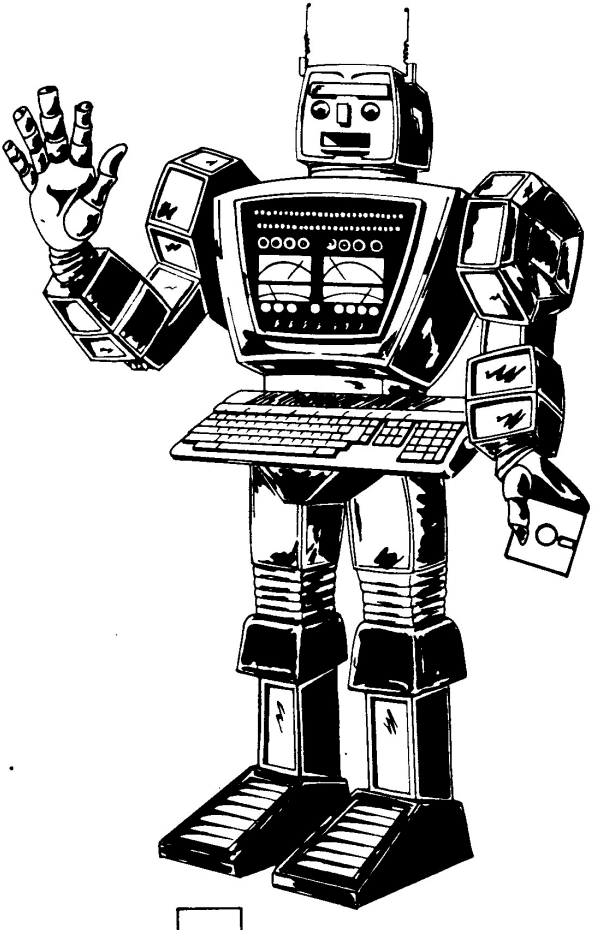
YOUR ATARI 8 BIT COMES ALIVE!



DISK INCLUDED

Written by Richard Leinecker

Edited by David Leinecker

Computer Spectrum, Inc.

Your Atari 8 Bit Comes Alive!

Written by

Richard C. Leinecker

Edited by

David B. Leinecker

Introduction

The purpose of this book is to enhance the understanding and enjoyment of the Atari 8 Bit computer. By building hardware projects, the user develops a better understanding of the machine in general, and at the same time, learns some basic interfacing techniques. These projects are geared towards educating Atari owners and not necessarily for the design engineer. As a result, many of the circuits and programs are simple so that they are easily understood. While the electronics novice will benefit from the contents of this book, the advanced readers will learn as well. Every project can be directly applied and used, but they can all be elaborated upon to develop more complex applications.

We are planning updates and other publications in the future. If you have an idea that you would like us to consider, please let us know.

While reasonable care has been exercised with regard to the accuracy of this book, the authors and publisher assume no responsibility for errors, omissions, or suitability for any applications. Neither do we assume any liablity for any damage resulting from the use of this book.

We ask for your feedback and offer our support in your endeavors. If you have a question or comment, please feel free to call our BBS at 305-251-1925.

Acknowledgements:

The author, editor, and publishers wish to thank Jack Durre' for his invaluable support and advice. We would also like to thank Tammy Leinecker for many hours spent proofreading and typing. A word of gratitude is extended to the Dade Atari Users Group (DAUG) for allowing these ideas to be presented over the last several months.

Table of Contents

1. Building and Testing

For many people, this section contains information they may already know. For them, it can either be used as an electronics review or as a reference section. For those who have little or no background in these subjects, it would be wise to spend some time reading this section.

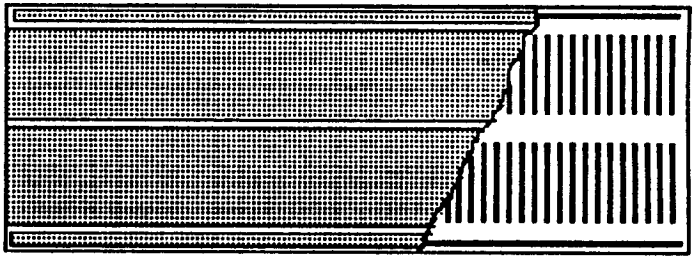
BUILDING and TESTING

The two most effective ways to build these projects are on solderless breadboards and perforated breadboards. The use of solderless breadboards is best when the circuit is first constructed so that modifications to the circuit and corrections can be made easily. To use the solderless breadboards, you simply insert components into holes which have underlying conductor strips. The components are easily inserted and removed, so construction is quick and easy. Perforated breadboards are very good once a project is ready for a permanent version. The components in the perforated breadboards can either be soldered or wire-wrapped. If you are especially bold, you may try to etch your own printed circuit board or have one of the many available companies make it.

Solderless breadboards can be purchased from almost any electronic parts supplier. Radio Shack may be the most convenient

because there is no delay time in ordering. The size of the breadboard needed depends upon the number of components there needs to be room for. But, in general, about 90-100 terminal points will be adequate. Some of the more complex projects such as the phone answering circuit may require more room. The best alternative in this case is to use two boards of the same size. Since they have the ability to interlock with each other, you will have enough space. Jumper wire kits can be purchased from some suppliers. This is a convenient way of obtaining the needed jumpers. If you don't want to buy the jumper kit, simply take some time to strip wires at each of their ends. Then, store these home-made leads in one place. It is much better to have jumpers prepared than to make them as you go.

To place parts on a solderless breadboard, push each lead into a hole. Underneath every five consecutive holes (depending on the type), there will be an underlying conductor. Be careful that you know how the underlying conductors are lined up inside of the solderless breadboard. On the following page is an illustration of a small board with part of it cut away exposing the underlying conductors.



Perforated breadboards can also be purchased through any electronics parts supplier. They are less expensive than the solderless breadboards. Here, just as with the solderless breadboard, a collection of jumper wires is very helpful. If you plan to solder the components, make sure that you have sockets for the ICs and transistors used. Before you begin the construction, it may be helpful to sketch the layout and pencil in the planned component locations. This will make for a compact and neat board. Some suppliers have push-in terminals that allow for easier soldering. These push-in terminals go into the breadboard and provide a convenient solder point. If you choose to wire-wrap the project, the same guidelines will apply.

When you solder, there are several suggestions that you should follow; make sure that all surfaces are clean.

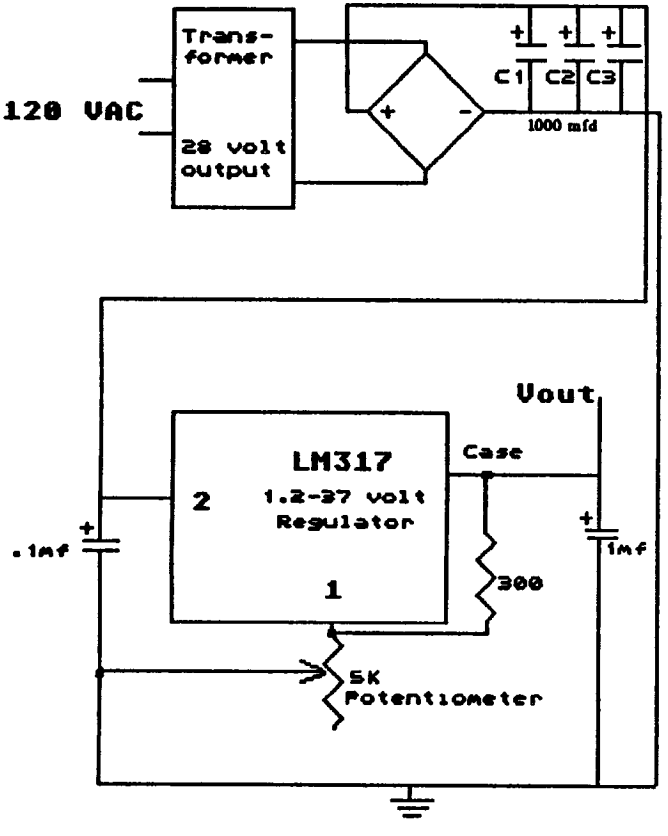
Do not use a wire with an oxidized metal end. If it is oxidized, you must cut the wire and restrip the end. Use small, rosin core solder. Heat the work (the component), then apply the solder. If the solder does not melt when applied to the work, the work is not hot enough.

Soldering any electrical components should be done with care. Besides making the appearance more pleasant, circuit branches may be more easily seen. Limit soldering time whenever possible. For parts which are easily damaged, you should use a heat sink that clips on. This will help you keep your components healthy. A low-power soldering iron from 15 to 35 watts should be used.

If you decide you want to use an etched PC board, you can get the necessary chemicals, boards, and ruboffs (transfers) from most parts suppliers. Radio Shack sells copper-clad boards, dry transfers, etchant resistant pens, and the etchant. From these ingredients, an adequate board can be made. If you want a professional job, you can design the artwork, then send it off and have the PC board made. Appendix A has a listing of companies that offer this service.

Two pieces of test equipment you will need are a multimeter and a logic probe. The multimeter should have voltage ranges of 0-5 volts, 0-25 volts, and 0-125 volts. Two suggested current ranges are 0-100 mA and 0-500 mA. If you can spend the extra money, a current range of up to one amp is sometimes helpful. You should be able to measure resistances from 0 to 6 megohms and have an accuracy to within 5%. The logic probe you choose should be able to handle both TTL and CMOS levels, indicate pulses, and protect against reversed polarity.

One more piece of equipment that makes things easier for the experimenter is a power supply. A combination 5 volt/12 volt power supply is fine for the projects in this book, but a variable range power supply will give more flexiblity in later endeavors. Your power supply should be able to handle at least one amp. If you prefer, you could construct your own. On the next page is the schematic for a variable power supply. You could put two separate packages together for two separate voltage levels.


Trans-
former
120 VAC
28 volt
output
+
-
C1
C2
C3
1000 mfd
Vout
LM317
1.2-37 volt
Regulator
2
1
Case
.1mf
1mf
300
5K
Potentiometer

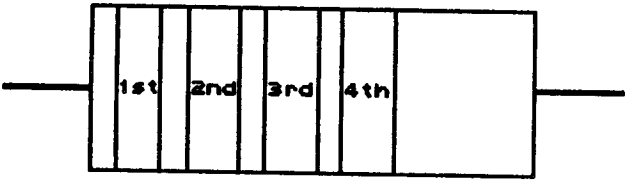
RESISTORS

Resistors limit the amount of current in a circuit branch. The voltage in the branch is divided proportionally across all resistors. Knowing this allows us to calculate the voltage in every point throughout. The basis for these calculations is Ohm's Law, which is E=IxR where E is the voltage in volts, I is the current in amps, and R is the resistance in ohms. For different situations, any one of the three variables may be unknown which leads us to two other useful forms of the equation: 1) **I = E / R** and 2) **R = E / I.**

Suppose that a certain component requires 5 volts and draws 2 amps of current. If the source voltage is 12 volts, you can figure out the needed resistance to drop the 12 volts to 5 volts. The voltage drop is 7 volts (12 volts source - 5 volts device). The appropriate resistor can be found by using Ohm's Law in the form R = E / I or 7 volts / 2 amps = 3.5 Ohms.

One important consideration when using resistors is the power rating. Most circuits in this book only need resistors with 1/4 watt power rating. If you suspect that a resistance may be subjected to too much current, use one of the following formulas to calculate the wattage to which the part will be subjected.

$P = E \times I$ $P = I^2 \times R$ $P = E^2/R$



Most resistors are color coded according to their value. Each of the first two bands represent a digit according to the following chart.

Color	Digit
Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

The third digit represents the multiplier. You multiply the first two digits by this number to get the total resistance. The following chart illustrates these values.

Color	Multiplier
Black	1
Brown	10
Red	100
Orange	1,000
Yellow	10,000
Green	100,000
Blue	1,000,000
Violet	10,000,000
Gray	100,000,000

"K" means 1,000 when it is used with respect to resistance values. A 1.2K resistor has a resistance of 1,200 ohms. Meg means 1,000,000 when used with resistance values. A 2 megohm resistor has a resistance of 2,000,000 ohms.

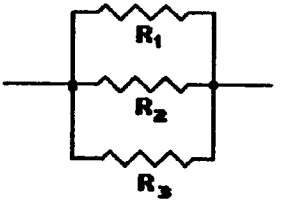
If a fourth band is present, it indicates the tolerance. A gold band means a tolerance of +/-5%, a silver band means a tolerance of +/-10%. If there is no fourth band, then a tolerance of +/-20% can be assumed.

The total resistance of resistors in series, as pictured below, can be found by simply adding their values.

$$R_1 + R_2 + R_3 = R_t$$

When resistors are in parallel, as pictured below, their total resistance can be found using the following formula.

$$\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} = \frac{1}{R_t}$$



CAPACITANCE

In this book, capacitors are used to pass an AC signal and block the DC current, to filter out unwanted voltage fluctuations, and to perform a timing function. Only two different types are used; electrolytic and disk.

