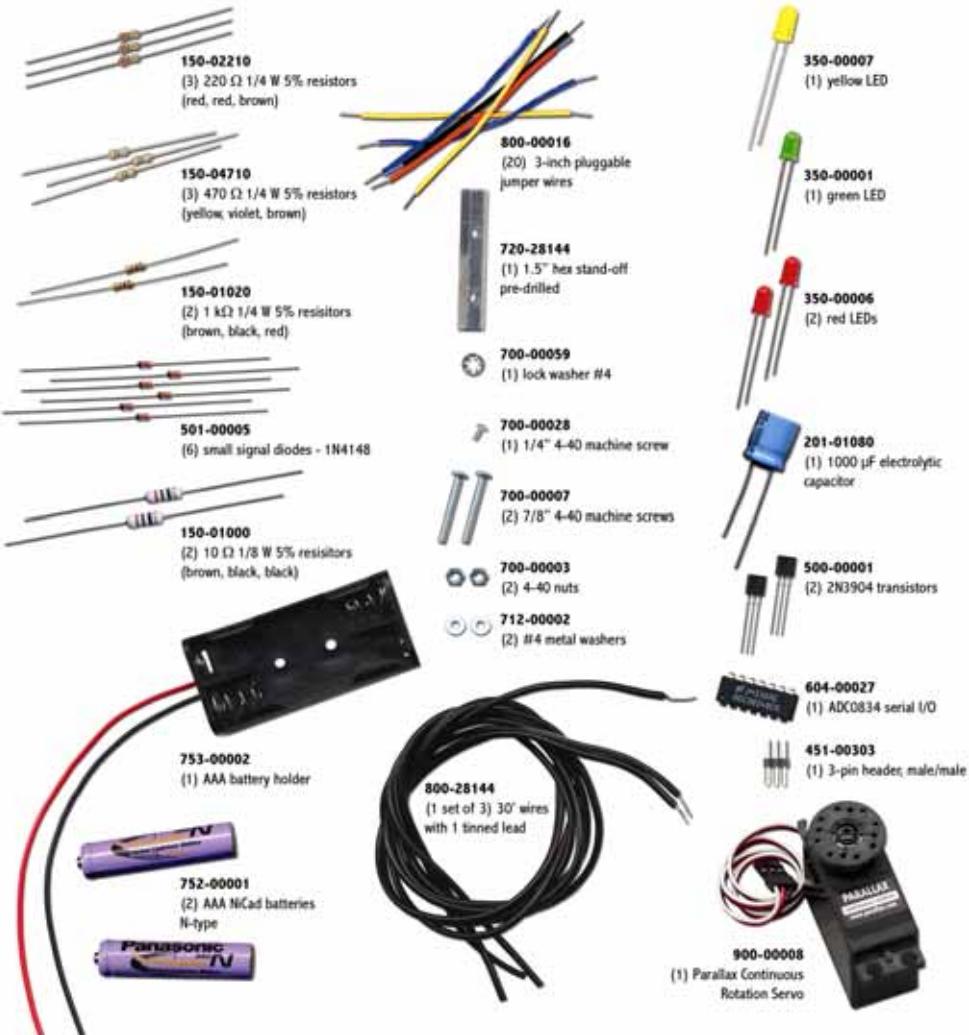


Process Control introduces aspects of industrial system automation with sensors, mechanical and digital switches, with on-off, differential gap, and PID control methods.



Advanced Robotics with the Toddler will teach you how to build and program a high-quality machined two-servo bipedal walking robot controlled by an embedded BASIC Stamp 2.

PARALLAX www.stampsinclass.com



NOT SHOWN: 3-Phase Alternator (Wind Turbine) Kit, and Solar Power Kit of Science.

Parts and quantities in the Experiments with Renewable Energy 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.