When you need to pass an AC signal and block the DC current, consideration must be given to insure that the AC signal will not be significantly attenuated. This can be done by calculating a capacitance value based on the lowest frequency you wish to pass. The following formula, where C is capacitance in farads and f is frequency in cycles per second, will allow you to find this capacitance value.

$$C = \frac{1}{2 \times 3.14 \times f \times 1{,}000}$$

To filter out unwanted fluctuations such as in a power supply, use the same formula as above. For half-wave rectifiers, the frequency will be 60. For full wave rectifiers, the frequency will be 120.

Capacitors do not charge instantaneously nor do they discharge instantaneously. Often, it is important to get a close approximation of this time period. Use the following formula, with resistance values in ohms and capacitance values in farads, to find the time in seconds.

Time = Resistance x Capacitance

If the capacitor in the schematic has a polarity marking such as -||- , it is an electrolytic capacitor and care must be taken to observe the polarity when constructing the circuit. Both the polarity and the value will usually be written on the case of an electrolytic capacitor. A maximum voltage will be written on the case. Do not exceed this voltage as the capacitor will be damaged. Less voltage than the maximum rating is fine.

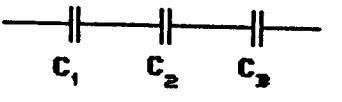
For the disk capacitors, the values have a code which can be interpreted as follows.

X	X	X - 3 digit code
1st	2nd	multiplier

Read the first and second digits as they appear. To get the value in picofarads, simply move the decimal point to the right the number of places indicated in the multiplier. A picofarad is a millionth of a millionth of a farad. To convert from picofarads to microfarads, move the decimal six places to the left. For most applications, a 20% tolerance is fine. Many substitutions will work as long as you stay within these limits.

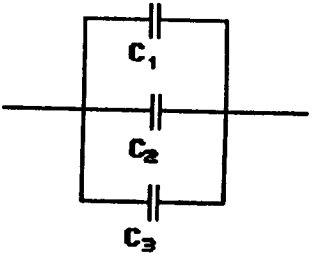
The total capacitance of capacitors in parallel is their sum.

$$\frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3} = \frac{1}{C_t}$$



The total capacitance of capacitors in series can be found by the following formula.

$$C_1 + C_2 + C_3 = C_t$$



DIODES

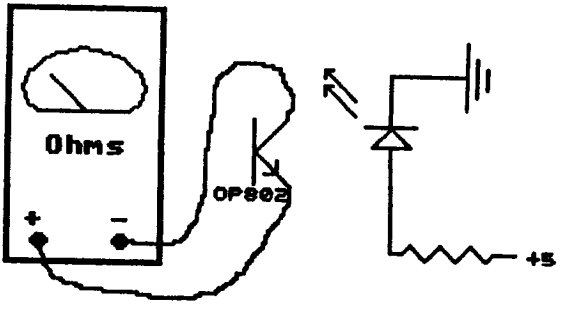
Diodes permit current to flow in only one direction. In this book, the two types of diodes used are rectifier diodes and light emitting diodes (LEDs).

The diodes you use will have a black, or dark, band at one end. This indicates the cathode and should be connected to the negative node in the circuit. This is called forward biasing, and will allow current to flow. The LEDs also have a means of indicating which lead is the cathode and which lead is the anode. The cathode lead will be the shorter one, and the anode will be the longer one. Connect the cathode to negative and the anode to positive for forward biasing.

The LEDs in general require a voltage of about 2 volts and usually draw about 20 mA of current (.02 amps). Exceeding these ratings will probably damage the LED. If your source voltage is greater than 2 volts, use Ohm's Law to select the correct resistor to drop the voltage to safe levels. If your source is 5 volts, the voltage drop needed is 3 volts. Since the LED will draw about 20 mA of current, set up the formula R = E/I or, in terms of the given values, R = 3/.02. Thus, the needed resistance is 1500 Ohms.

The infrared LEDs will be biased in the same way, but be certain to check the manufacturer's specifications as to the current so that you can accurately calculate the voltage-dropping resistor. Since infrared LEDs cannot be seen, you can test their

operation with a phototransitor in the following configuration.

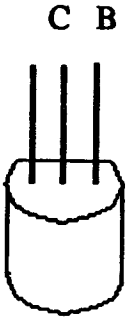


TRANSISTORS

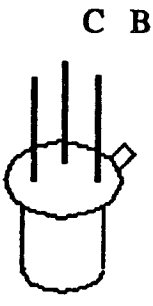
In the age of ICs, transistors still have a place. A single, general purpose transistor can be used for a wide variety of purposes. This makes them very flexible. A single transistor takes up less space than an IC and may be more practical when only one gate is needed.

Bipolar transistors (NPN or PNP) usually have three leads. They are the emitter, the base, and the collector leads. Following is an illustration, looking at it from the bottom, with leads pointing up. If it is a plastic package, one side will be flat. This is how you identify the positions. If it is a metal package, the small tab on the case is the reference used to identify the positions.

View of Bottom:



Plastic Case



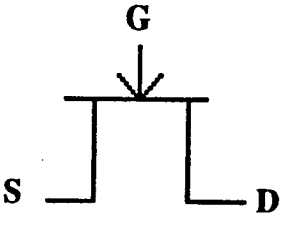
Metal Case

The schematic representation of NPN and PNP transistors is as follows.

NPN

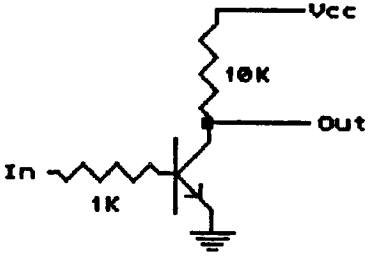
PNP

FET (or JFET) transistors have no standard format for their leads. The three leads will be the source (S), the gate (G), and the drain (D). The schematic representation is on the following page.

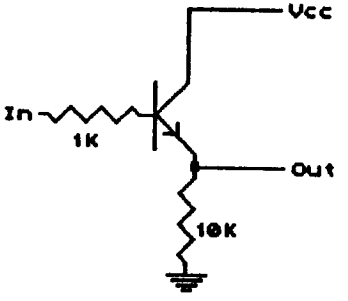


Transistor switches are small and compact. One configuration may take the place of an entire IC. These switches can also be combined for more complex uses and still take up less space than many ICs. In theory, the transistor is in one of two states, on or off. This is done by biasing it such that only two regions of the load line are used; cutoff and saturation. Following are the schematics for two simple NPN transistor switches. One is inverting and one is non-inverting.

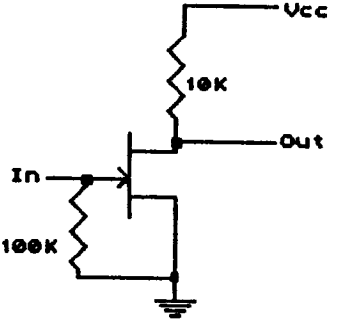
Inverting



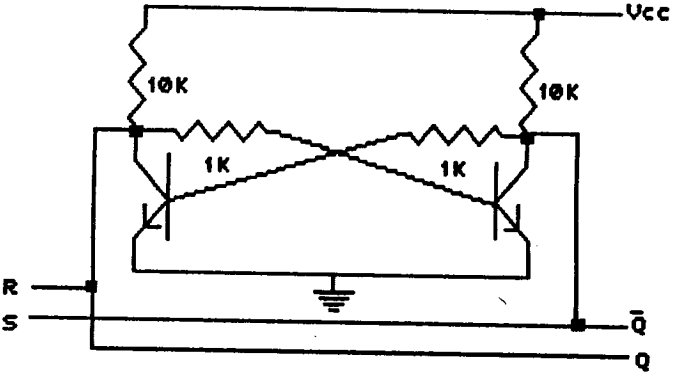
Non-Inverting



An FET switch may be preferable because the input resistance is far greater and will not load down previous stages as much. Below is the schematic of a simple FET switch.



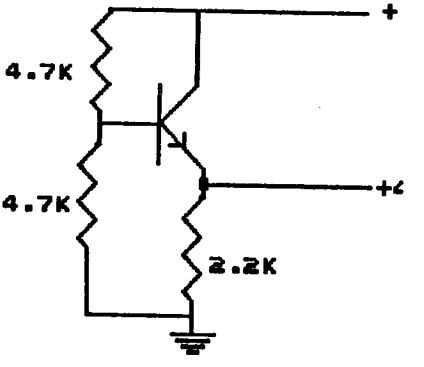
Using two NPN transistors, you can construct an RS flip-flop. An RS flip- flop can be used to "set and reset" the output of a device. One of these (in IC form) is used in the phone answering project. The schematic is on the following page.



Bipolar transistors can also be used for voltage regulators. To bias the base for the necessary voltage use the formula.

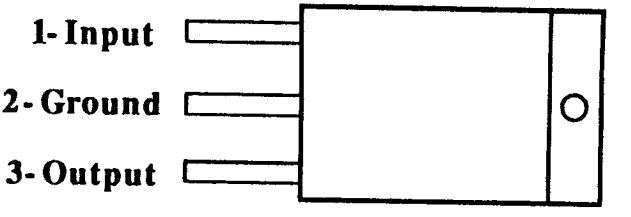
$$\frac{R1}{R1+R2} \text{ x Vcc-.7 volts=regulated voltage}$$

Below is a circuit using an NPN transistor which provides a regulated output voltage of 4.3 volts.



VOLTAGE REGULATOR ICs

Fixed voltage regulators are ideal for your own power supplies and for adjusting your power supply to make additional voltages available. Following is a diagram of a 7805 regulator, but the same diagram would apply to a 7812 or a 7815. The 7812 gives a regulated output of 12 volts. The 7815 gives a regulated output of 15 volts.

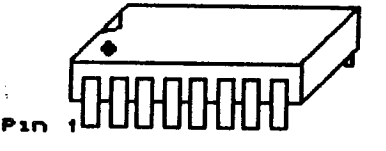


Earlier in this section, a power supply using a variable voltage regulator was shown. Refer to that diagram if you prefer a variable voltage power supply.

ICs

Integrated circuits are used in the bulk of the projects in this book. They are self-contained circuits by themselves. All operating requirements must be observed.

When the IC is right side up, pin 1 is at the lower left as shown below and is determined by the dot, indentation, or other marking as shown.



Sometimes, the date of manufacture is printed on the IC in addition to the part number. To complicate things, the part number may look slightly different than what you ordered. Before sorting your chips becomes a problem, get in the habit of keeping them in separately labeled locations so that correct part numbers can be identified.

When soldering an IC into a project, you should use a socket to protect it. The sockets with tin leads are fine but those with the gold leads solder easier. Exercise care when inserting them into the socket. Sometimes the pins of the IC on one side may have to have a slight pressure applied so that they fit in the socket better.

The three types of ICs used in this book are TTL, CMOS, and linear. TTL means transistor-transistor logic. Sometimes LS versions of TTL ICs are used. LS stands for low power schottky and has to do with the manufacturing process. CMOS stands for complementary metal oxide semiconductor. Linear ICs have an output that is proportional to the input. Some linear ICs can be used in a non-linear fashion.

The operating requirements for TTL ICs are:

1. Vcc must not exceed 5.25 volts.
2. Input signals must not fall below ground.
3. Unused inputs usually assume the high state, but if you intend it to be high, connect it to Vcc.
4. If an input should be in the low state, connect it to ground.
5. Connect unused inputs to Vcc to prevent unnecessarily high current consumption.
6. Avoid excessively long connecting wires.

One TTL output will drive as many as 10 TTL gates or as many as 20 LS gates. One LS output will drive up to 5 TTL gates or as many as 10 LS gates.

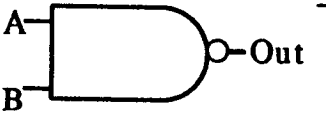
If your circuit is not operating properly, check the following: 1) all pins go somewhere, 2) all IC pins are in the socket, 3) all operating requirements have been met, 4) all connections have been made, and 5) Vcc is not more than 5.25 volts nor less than 4.75 volts.

CMOS ICs use less current than TTL and operate over a 3-15 volt range. The operating requirements for these chips are:

1. The input to any pin should not exceed Vcc except in the case of the 4049 and the 4050.
2. All unused inputs must go somewhere, either Vcc or ground.
3. Do not connect a signal when Vcc is off.
4. Avoid static on the IC at all costs; they are very sensitive and can be easily destroyed.
   a. Always store in a conductive container.
   b. Do not place on a non-conductive surface.
   c. Make sure that you are discharged of any static build-up before handling.
5. When interfaced with TTL, make sure Vcc is at least 5 volts.

You may want to design your own external logic either going into the computer or coming out. Following are the basic logic gates and their truth tables.

NAND Gate



A	B	Out
L	L	H
L	H	L
H	L	L
H	H	L

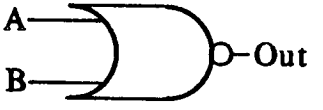
AND Gate

A	B	Out
L	L	L
L	H	L
H	L	L
H	H	H

OR Gate

A	B	Out
L	L	L
L	H	H
H	L	H
H	H	H

NOR Gate



A	B	Out
L	L	H
L	H	L
H	L	L
H	H	L

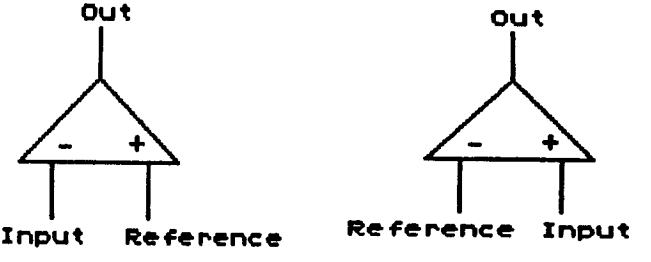
EXCLUSIVE OR Gate

A	B	Out
L	L	L
L	H	H
H	L	H
H	H	L

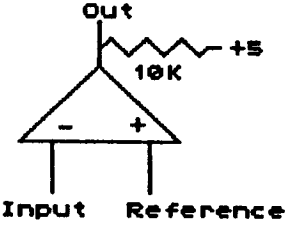
In the diagrams, the L means low and corresponds to the zeros in binary. The H means high and corresponds to the ones in binary.

Voltage comparators, specifically the LM339, are used many times for the projects. They compare a reference voltage to another voltage. Usually the reference voltage will be set up with fixed resistors, but it is possible for the reference voltage to float also. Below are the basic configurations. The non-inverting comparator's output is low when the input voltage (pins 5, 7, 9, or 11) falls below the

falls below the reference voltage (pins 4, 6, 8, or 10). For an inverting comparator, simply reverse the roles of reference and input voltage. The reference voltage will be on pins 4, 6, 8, or 10. The input voltage will be on pins 5, 7, 9, or 11. When the input voltage exceeds the reference voltage, the output goes low.



For TTL compatable output, connect a 10K resistor from the output to +5 volts as shown below.



TTL Output
Non-Inverting

2. Numbers!

A working knowledge of binary and hexadecimal numbers is very important to effectively understand the programming aspects of the projects in this book.

The reason that computers need to use binary numbers is that all of their internal circuitry uses a system of transistors that are either on or off. Their voltage levels in the "on" state approach 5 volts (TTL), and in the "off" state, approach ground, or 0 volts. The numeric system used to represent these on and off states is the binary system where an "on" is notated as a 1 and an "off" is notated as a 0. This is of special interest when building hardware projects. Hardware ports which are parallel read and write data in binary. The software values in the registers must reflect the data. If each bit of the interface is considered as a separate device-related bit, the programmer must consider this in the routine.

Binary numbers are in base 2 as opposed to our normal base 10 decimal numbers. When we count in base 10, we carry to the next digit after reaching 9. In base 2, we carry to the next digit after reaching 1. In base 10, it is helpful to think of the place values in terms of 10 raised to the power of the place. In base 2, this is also helpful

except that you will think of 2 raised to the power of the binary place. Below is a chart illustrating the binary values of each place and its representation as 2 raised to a power with respect to the place.

Binary Place	Exponential Form	Value
0	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128

The above chart is sufficient for 8 bit numbers, but for larger numbers than this, simply continue the chart.

To ascertain the value of a given binary number, sum up the values of the places which have a 1. You must realize that the computer counts places starting with the 0th place. We would ordinarily think of this as the 1st place but the computer sees it as the 0th place. The next place according to the computer is the 1st place, what we would normally call the second place. For an example of assessing the value of a binary number, let us consider 1101. You see that going from the right to the left you have the values of 1 + 4 + 8. 1 from the 0th place, no value from the 1st

place, 4 from the 2nd place, and 8 from the 3rd place. Summed up, this equals 13. The decimal equivalent of 1101 is 13. An example with a larger number is 11010011. The values you must add to get the total would be 1 + 2 + 16 + 64 + 128, this would give you 211.

There are a number of bitwise logical operations which you can perform on these binary numbers from assembly language or from the C compiler. All of these operations are performed on a number, usually in a variable or some specific memory location or register. The program- mer then specifies another number, the operand, with which the operation is carried out.

The logical AND operation takes the number in question and performs the operation with respect to the operand. The bits that remain are only those which both the number and the operand had in common. The other bits are lost, or as is commonly said, they are "masked out."

The result of the logical OR operation is that the bits that remain after the operation is performed are those in the number or the operand or both.

Another common operation is the exclusive OR. This follows the logic of the exclusive OR seen in the IC section of the electronics review.

Most assemblers or compilers need numbers in hexadecimal form. This is a base-16 system where the digits go as follows:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

The letter A represents a decimal value of 10, B represents 11, C represents 12, D represents 13, E represents 14, and F represents 15.

For an example of assessing a hexadecimal number, let us consider 8C. The value of the digit containing C is 12. The digit containing the 8 represents 8 x 16 or 128. Remember that since this is a base 16 numbering system, the second digit over from the right is the 16s place. The third digit over from the right is the 256s place, which comes from 16 to the second power. Continuing to the left, for each additional place from the right, you raise 16 to the power of one more.

3. Programming Tips

This chapter will describe various programming techniques that will be helpful to anybody using the eight bit Atari computers. Many of these techniques were used in the demonstration programs, so familiarity with them will enable better understanding of the projects and this book in general.

This chapter is organized in such a way that it provides a quick reference section as well as a comprehensive instructional chapter. Specific memory locations are discussed. These are helpful memory locations, and their use can be seen throughout the demonstration programs. Next in the chapter is a brief discussion of graphics modes. The graphics modes are used in several places within the programs but could be exploited more fully with some creativity. Using machine language subroutines is explained and made manageable. This will be extremely useful for you as you incorporate machine language subroutines from your BASIC programs in order to increase the speed of execution and the efficiency. Display lists, display list interrupts, and vertical blank interrupts are then explained in detail. These will become powerful tools once you have learned them and are able to use them in your programming. They are used throughout the demonstration programs.

I. Useful Memory Locations

Part of your computer's memory is dedicated to the operating system. In this part of memory, the system maintains itself and all of its various functions. You can gain access to these locations by using two simple statements from BASIC; poke and peek.

To read a value in a memory location use the peek command. The format of this command is X=PEEK(LOCATION). This will put the value of that memory location into the variable named X. The location number will range from 0 to 65535 which is all of the memory on a 64K machine. The value returned to the variable will be in the range of 0 to 255. You might then ask how the computer works with numbers larger than 255. The answer is that the computer combines two or more locations to form larger numbers. In most instances, the operating system stores them in two consecutive locations. Their order, oddly enough, is the least significant byte followed by the most significant byte. To evaluate these two bytes, multiply the second byte (most significant) of the two consecutive bytes by 256. You then add the value of the first (least significant) byte to obtain the correct value represented by the two bytes.

An example of reading two consecutive memory locations that form a larger number

is found when finding the location of the display list. The two memory locations that hold this value are 560 and 561. To evaluate the decimal value of these two locations, try this routine: X=PEEK(560)+PEEK(561)*256. This will assign the value, in a base ten decimal number, of the display list location to the variable X.

In order to put values into memory location, you must use the poke command. The format of this command is POKE LOCATION,VALUE. The location can be any number from 0 to 65535. The value that you put into that location can be any number from 0 to 255. Once again, for values past 255, two separate bytes must be used.

Before going on, try a simple experiment to see how these work. In immediate mode from your computer, type in PRINT PEEK(710). A value of 148 should be printed on the screen. This is the color of your screen that the computer begins with by default. To change your screen color, type POKE 710,0. This will turn your screen to black. Experiment with other values from 0 to 255 to see what colors you can produce on the screen. Remember that you PEEKED a value of 148 before you started. If you want to return to the original color, you will have to POKE 710,148.

Here is a list of the locations that we found most useful:

Location(s)	Usage
18,19,20	These three locations comprise an internal realtime clock. Location 20 increments every vertical blank (1/60 of a second). When location 20 reaches 255, it increments location 19 and recycles back to 0. When location 19 reaches 255, it increments location 18 and then recycles back to 0.
65	Turns disk and cassette I/O noise either on or off. Poke with a 0 for silence during these operations or any non-zero value to make the noise resume.
82	Controls the left margin. A value of zero allows text to be printed all the way on left part of the screen. Any other value will move the left margin in by the number of characters stored in location 82.

83	Controls the right margin. Initialized to a value of 39 when the system boots.
195	The error number that occured last. This is useful when you have an error trap set up because when it is implemented, your program can find out the exact error number.
203-206	Four free bytes unused by the operating system. These four locations were used frequently as temporary holding locations from the machine language subroutines.
546,547	Immediate vertical blank vector.
548,549	Deferred vertical blank vector.
564	Light pen horizontal value.
565	Light pen vertical value.
624-631	Paddle registers. There are eight of these because the

	original 800s had four joystick ports. The newer 800XLs have only two joystick ports so only the first four of these locations are used.
632-635	Stick registers. All four are used for the 800s while only the first two are necessary for the 800XLs.
644-647	Trigger registers. All four are used for the 800s while only the first two are necessary for the 800XLs.
710	Screen color for graphics 0 mode.
741,742	Pointer to the top of free memory. Used in the program called OVERMAKE to make space for the overlay in some cases.
752	Cursor inhibit flag. A zero turns the cursor on while a one turns it back off.
764	Internal hardware value for the last key pressed. A

value of 255 indicates that all keypresses have been processed. A value other than 255 indicates a keypress that requires processing.

1536-1791 Page six. This a set of 256 bytes that can be used for any machine language routines or data storage by the user. It is not used by the OS.

54016,54018 Used together, these two locations can be used to configure the joystick ports for either input or output. There will be more about that in the joystick port chapter.

For a complete description of all locations in the operating system, consult "Mapping the Atari" by Compute! Books.

II. Graphics Modes

Most of the demonstration programs use graphics 0, the mode that is best for text. Others use graphics modes that are better suited for visual displays. Below is a chart of the resolutions possible with the first nine graphics modes.

Mode	Columns	Rows	Colors available
0	40	24	11
20	24	52	20
12	53	40	24
44	80	48	25
80	48	46	160
96	47	160	96
48	360	192	1

III. Machine Language Routines from BASIC

Atari BASIC has a command that calls a machine language subroutine. The format of the command varies depending upon the format of the data that comprises the subroutine. In general, though, the syntax from BASIC is USR(ROUTINE). When this command is used, control of the program passes to a machine language subroutine.

We will deal with only two ways of developing a subroutine although many more exist. The first and easiest way is to use one of the available assemblers. These will convert the pneumonic codes into the actual machine codes that the 6502 microprocessor understands. The second and less convenient way to create these subroutines is to hand code your instructions using a chart. This takes a great deal more time and there is a much greater chance of including errors in your code. In any case, you must have your set of instructions in the form of actual machine language instructions that the processor can understand.

From the BASIC program, there are three ways to call the subroutine. They are carried out by using a specific memory address, by using a string variable which contains the code and then using that address of the string, and finally by having the the code contained inside a string within the USR command itself.

In order to call a subroutine from a specific memory location, you must first load the code into the appropriate memory locations. Page six (1536-1791) is a very convenient place since it is out of the way of the operating system and BASIC. You then use the command X=USR(LOCATION). The variable X is a dummy and no value is

assigned to it or taken from it. The value contained in LOCATION is the starting address of your routine.

In order to load a routine into memory, there are two very convenient options. You can rename the assembled file to AUTORUN.SYS and it will load in at the proper starting address. It is then ready for your use whenever you call it from BASIC using the USR command. Another good way to load in routines is to use data statements and simply POKE the data into the proper locations. The routines can then be called from your BASIC program using the USR command.

If you prefer, your machine language code can be put into a string variable. Each consecutive byte of the code is assigned to each consecutive byte of the string variable. The format of the command from BASIC is then X=USR(ADDR(A$)). X is a dummy variable and A$ is the string variable that contains the machine language code. Notice that the starting address of A$ must be passed in the command.

A slight variation of the previous method is to insert the code into the actual USR command itself. You do this by typing in the characters that correspond to the ATASCII values of the codes. A typical usage of this format might be X=USR(

ADDR("gq1^%@1448(*(+_.aks")). The characters look like total gibberish to us, but when translated to numbers the 6502 will understand the codes (provided that your original routine is correct).

With whatever form you decide to use the USR command, you have at your disposal, when using this command, another very powerful option. From the BASIC program, you can pass values to the subroutine. These values might be anything that is important to the routine or something that the routine will perform an operation on. To pass values, use the following format: X=USR(LOCATION,VALUE1,VALUE2,VALUE3,...). Theoretically, you can pass up to 255 values to your subroutine. It is important to know how this information is received by the subroutine in order to use it correctly.

When a machine language subroutine is called from BASIC, as it goes to that routine, it puts values on the stack as it goes. The first two bytes that are put onto the stack are the address of the current BASIC program command being executed. It does this so that it knows where to go back to when it encounters the RTS in the routine. The next thing that is put onto the stack are all of the arguments that are being passed. These were designated as VALUE1,

VALUE2, VALUE3,... in the above paragraph. The value of these arguments is put onto the stack with the most significant byte first, followed by the least significant byte. Even if the value passed is less than or equal to 255, two bytes are still passed. The last thing that is placed on the stack is a single byte which contains the number of arguments that are being passed from BASIC. Even if no arguments are passed, this byte is placed on the stack as a zero.

As stated before, up to 255 arguments can be passed. The problem will be that the stack, which is 256 bytes, will overflow and the address of the current BASIC command will be lost, not to mention any data overflow. When values are placed on the stack, the most recent byte goes on top. As more and more data is placed onto the stack, the previously pushed data goes lower and lower. To get any of the data from the stack, you must first pull off data that is on top. Make sure that when you write your machine language code that you remember this system.

Suppose that you call a machine language routine that expects no arguments to be passed from the BASIC program. If you do not pull the byte from the stack that was put there indicating the number of arguments sent from BASIC (a zero in this

case), the computer will mistakenly interpret this as one of the bytes that represents the address of the BASIC command which the program must return to. To avoid this problem, make sure that you pull this single byte from the stack before you do an RTS to get back to the BASIC program.

If you have passed one or more arguments to the subroutine, you must pull them off the stack in the order of the most significant byte first, followed by the least significant byte. This is because they were placed on the stack in the order of least significant byte first, followed by the most significant byte. Don't forget that the rule when pushing to and pulling from the stack is that the last byte on is the first byte off.

Included on the disk is a program called USRMAKER. This program will take a machine language routine from disk and convert it to a format that is usable by your BASIC program. To create the routines, use one of the available assemblers. The USRMAKER program will discard the first six bytes of the file as these contain information pertaining to the starting address. The program will ask you what format you want the data in. Any of the previously discussed formats can be chosen. If the data contains a 155 however, only the data statement option can be used. This is because the ATASCII

character of 155 is a carriage return and cannot be represented on the screen as a character.

IV. Display Lists

If you have ever wondered about where the screen display resided in memory, this section will shed a great deal of light on the subject. The entity that controls the location of screen memory and the graphic mode of each line is called the display list. It is a list which describes to the computer the type of screen display that is currently being used.

The ANTIC chip controls the video display that you see on the screen. This chip is very valuable because it takes care of a lot of the mundane details concerned with taking screen memory and turning it into a visual image on the screen. In this way, the 6502 microprocessor can do other important things like computations for your program. The display list is where the data which specifies the way that the screen data is to be displayed can be found.

The display list is a set of numbers that designates graphics modes for each line on the screen that is to be displayed. Each graphics mode has its own specific number that when placed in the display list will alert the ANTIC chip as to the specific mode.

These numbers do not correspond to the graphics modes that are used in BASIC.

Following is a list of the ANTIC modes.

ANTIC Mode	BASIC Mode	Scan Line Width
2	0	8
3	none	10
4	none	8
5	none	16
6	1	8
7	2	16
8	3	8
9	4	4
10	5	4
11	6	1
12	none	1
13	7	2
14	none	1
15	8	1

The above ANTIC modes are important to know when you are designing your own custom display list.

The first thing that you have to do when experimenting with display lists is to find the current list. Two locations in the operating system will contain the location in least significant/most significant format. These two locations are 560 and 561 respectively. To get the value, use the

formula X=PEEK(560)+PEEK(561)*256. This address will vary from graphic mode to graphic mode. The length of the display list will also vary from mode to mode; the higher the resolution, the longer the display list. The longer display list is necessary because there are more individual lines in the higher resolution screens.

Now that you have found the display list, you must know what is in it and what these things all mean. In BASIC, the first three bytes of the display list will be 112, 112, and 112. The purpose of these values is to place three blank lines at the top of the screen. If you look at your screen, you will notice the black border at the top of the screen. This prevents monitor (or TV) overscan.

The next byte found in the BASIC display list will be 66. There are actually two different pieces of information found in this number. First of all, bit 6 (index 0) of the number which has the value of 64 when set, tells ANTIC that the next two bytes of data in the list make up the address of screen memory. This is the start of the place where the actual screen memory data is kept. Another part of this number is 2 which indicates the graphics mode 0 or the ANTIC mode 2. Just remember that if bit 6 is set to one, the next two bytes will indicate the address of screen memory.

Now that we have explained the first six bytes of the BASIC display list for graphics zero, we see a succession of 23 twos. These 23 twos indicate 23 lines in graphics 0 or ANTIC 2 mode. If there are 24 lines in graphics 0 though, where is the 24th line in the display list? It is contained in the byte that had the value of 66. The value of 66 represented the graphics mode of 0 or ANTIC mode of 2. Also, the 6th bit was set indicating that the next two bytes would give the starting address of screen memory.

The next byte to explain in the BASIC display list has the value of 65. The 6th bit (with a value of 64) is set to 1 and indicates that a memory location pair will follow. The value of one that is also present tells ANTIC to wait for a vertical blank before returning control to the 6502. What memory location do the last two bytes represent? They represent the starting address of the display list.

You can now modify this BASIC display list to do many other things. Mainly, though, at this point you should try to experiment with mixed graphics modes. You can make some lines in one graphics mode while others might be in a different graphics mode.

Below is a brief explanation of how the internal chips work together to

implement the screen display. The ANTIC interrupts the 6502 to get the information from the display list and display memory. The 6502 then resumes its operation. The ANTIC checks the graphics mode for the display and then sends that information as well as the actual screen data to be displayed to the C/GTIA chip. The C/GTIA then converts the information that the ANTIC sent into signals that your TV or monitor can display. The C/GTIA is the actual interfacing chip between your computer and the video device that is attached. The entire process is done every 1/60th of a second which is so fast that our eyes cannot see it happening.

Try running the demonstration program DLDISPLAY. It will find the display list and put the values on the screen.

V. Overlays

The overlay is a useful tool because it provides a space on the screen for displaying information which stays in the same place regardless of how much the rest of the screen scrolls. It is ideal for displaying the status of registers, variables, or locations while you are programming and need to monitor the status of important things. As an example of how this can be used, two programs are included on the demonstration

disk. These programs are called OVRLAY1 and OVRLAY2. To run either of these programs, they must be renamed to AUTORUN.SYS, and you must reboot. The program OVERLAY1 displays four different pieces of information. In the upper left corner, it shows the status of joystick one in binary form. If the individual bit is connected to ground or if it is receiving a TTL level low, then it will be a zero on the display. If it is not connected to ground or it is receiving a TTL level high, it will be a one on the display. Directly below that is the decimal value of this binary number. In the upper right hand corner is the status of the trigger button of joystick port 1. Below that is something labeled "Enter." This actually keeps track of the status of paddle 0. If paddle 0 is connected to ground, it will say "ON." Otherwise, in the unaltered state, it will read "OFF." The reason for this is that in a project that comes later in the book, the paddle 0 is used for an enter key.

If you want to turn the overlay off, press the start button. This will give you more space on the screen to do your programming. To turn the overlay back on, press the option button. The overlay values are automatically updated using a vertical blank interrupt. These will be discussed later in this chapter.

The program OVRLAY2 returns more

information to the screen and is intended to be used with the data selector hardware. Either overlay will help you when you are building your projects, because you can see the status of the joystick port.

As described previously, the display list tells the ANTIC chip where to find the data for screen memory. This is not limited to a single pointer. You could point to several different places which might contain screen memory. BASIC will update the screen memory beginning at 40000 with the print statements and the scrolling that you are accustomed to seeing. If you place screen memory anywhere else, it will not be disturbed by print statements and the system's scrolling. To create a pointer, you must set bit 6 (value of 64) to 1 in the display list. This will tell ANTIC that the next two bytes point to screen memory.

There is a program included on the demostration disk called OVERMAKE. It will create an overlay according to how you answer questions when prompted by the program. When you run the program, it will ask you how many lines you would like for the overlay. It is creating an overlay in graphics 0, so be aware of the fact that these will be text lines. Next, you will be asked if you want a line separating the overlay from the rest of screen memory. If you answer yes, a thin black line will

separate the overlay from the rest of the screen. This line is done in ANTIC mode 13 or graphics mode 7.

After you have specified the size of the overlay, you will be asked where in RAM you would like this separate screen memory to reside. Page six (1536-1791) is a good choice, but if the overlay is too big, it will not fit. If you do not have enough room in page 6, or you want to use page 6 for other purposes, try using the area towards the top of free RAM. If you go too high and there is not enough room, the program will tell you that it is going to adjust it down to a suitable location. The memory locations in the operating system that keep track of the top of free RAM are then adjusted so that your BASIC program does not overwrite into your overlay area.

The overlay program will then ask you if you want to display a string in the overlay area. If you do, type in what you want to appear remembering that the overlay will display 40 characters per line. If you do not want to enter a string, just hit return.

The program will then create the BASIC lines necessary to enable the overlay. After it does this, it will delete itself (the overlay program). What you will be left with is the BASIC program that you will need to implement the overlay. Just type

RUN and watch it happen. This program can then be incorporated into your other BASIC programs if you want. You can do this by renumbering the lines (if necessary) and listing it to disk (LIST"D:PROGRAM"). You can then enter it into your BASIC program (ENTER"D:PROGRAM").

After you have implemented a custom overlay, you may want to run the program DLDISPLAY. This will display all of the pertinent information of your new display list.

You should be aware that the ATASCII values of each character are different from the internal screen memory values needed to display the same characters.

The following routine will adjust an ATASCII value to the internal screen memory value:

```
10 REM X  is the ATASCII value in
          question
20 REM Y  is the adjusted screen value
30 Z=0:IF X>128 THEN X=X-128:Z=128
40 IF X<32 THEN Y=X+64+Z:GOTO 70
50 IF X<96 THEN Y=X-32+Z:GOTO 70
60 Y=X+Z
70 REM  Now do the next character or
        stop
80 REM and insert into screen memory.
```

VI. Display List Interrupts

The display list can be exploited even further than we have already done by enabling horizontal blank interrupts. These interrupts will happen when the raster operation is undergoing a horizontal synchronization. By using this technique, operations can happen automatically whenever you specify in the display list that an interrupt is to occur.

As you remember, the display list is composed of a series of numbers that tell the ANTIC chip what graphic mode to display on each line. For each graphic mode, there are a certain number of bytes that make up the display list. By inserting the proper instructions in the list, a display list interrupt can be created.

When a picture is drawn on the screen, an electron beam starts in the upper left corner and scans the entire screen down to the bottom right a single scan line at a time. Between the time the pixel on the far right is fired and the beginning of the next scan line is begun, there is a bit of time during which other tasks can be performed. It is in this time that the DLI (display list interrupt) works when enabled. That particular DLI will occur every sixtieth of a second, every time that command is encountered in the display list.

The time that the computer has for a DLI is limited to about 35 machine cycles, so only short routines are possible.

The DLI is fairly simple although you must know machine language to write the code that is to be performed. The DLI routine is enabled when it finds that a byte in the display list has the seventh bit set to one. The ANTIC chip then holds the flag that was set by the seventh bit until the end of the graphic mode line. ANTIC then looks at location 54286, and if bit 7 of this register is set to one, the DLI is performed. If it is not set to one, the DLI is ignored. To perform the interrupt, the 6502 jumps to the address which is pointed to by locations 512 and 513. These two locations make up the pointer to the interrupt in least significant byte/most significant byte format. After the interrupt is finished, control returns to the main program.

Your interrupt routine must save all of the 6502 registers before it does anything. You do this with the following:

```
PHA *Push the accummulator
TXA *X register to the accumulator
PHA *Push the accummulator (X register
      value)
TYA *Y register to the accumulator
PHA *Push the accumulator (Y register
      value)
```

Now that all the registers are saved, you can do whatever it is that you want your interrupt routine to do. If you do not intend to use any of the registers, you do not have to save their contents to the stack. This will save valuable processor time. Finally, you must retrieve the register contents from the stack. Try the following:

```
PLA *Get the Y register value into the
      accumulator
TAY *Put it back in the Y register
PLA *Get the X register value into the
      accumulator
TAX *Put it back into the X register
PLA *Get the original contents of the
      accumulator
```

The interrupt must end with an RTI command. This will send it back from the interrupt to the main program. The DLI can do about anything you have time for. Usually, it is used to change screen colors on the fly.

The procedure to enable the DLI is as follows. First, put your routine in memory where it can be vectored upon enabling of the interrupt. Next, alter the locations 512 and 513 to point to your routine. These are in least significant/most significant format. Choose the display list byte to alter. The only restriction is that it cannot be

one of the bytes that points to screen memory locations. The byte you choose will determine where on the screen the raster will be when the interrupt is enabled. The byte you choose must then have bit 7 set to one. You can do this by adding 128 to the value of that byte. Finally, set bit 7 of location 54286 to one so that the processor will execute the interrupt.

VI. Vertical Blank Interrupts

Vertical blank interrupts are executed every sixtieth of a second. The difference between the display list interrupt and the vertical blank interrupt is the time during the video process that each one happens. The vertical blank interrupt (VBI) occurs between the time that the electron beam finishes an entire screen redraw and the start of another screen redraw. There is much more time during this period for these interrupts, so they can be much more complex and time consuming than the DLI.

There are two different types of VBIs that you can use. The first is an immediate VBI and happens before the OS update. During this period, you can implement routines that use up to 2,000 machine cycles. The second type of VBI is a deferred VBI. This happens after the system's own update has occurred. Your routine can be up to

20,000 machine cycles. As you can see, using these interrupts gives you a lot of time to work with. The deferred VBI begins while no screen drawing is occurring, but after about 2,000 machine cycles, the screen redraw begins. The deferred VBI can continue, but any graphics that are done past this point may look bad since the screen redraw is taking place.

The first thing to do when setting up a VBI is to decide which type you need to use. If 2,000 machine cycles is enough, the immediate VBI might be best. If you want to overwrite one of the systems functions, such as a BASIC command that you rename or revector, then the deferred VBI is better since this will happen after the system's own update has occurred. One consideration is that both types of interrupts steal processor time from the main program. If the speed of execution of the main program is very important, you might have to keep the interrupt routine short. If the main routine's execution time is not critical, the interrupt routine will not adversely affect it.

After deciding what type of interrupt to use, decide what it must do and where in memory you are going to put it. Once again, page six (1536-1791) is a good choice since it is out of the way of the OS and BASIC. You can either load the routine as an AUTORUN.SYS file or you can load it from

BASIC by poking it into memory.

If you are going to use an immediate VBI, set the two locations 546 and 547 to point to your routine. These two locations are in most significant/least significant order, but be careful because this is not the most frequently used order. If your routine is to be a deferred VBI, set the locations 548 and 549 to point to your routine. They also are in most significant/least significant order. The routine must terminate in one of the two following ways. For immediate VBIs, end with a JMP $E45F. For deferred VBIs, end with JMP $E462. You can increase the speed of your immediate VBI by jumping over the system's interrupt. To do this, end your immediate routine with JMP $E462 instead of JMP $E45F.

When you are setting the pointers of the VBI to your routine, there is a small chance that the interrupt will occur between the loading of the two pointer bytes. If that had happened, the vector would have sent the routine to some unknown destination, and who knows what would have happened. Fortunately, the Atari OS has a method to prevent this from happening. The routine located at $E45C will store the pointers to your routine safely. To use this routine, load the X register with the high byte (MSB) of the address of your routine, and load the Y register with the low byte (LSB) of the

address of your routine. Then, load the accumulator with a 6 for an immediate VBI or a 7 for a deferred VBI. At the end of this short routine, do a JSR $E45C. When you do this, the pointers will be put safely into the correct locations, and your routine will begin to operate within one sixtieth of a second.

Below is an example of the Assembly code and a BASIC routine that performs this function.

```
LDY #MSB  *Load the least significant
            byte of the VBI address
LDX #LSB  *Load the most significant
            byte of the VBI address
LDA #6 (or 7)  *6 for immediate, 7 for
                 deferred
JMP $E45C  *You will be returned here
             after the pointer routine is
             done
```

```
10 FOR X=1 TO 11
20 READ A
30 POKE 1536,A:REM Put this short
                routine wherever there
                is room, or even put it in
                a string!
40 NEXT X
50 X=USR(1536)
60 REM Now the VBI pointers have been
       installed
100 DATA 104,160,MSB,162,LSB, 169,
          MODE,32,92,228,96
```

Note that MSB means most significant byte and LSB means least significant byte. These are sometimes referred to as high byte and low byte respectively. Note that in the DATA statement, you will have to insert your own pointers, and for the mode, you will put either a 6 or a 7.

The Joystick Ports

Many people who build projects that are to be plugged into the joystick port find that they get hung up on correctly wiring the device to the nine pin plug. In order to avoid any confusion, this subject will be covered thoroughly. The physical description of the ports, the pin assignments, making connectors, and data I/O will be covered in this chapter.

The older Atari 800s have four joystick ports located in the front of the machine. They are labeled from one to four with the port number. The XLs and XEs have two joystick ports located on the right side when looking at the keyboard from the front of the computer. They are labeled with their port; number, either one or two.

When you program in BASIC, the ports are numbered beginning with zero. To find the value of joystick one, you use the command X=STICK(0). Notice that the port designation is zero for the port that is physically labeled one. Any references made to the joystick registers in publications also refer to the first port as zero, the second port as one, and so forth for the older 800s.

Figure 4.1 is a chart representing the operating system registers and the BASIC commands that correspond to the physically labeled ports.

Physically labeled
_____800s_____

port #:	1	2	3	4
BASIC:				
STICK(#)	0	1	2	3
STRIG(#)	0	1	2	3
PADDLE(#)	0,1	2,3	4,5	6,7

Note that there are two paddle inputs for each port.

HARWARE REGISTERS:

Stick	632	633	634	635
Trigger	644	645	646	647
Paddles	270,271	272,273	274,275	276,277

Figure 4.1

The hardware registers can be read by peeking that particular location.

Each port contains nine pins which all have specific functions. The nine pins of each port are configured in the same way. Illustration 4.2 is a view of the port as seen when looking straight at the computer. The chart following the illustration lists the functional description of each pin.

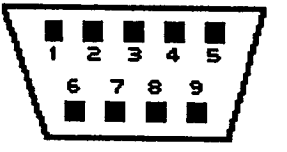


Figure 4.2

Pin Functions:

Pin#	Function
1	Data Bit 0
2	Data Bit 1 Data bits are
3	Data Bit 2 zero indexed
4	Data Bit 3
5	Paddle A
6	Trigger
7	5 volts (Safe to draw 300 ma.)
8	Ground
9	Paddle B

Pins 1-4 are held to five volts by a 10K internal resistor and are TTL compatable. If they are set for I/O, they will either be at a level approaching zero volts for a TTL low signal or at a level approaching five volts for a TTL high signal. These four pins can be set either for reading or for writing data. With the pins set for input only, which is the systems default upon reset or coldstart, they

return either a one or a zero to the operating system. If the pin is not connected to ground, or it is reading a TTL high signal, a one can be read. If it is connected to ground, or it is receiving a TTL low signal, a zero can be read.

The value returned by the BASIC command STICK(#) or by peeking the corresponding hardware register, will be a value from 0-15. With pins 1-4 all high, or at five volts, the value will be 15. With pins 1-4 all low, or at ground, the value will be 0. For values between 0 and 15, with bits 1-4 in differing states, you will need to check the individual bits of the value read. Pin 1 corresponds to bit 0, pin 2 corresponds to bit 1, pin 3 corresponds to bit 2, and pin 4 corresponds to bit 3.

The STRIG(#) command will return either a zero or a one depending upon the state of pin 6. If pin 6 is high, or unconnected, the value read will be a one. If pin 6 is low, or at ground, the value read will be a zero.

Pins 5 and 9 are connected to two separate internal analog to digital converters. These pins are meant to read voltage levels from zero to five volts and all values in between. The value can be read from BASIC using the command X=PADDLE(#) or by reading the hardware register.

Pin 7 of the port is a five volt source. This can be used for a variety of things. Some of the projects in this book are powered using this source. *You must be careful, however, to draw less than 300 milliamps.* This is a safe level of current to draw, but avoid greater drains than this level.

Pin 8 of the port is the ground connection. It will be used for every project you build. It must be connected to the ground of your project.

When you make the connecting wires for your joystick hardware projects, there are some things that you should keep in mind. The DB-9 connector that you need goes into the computer and is technically called a DB-9 socket. This is because it accepts the 9 protruding pins of the port. The port is technically called a DB-9 plug. When ordering the piece that goes into the computer, make sure that you order what is called a *DB-9 female socket*.

Make sure that when you solder to the DB-9 socket, you realize that you must keep the proper perspective. Make sure that the wires you solder will go to the correct pins of the port. Many people make these connections exactly opposite to what they should because they look at the socket from the wrong side when figuring out where to make the connections. Figure 4.3 will show

the correct way to look at these sockets when making the connections.

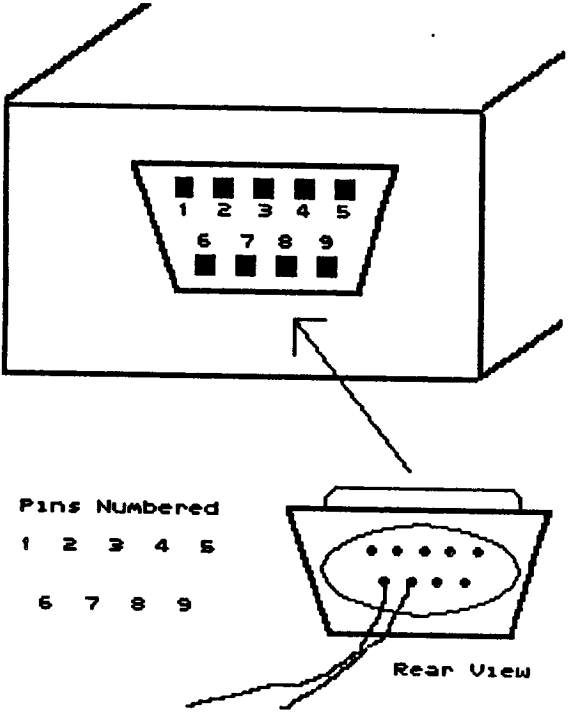


Figure 4.3

The plastic hoods that you get with the DB-9 connectors will prevent full insertion of the connector. These hoods are important because they protect the soldered connections and help keep them intact. The way to deal with this problem is simple. Snip the top and

bottom plastic piece that will extend beyond the flat metal portion of the DB-9 connector. You must then install the screws that hold the hood together with the exception of the two that are meant to protrude past the flat metal portion of the DB-9 connector. In order to hold the hood together, you must use superglue and hold it while it sets. This plug will then be perfectly adequate for your use.

There are several ways to read data from the joystick ports. To read the states of pins 1-4 from BASIC, you can use the command X=STICK(#). The variable X will contain a four bit binary number which corresponds to the states of the joystick pins 1-4. The decimal value that you will see when printing this on the screen will be from 0-15. From BASIC, you can also peek the register in the operating system. The value here will also be a four bit number ranging from 0-15.

To read the value of the trigger pin of each port from BASIC, you can use the command X=STRIG(#). The value returned will be either a one or a zero depending upon the state of pin 6 of the port. You can also peek the corresponding location in the operating system.

To read the value of the paddle pins of each port from BASIC, you can use the command X=PADDLE(#). The value returned will be one from 0-228 depending

upon the voltage that is on that pin of the port (voltage values range from 0-5 volts). There is a bug in the operating system that prevents the entire range of 0-255. Therefore, the entire voltage range cannot be measured. There will be more about this problem in the analog data section. You can also read the paddle values by peeking the corresponding location in the operating system.

Remember that Chart 4.1 has a list of the operating system locations that correspond to the different port pins.

One other option to read the ports is to peek location 54016. This location contains a combination of the pins 1-4 for the two ports. For the older 800s, location 54017 has the combination of the pins 1-4 for the upper two ports. The lower four bits of the value returned correspond to the four bits of the lower joystick port of the pair. The upper four bits of the value returned correspond to the four bits of the upper joystick of the pair.

To write data to the joystick ports, you must first configure it for output. The pins of each port that are used for output are pins 1-4. You can write to as many or as few of these pins as you desire depending upon how you configure the register. For the purposes of this discussion, you must consider bits 1-4 of both ports (or both pairs of ports for the

800s) as a single 8 bit number. The lower four bits are in the lower port while the upper four bits are in the upper port.

To configure the ports for output, begin by poking 54018 with 56. Use the BASIC command POKE 54018,56. You must then figure out which bits you want to configure for output. Whichever bits are intended for output must then be added up. For example, you may want to write to bits 1,2,4, and 6. Add up the bit values of 1+2+8+32 and get a sum of 43. Put these in a variable named BITS (any name is OK). By the way, these bits will correspond to port 1 (pins 1,2,4) and port 2 (pin 2). The next thing that must be done is to poke this value into location 54016, as POKE 54016,BITS. Finally, you poke 54018 with 60.

Now, to control these bits, simply put the value of the bits that you want on into the variable BITS and poke the location 54016 with this. Try typing POKE 54016,BITS. You will have no effect on the bits that are not configured for output.

A word of extreme caution at this time is in order. ***Never connect pin 7 (5 volts) directly to pin 8 (ground).*** This will draw more current than the system is capable of delivering and may burn out your PIA chip. Some of the projects connect resistances between these two pins which is acceptable. As long as the current drawn is less than 300

milliamps, you are safe. If you have any questions as to how to calculate the current, see the electronics reference section.

5. Switch Projects

For the first set of projects, we will start off slowly. These projects will help you to become familiar with the joystick ports and with making the DB-9 connectors. Remember that even simple projects can become useful tools with some creativity.

Parts List

1 - 9 Pin Female Socket (Hood Optional)
1 - Momentary, Normally Open, Push-button Switch
Hookup wire - 26 gauge solid wire is best for this project. Telephone wire is also very good.
Project box to mount switch(es) on is optional.

First, let's review the concept of the four data bits of each joystick port. These data bits correspond to pins 1-4 of each port. Together, two ports form an 8 bit number which can be read from location 54016. When these pins of the joystick port are in a low state, or approaching ground, they will return the value of zero. When they are in the high state, or approaching 5 volts, they will return the value of one. All of these projects and demonstration programs will use the joystick ports in the default setting; i.e. configured for input only.

Internally, pins 1-4 of each port are connected to 5 volts through a 10K resistor.

These resistors hold the hardware bits to high when they are unconnected externally. When the external pins are connected to ground, the 10K resistors have no effect upon the voltage levels read. This is because the current drawn through the resistor is insignificant and will not raise the pins' voltage levels above ground when it is connected externally to ground.

By connecting a switch between ground and pins 1-4, you can change the state of these pins and the values returned to the operating system. When the switch is closed and making a connection to ground, the value returned in that bit will be a zero. When the switch is open, making no connection to ground, the value returned in that bit will be a one.

For the first experiment, connect the switches between pins 1-4 and ground of joystick port one. Make sure that you wire them correctly. Once again, make sure that you do not make a direct connection between pin 7 (five volts) and pin 8 (ground).

The demonstration program provided for this experiment is called SWITCH. When you first run the program, you will notice that the state of the four data bits of joystick one are displayed. When you close any of the momentary switches to ground, the state displayed will be a zero. Otherwise, you will see a one as the state of the bit.

Notice that at the top of the screen it says in inverse video "Momentary Mode On [OPTION]." This means that the switch displays on the screen are responding only to the actual state of the bits. When they are connected, the state is displayed. If you want to see a toggle mode, you can press [START]. That line of video will then be in inverse indicating the mode. When you press one of the data bits, the state indicated on the screen will change. If it is on, it will toggle to the off state. If it is off, it will toggle to the on state. This is done with the software. Thus the port state does not actually toggle. To go back to the momentary mode, press [OPTION].

Another use of this simple switch experiment is to use it for a stopwatch. There is a demonstration program included called STOPWTCH. When you run the program, you will see that it is waiting for an event on bit one of the first joystick port. To start the timer, connect this data bit to ground and then release it. To stop the timer, simply press the same button again. This stops the timer and the elapsed time stays on the screen. To clear the time, you must connect data bit two of the port to ground. Once this is done, it will wait for another event on bit one. When that happens, it will restart the timer.

The stopwatch program uses the real time clock at locations 18, 19, and 20. It zeroes all of these locations when the timer starts and then continues reading them while updating the time on the screen. The combination of these locations are converted to seconds.

In the motion experiment chapter, the stopwatch program is modified somewhat to find the speed of a moving object. If you want to know about this modification, read the relevant section of the motion chapter.

Two switches can be used for a demonstration of logic gates. These logic gates are used throughout the computer's circuitry to make decisions at the lowest level. When programming, logical operators are used to make decisions. The same principles that are illustrated in these hardware gates can be applied to the logical operators that are used in software programming.

There are many different types of logic gates. There can be any number of inputs to these gates. For clarity and in order to stay within the scope of this book, we will only cover gates with two inputs. In this category there are five types that will be illustated. They are the AND, NAND, OR, NOR, and XOR (exclusive or) gates. A system of notating all possible inputs and the resulting output is used in electronics books. This

notating system is called a logic chart, or truth table. These are the same truth tables that are used for programming operations and for formal logical arguments. The gate with its truth table will be presented now so that you can understand their operation.

AND Gate: The AND gate's output is low until both inputs are high. When both inputs are high, the output of the AND gate is high.

AND Gate Truth Table

Inputs	Output	A	B
0	0	0	1
0	0	0	1
0	1	1	1

NAND Gate: This is the logical inverse of the AND gate. The NAND gate's output is high until both inputs are high. When both inputs are high, the output of the NAND gate is low.

NAND Gate Truth Table

Inputs	Output	A	B
0	0	1	1
0	1	0	1
1	1	1	0

OR Gate: The output of the OR gate is low if no input is high. When either input is high or both inputs are high, the output is high. The XOR (exclusive or) gate has a different result for both inputs high.

OR Truth Table

Inputs	Output	A	B
0	0	0	1
0	1	0	1
1	1	1	1

NOR Gate: The NOR gate is the logical inverse of the OR gate. The output of the NOR gate is high if both inputs are low. When either input is high or both inputs are high, the output is low.

NOR Truth Table

Inputs	Output	A	B
0	0	1	1
0	0	0	1
0	1	1	0

XOR Gate: The XOR (exclusive or) gate's output is low when either both inputs are low or both inputs are high. When either input is high but not both are high, the XOR gate's output is high.

XOR Truth Table

Inputs	Output	A	B
0	0	0	1
0	1	0	1
1	1	1	0

The demonstration program provided for illustration of these gates is called LOGIC. When you run it, you will see a truth table to the right of the screen and the electronic symbol for that gate to the left of the screen. When the program first runs, the AND gate is shown. You can change the gate by pressing the key which corresponds to the gate desired. These keys are listed at the top of the screen.

To use the program, bits one and two of joytick port one must be connected to a switch. When the switch is closed, it must connect the data bit to ground. This will alter the state of the bit and the display. You can use either momentary switches or toggle switches.

Notice that the inputs of the gate schematic reflect the state of the joystick bits. If the bit is connected to ground, the digit displayed will be a zero. If the bit is not connected to ground, the digit displayed will be a one. The output of the schematic reflects the correct output for that particular

gate. Each different gate will have different results for the various combinations of inputs.

When you change gate displays, the truth table changes to the correct output results. The schematic representation will also change according to the desired gate. Along the left side of the truth table, a small circle can be seen that indicates the current case. This will depend upon the state of bits one and two of the joystick port.

6. Event Detectors

Event Detectors

Event detectors allow the computer to know when an event has occurred. The uses of event detectors are endless. Their application is only limited to your imagination.

Switch Detectors

The most basic type of detector is made from a simple switch. An event is detected when the detector device connects any of the joystick data pins to ground. Switches can be placed between the pin and ground, and when they are opened or closed, the computer will be aware of their status.

The first type of switch you may consider is the door alarm variety. They come either normally open or normally closed. The type of switch you choose will depend on your application. An obvious use of these door alarm switches is in an alarm system. Using the joystick port, you could have up to 10 of these monitored at once. Eight would be between the joystick data bits (pins 1-4 of each port), and two would be between the trigger pins and ground.

In a later chapter, encoders will be discussed. An encoder can monitor up to 255 different alarm switches. You could also use one of the data selector projects and switch back and forth checking different sets of devices.

Another switch you might want to try is a micro switch. Unlike the alarm switches, these require very little movement to be opened or closed. Once again, these could be used as alarm switches, but their application would be different. They could be placed under objects that may be walked on or placed next to objects that move.

Visible Light Detectors

Most people have walked into a store and heard some sort of noise that alerts the store attendant that a customer has entered. Most of these detectors use a visible light beam and a sensor. Three versions of this type of system follow.

The light emitter for all three devices is virtually the same. The requirements are a narrow, focused light beam with a power source, either AC or DC. Your choice of a light source will largely be determined by whatever is most convenient.

The first detector is made from a single phototransistor. An OP802 works well, but any low current phototransistor with a relatively low saturation resistance is fine. If you use an OP802, you will notice, looking at it from the bottom, that there are three leads and a protruding tab on the metal case. The middle lead, the base of the phototransistor, will not be used and can be cut off. The lead closest to the tab on the case is the emitter

and will be the one connected to ground. The other lead, the one farthest from the tab, is the collector and will be connected to the pin of the joystick port.

Solder wires to the OP802's leads. These leads will go to the joystick port of the computer; one to the data bit and the other to ground. Install the OP802 on one side of the area to be monitored, and place the light source across from it. Aim the light beam at the phototransistor. When the light beam is aimed directly at the phototransistor, the computer will think that the joystick pin is connected to ground. This is because the phototransistor will be saturated and will exhibit a very low resistance. When the light is taken away, the computer will think that it is open to ground. This is because when the phototransistor is exposed to very little light, it has a high resistance.

Another effective visible light beam detector is a photovoltaic cell. It will not work in the same manner as the OP802. The photocell produces a DC voltage when light is applied. The joystick port data bits cannot discriminate between different voltage levels but only between the high and low states. For this reason, the positive side of the photocell must go to one of the analog to digital converters. This can be fed from pins five and nine of each port. The computer will then read a value from 0-228. One

advantage of this detector over the previous detector with the OP802 is its ability to adapt to different levels of ambient light. The device itself does not adjust, but in your program you can constantly average the values that are incoming. This can be used as a reference level from which the computer will base its decisions. Figure 6.1 shows the schematics of both versions.

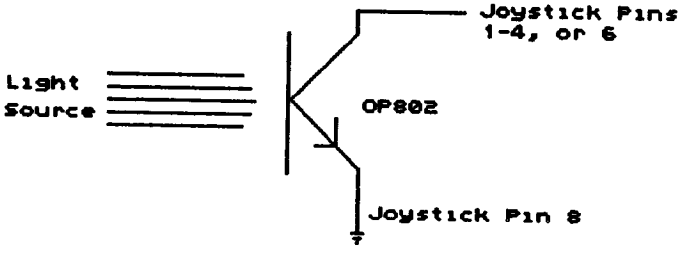


Figure 6.1a - Basic Phototransistor Detector

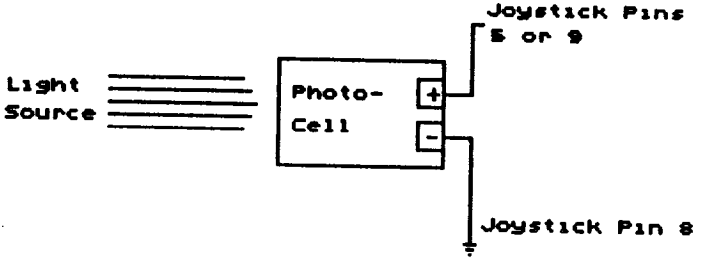


Figure 6.1b - Basic Photocell Detector

The ambient light in the surroundings may affect the operation of either of the previous circuits. One way to allow for this factor is to set up a device that uses two phototransistors. One phototransistor is the detector, and the other sets up a reference voltage based on the ambient light. The reference voltage floats and is dependent upon the ambient light. Both phototransistors are used as part of a voltage divider. The one which produces the reference voltage is only slightly different from the one that acts as the detector. This is because of an additional 100 ohms in the

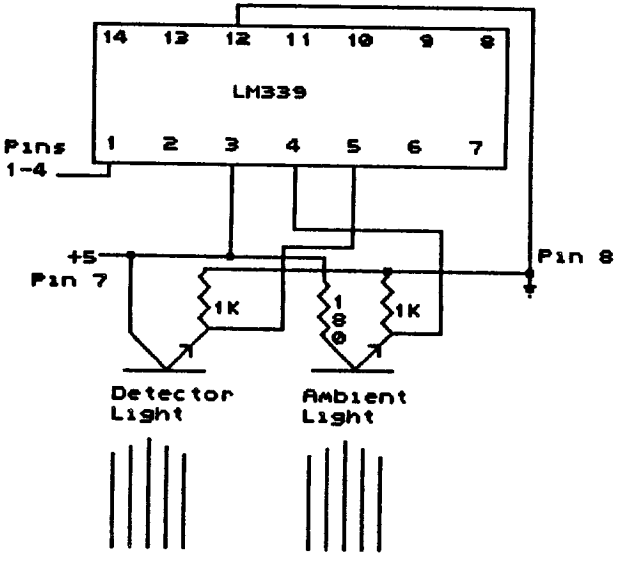


Figure 6.2 - Enhanced Detector

divider. Because of the difference in resistances and the resulting reference voltage, it only takes a slight variation in the detector light level for the comparator to go low. This, of course, alerts the computer to the event. Figure 6.2 is the schematic of this circuit.

Infrared Light Detector

Both of the visible light detectors that use phototransistors will work with an infrared light source. Infrared LEDs are the most convenient source to use and are readily available from parts suppliers. When selecting the type, try to choose one which has a high output. Radio Shack sells a version that emits about 1.5 milliwatts [catalog number 276-143 (TIL906-1)]. If you can get one which radiates about 6 milliwatts, your range will increase greatly. The angle of radiation is also important. This is the angle at which the infrared beam is emitted. Try for about 20 to 30 degrees. Avoid more than 50 degrees because the energy will not be as concentrated at this angle of radiation.

To drive the infrared LED, connect it to a 5 volt source with a 1/2 watt, 330 ohm, voltage-dropping resistor. Since it is impossible to see when the LED is on, you can use the circuit that is illustrated in the electronics chapter in the diode area.

Touch Switches

Two other circuits you might enjoy working with are different types of touch switches. The first one works when something such as a finger bridges two conductors. The second works by sensing a very small AC signal.

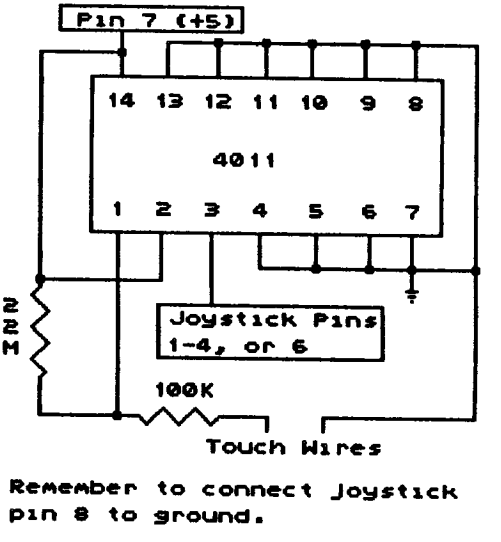


Figure 6.3 - Touch Switch

Figure 6.3 is the schematic of a touch switch that works when the touch wires are bridged by skin resistance. An etched plate may be more convenient to use than touch wires.

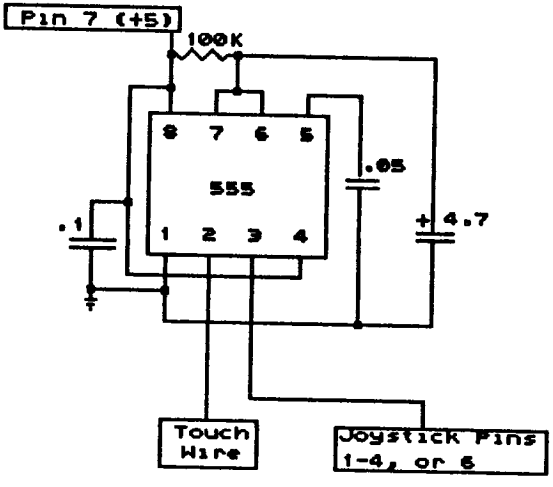


Figure 6.4 - Touch Switch

Figure 6.4 is the schematic of a touch switch that works when a single wire is touched. This switch works best indoors due to stray AC signals which might be present outdoors.

The demonstration program provided is called DETECT. It will show the status of all eight joystick data bits, both trigger bits, and all four paddle registers. The joystick data bits and the trigger bits will be used for the OP802 detectors and the switch type detectors. The paddle pins will be used for the photocell type detectors.

7. Motion Experiments

In every physics or physical science course, the subjects of speed, velocity, and acceleration are discussed. These topics are very important as they discuss the natural laws which keep the universe in motion. If you learned about these topics in the classroom or have read about them, this experiment will provide a hands-on experience of the subject that will reinforce the learned concepts.

First, let us review the idea of speed. We will be dealing with average speed in this book. Speed is different from velocity in that speed has no direction while velocity does. Average speed is defined as the total distance divided by the total time. If you go a distance of 5 miles in a time of 5 minutes, the speed is 1 mile per minute. In our more normal units, this converts to 60 miles per hour.

There is a set of standard units for measurements used in the scientific community. The name of this system is the "International System of Units." This is usually abbreviated as SI. The accepted unit of speed in the SI system is meters per second, or m/s. For this experiment we will be using centimeters, as space is a consideration. The accepted unit of speed when using centimeters is centimeters per second, or cm/s. To convert from m/s to cm/s, multiply the value by 100. To convert

from cm/s to m/s, divide the value by 100.

The word average was used with the word speed. The reason for this word usage is that almost everything has a variation in its speed, even though this might not appear to be the case. For this reason, it is more correct to talk about average speed. Suppose a car begins a journey in Miami, Florida and drives to Orlando, Florida. The distance is approximately 240 miles. If the car took 6 hours to complete the trip, we could calculate its speed at 40 miles per hour. That does not take into account stops or changes in speed along the way. Suppose that our driver was observed going 60 miles per hour near Ft. Lauderdale. How would this be possible? The first two hours of the trip, he went 60 miles per hour. The last four hours of the trip, he was going 30 miles per hour because his car was having problems with its cooling system. This example illustrates the reason for using the word average in a discussion of speed. Since we cannot measure the speed at every point (an infinite number), we use the average speed in our discussions.

Acceleration is a change in velocity or speed. Here, we will use it as a change in speed as we are not including direction in our discussion. Here again, we must use the word average in our discussion for the same reasons stated previously. To calculate the

average acceleration of an object, divide the change in velocity by the time it took to make the change. Suppose a car is going 40 m/s and the driver decides to change speed. He pushes the accelerator and arrives at a new speed of 60 m/s. As he pushes down the accelerator pedal, he notes the time as 10:44:10. At the very instant that the speed of 60 m/s is reached, he notes that the time is 10:44:50. The time it took to change to the new speed was 40 seconds. The change in speed was 20 m/s. Dividing 20 m/s by 40 seconds, you get a value of 1/2. You must also divide the units. Doing this leads us to the SI unit of acceleration, meters per second squared, or m/s2. The car in this example had an average acceleration of 1/2 m/s2.

The acceleration experiment in this chapter calculates the average speed and acceleration (if any) of an object traveling between a set of light sources and a set of detectors. With two detectors, an average speed could be determined in the following way. First, the distance between the two detectors must be measured. Next, the computer must wait for the first detector to be set off. At this point, it must begin counting the time. When the second detector is set off, the timer is stopped. The speed can be found by dividing the distance by the time.

There is a program on the demonstration disk called SPEED. It can be used to measure average speed between two points. You will be prompted for the distance that you will be timing. After that, the computer will wait for you to press a button which connects pin 1 of the joystick port to ground. The timer will keep track of the elapsed time. When the object reaches the ending point, press the stop button which connects pin 2 of the joystick port to ground. The time and speed will be displayed on the screen.

Two detectors are adequate for determining speed but not for determining acceleration. Remember that acceleration is the change in speed, so at least two separate speeds must be determined. To do this, more detectors have to be added. The experiment in this chapter uses four detectors. Between every two of the four detectors, a speed can be determined. With four detectors, there will be three separate speed values. Of course, with no acceleration, these three values will be the same. With three speeds, two changes in speed can be calculated. With these changes and a known time, two separate acceleration values can be calculated. The computer's timer keeps track of the time between detectors. The change in speed between speed one and speed two is divided by the time it took to make the change (the time interval between detectors

two and three). The change in speed between speed two and speed three is divided by the time it took to make that change (the time interval between detector three and four). These two acceleration values can then be averaged to get an average acceleration value. These calculations are based on a constant acceleration. A changing acceleration will not give accurate results with this system.

Parts List

4 - Flashlight Assemblies
4 - OP802 Phototransistors or comparable equivalents
1 - Female 9 pin Joystick Socket
1 - 6 volt Lantern Battery
4 - 6 volt Krypton Bulbs
Hookup wire, 3 feet baseboard moulding

The hardware is very simple to build and has two main and separate components. The first component is the set of light sources, and the second one is the set of detectors.

The light sources can be anything that you find convenient to use, but described here is the system we used for our test version. We first obtained four regular flashlights, the type that run off of two "D" batteries. We then removed the reflector assemblies. The bulbs were replaced with six volt krypton bulbs. Using six volt bulbs,

we were able to use lantern batteries which are easy to obtain and last for a long time. The krypton bulbs are 70% brighter than ordinary bulbs, according to the manufacturers. The brightness of these bulbs allowed a greater distance between the light sources and the detectors.

To mount the reflectors, we used a piece of plastic baseboard moulding. Four holes, each 1 1/2 inches in diameter, were cut. These holes must be spaced at a uniform distance when you put your project together. The reflectors were then glued into place aiming through the moulding. Wires were soldered in parallel to the bulbs. The wires then had an alligator clip attached to the end for easy connection to the battery. Figure 7.1 illustrates the light source assembly.

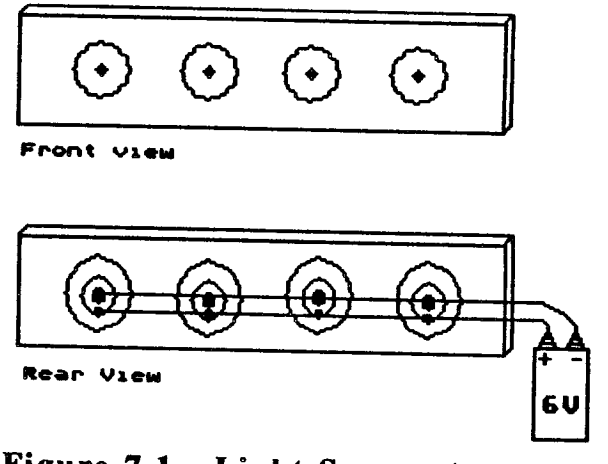


Figure 7.1 - Light Source Assembly

The second component was the set of four detectors. Each detector is simply a phototransistor. When the phototransistor is exposed to enough light, it conducts and acts like a closed switch. When the phototransistor is exposed to very little light, it does not conduct and acts like an open switch. The phototransistors can then be connected between the joystick pins and ground. When the phototransistor is exposed to light and is conducting, the joystick pin is connected to ground. The computer interprets this as such. When the phototransistor is not exposed to light and is not conducting, the joystick pin is not connected to ground and the computer can make the associated interpretation.

The emitter of each phototransistor should be connected to ground, pin 8 of the joystick port. The collectors should be connected to the data pins, pins 1-4 of joystick port one. Figure 7.2 is a schematic diagram of the circuit.

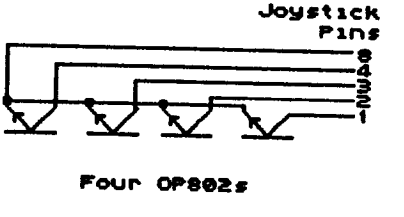


Figure 7.2- Detector Schematic

The physical configuration of the detector assembly is similar to that of the light source assembly. A piece of plastic baseboard moulding can once again be used. If you use OP802 phototransistors, 1/4 inch holes should be drilled. If you are using another type, make sure that they fit very snuggly in place when inserted. The distance between the detectors must be the same exact distance that you left between the centers of the light sources. They must also be the same distance from the top and bottom of the moulding as the centers of the light sources were. Figure 7.3 is a diagram of the detector assembly.

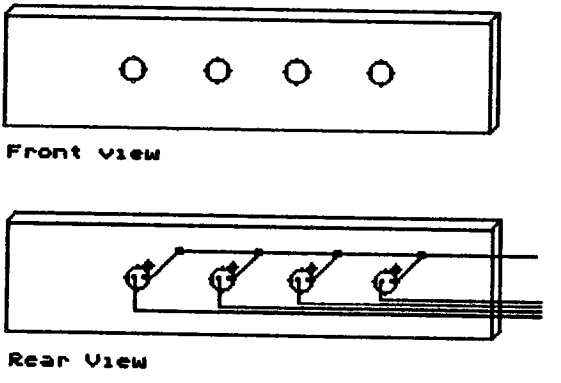


Figure 7.3 - Detector Assembly

When the light source assembly is placed across from the detectors, each light beam should shine directly on the corresponding detector. Figure 7.4 is a diagram of both assemblies across from each other.

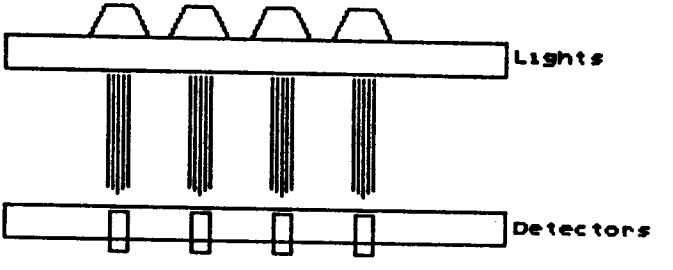


Figure 7.4 - Alignment
(View from Above)

The demonstration program provided which calculates both speed and acceleration is called SPDACCEL. After the title is drawn on the screen, an alignment checking routine is executed. It will display any phototransistors that are out of alignment or go to the main program if they are all in alignment.

You must type in the distance between each phototransistor in centimeters. After this is done, the computer waits for the first event detection. Once this happens, the real time clock is zeroed and the time of each phototransistor is recorded in sixtieths of a second. These times are recorded by placing

the values from the real time clock, locations 19 and 20, into 1536 and 1537, 1538 and 1539, 1540 and 1541, and 1542 and 1543. Each pair of times recorded can then be converted to seconds by dividing by sixty. The first location of each pair is taken from location 19 of the real time clock, and the second location of each pair is taken from location 20 of the real time clock. Every time that location 20 reaches 255, location 19 is incremented. The time in seconds for both locations can be calculated as follows: TIME=(PEEK(1536)*256+PEEK(1537))/60.

The phototransistors are checked in reverse order. The computer expects bit 4 first, followed by bits 3, 2, and 1, in that order. It does this with the machine language subroutine that is in the string variable B$. Once all four detections are complete, the computer calculates the speed and acceleration and displays it on the screen.

Another very good experiment that the computer can perform is one that involves a simple pendulum. A pendulum of this type is one that has an object at the end of a string. The string is suspended from something that allows freedom of movement. The mass of the string must be much less than the mass which is suspended from it so that the string's mass does not significantly affect the experiment. Figure 7.5 is an illustration of this type of pendulum.

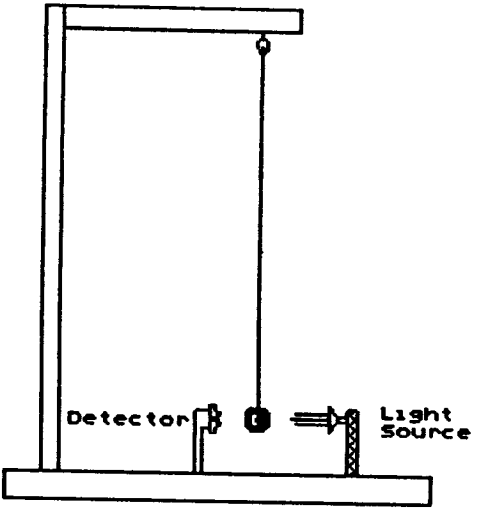


Figure 7.5 - Pendulum

When you pull the bob (the mass at the end of the string) back and let it go, it will swing back and forth for some time until it comes to a stop. We speak of the time it takes to swing from one extreme and back again a cycle. The frequency of the pendulum is the number of cycles per second. The period is the time it takes to complete a cycle.

Interestingly enough, the period and the frequency do not depend upon the distance you pull the bob back before you release it. It depends upon the force of gravity and the length of the string holding the bob. The theory and math is far too complex to go

into in this book, but with some research you could obtain the information if you were so inclined.

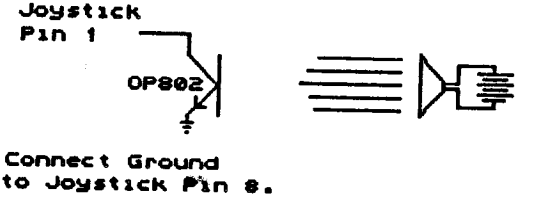


Figure 7.6 - Pendulum Schematic

In order to do this experiment with the computer, you must have one light source and one phototransistor. Connect the phototransistor between pin one and ground of joystick port one. Aim the light source at it and allow enough room between them for the pendulum bob to swing.

The demonstration program provided is called PENDULUM and will first ask for the length of the string. It will then display the frequency, period, and bob mass on the screen. It continuously updates this information.

8. Encoders

One of the problems with presenting data to the joystick ports is the limited number of joystick port pins. If you use each pin for one of the detector projects, you will soon run out of data bits. You could stretch the use of the ports by using the trigger pins for an additional two detectors, but this still limits the number of external devices that can be connected.

To overcome the difficulty of the limited number of data bits available, the data that is to be input can be encoded. By doing this, the information is encoded from raw external bits. Each corresponds to a certain device number. The encoder then transforms the information to a binary number which can be interpreted by the computer as the port is read.

There are many different types of encoders which can encode different numbers of input lines. The first encoder that will be discussed is a simple diode encoder. After that, two different IC encoders will be described. A NAND gate encoder is included because there are some applications where this type might be desirable.

Figure 8.1 is the schematic of a diode encoder. It will encode up to ten separate lines into four bits of a binary number. The four bits of the binary number are then connected to the four data bit pins of the joystick port. Pin eight of the port provides

ground for the circuit. When one of the ten data lines is connected to ground, the computer will see the resulting binary number at the joystick port. Make sure that the cathode of every diode goes to ground and that the anode of every diode goes to a data pin.

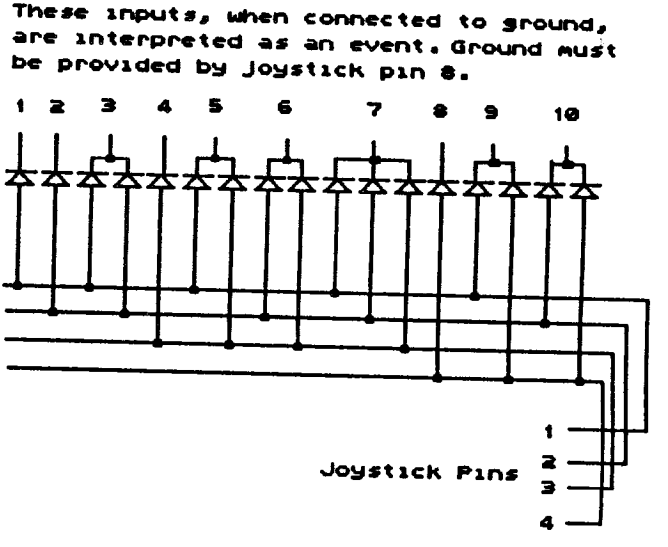


Figure 8.1 - Diode Encoder

One problem with the diode encoder is that it is not prioritized. If more than one external line is connected to ground, the binary value will be incorrect as the bits of two separate numbers will be present. The are some IC encoders that are prioritized and will prevent this from happening. The first

external line to be connected to ground will show up as the binary data, even if another external line is connected to ground while that external line is still connected.

Figure 8.2 illustrates the schematic of a 10 to 4 line encoder. A 74LS147 IC was used. Ten 10K resistors were used to hold the inputs high until they were connected to ground.

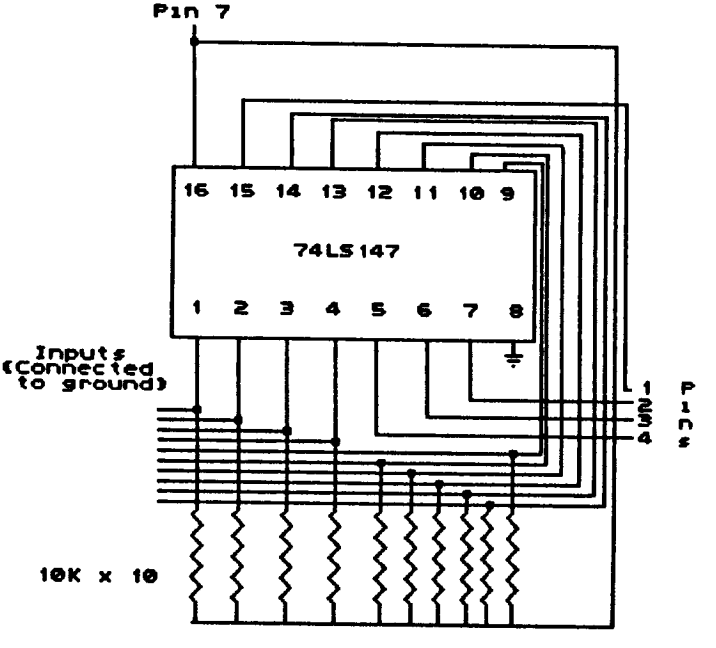


Figure 8.2 - 74LS147 Encoder

Often the 10 to 4 line encoder will not be sufficient for your needs. Another problem with the above IC is that there is no signal that tells you when a key has been pressed. This is a problem when external data line zero is connected to ground. The value of zero is returned to the computer, but the computer does not see that a key has been pressed.

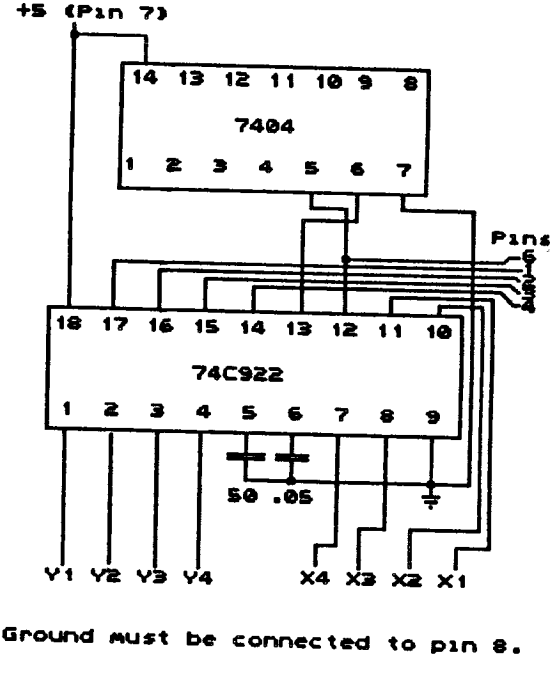


Figure 8.4 - 74C922 Encoder

To fix this problem, a different IC can be used. This IC is a 74C922 16 to 4 line encoder. It provides an additional number of external data lines over the 74LS147. It also has a pin which indicates when a key has been pressed. Figure 8.3 is the schematic of this circuit.

A very practical application for these encoders is in a keypad circuit. A keypad can be purchased from any of the parts suppliers. You could also use a set of single pole, momentary, normally open, push button switches. Figure 8.4 on the following page is a diagram of a keypad which is encoded by a 74C922.

The demonstration software provided is called ENCODE. When you run it, you will see the data at joystick port one. Press start if your circuit uses the 74C922 which signals when an external line has been connected to ground. If you have pressed [START] and wish to return to the mode used for the other encoders, press [OPTION].

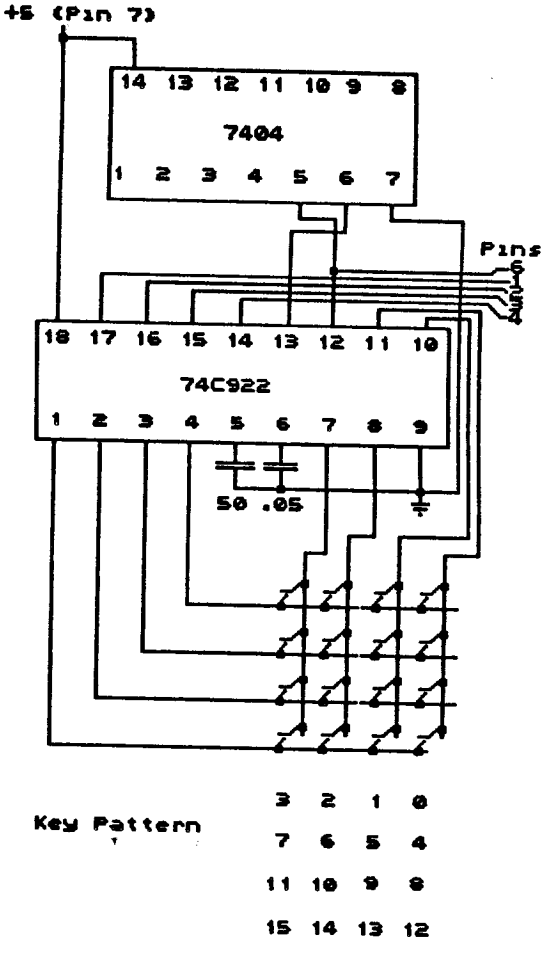


Figure 8.4 - Encoded Keypad

9. Light Pen

This chapter describes the construction of a light pen and related programming techniques. The lightpen, when held up to the screen, senses its relative screen position. The computer can use this information to perform tasks such as menu selection or drawing. The demonstration software which is included will not return values accurate enough for drawing, but with the right modifications, it will approach the necessary accuracy.

Before discussing the light pen itself, the operation of the monitor must be understood. The monitor is often referred to as a cathode ray tube, or CRT. The latest term which is now being used is VDT, or video display terminal. In general terms, it is a large vacuum tube with an electron emitter at the small end. This electron emitter is called the cathode. It is frequently called an electron gun. It shoots electrons at the large part of the tube, or the screen, which the computer user sees. This electron beam is what produces images on the CRT screen.

The beam draws the screen in small units called pixels. These must be directed so that the correct image is visible on the screen at the correct time. Direction is done in a very systematic way. First, the beam, or raster, is directed to the top left corner of the screen. The raster proceeds horizontally across the screen illuminating what is called a scan line.

The brightness and color of each pixel during this process is determined by the value in screen memory.

Once the raster has drawn a scan line, it blanks for a short time while synchronizing with the horizontal sync clock. It then repeats the procedure for the next scan line down. After this procedure has been repeated enough times, the entire screen has been drawn. This redrawing takes place every one- sixtieth of a second in NTSC and every one-fiftieth of a second for PAL systems. The screen redraw does not take this long however. There is a great deal of time during vertical blanks where the electron beam proceeds back to the starting position and waits to begin again. The actual screen update takes only 17,000 microseconds.

The light pen design in this book operates using a voltage comparator. The phototransistor inside of the pen is part of a voltage divider as pictured in Figure 9.1.

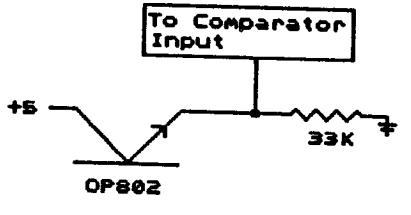


Figure 9.1 - Phototransistor in Voltage Divider

Voltage is divided proportionally across the two resistors according to Ohm's Law. The phototransistor's resistance changes according to the amount of light it is exposed to. The greater the light on the phototransistor, the less the resistance.

In the circuit branch of Figure 9.1, as the phototransistor's resistance decreases (with more light), the voltage out of the divider is raised. Conversely, when the phototransistor's resistance increases (with less light), the voltage out of the divider is lowered. The output of this voltage divider is then fed to an LM339 voltage comparator. The comparator checks to see if this voltage is above the reference voltage, and if so, the comparator's output goes low. The firing of the pixel directly below the pen within the phototransistor is what causes the sequence of events that makes the comparator's output low.

The reference voltage can be adjusted for different sensitivities. Increasing the 1.2K resistor increases the reference voltage and decreases sensitivity. The reason that the sensitivity is decreased is that it takes a greater amount of light on the phototransistor to make the voltage high enough to exceed this reference voltage. Decreasing the 1.2K resistor will decrease the reference voltage. The amount of light necessary for the voltage divider to trigger

the comparator will then be less. Figure 9.2 is the schematic for the comparator portion of the circuit.

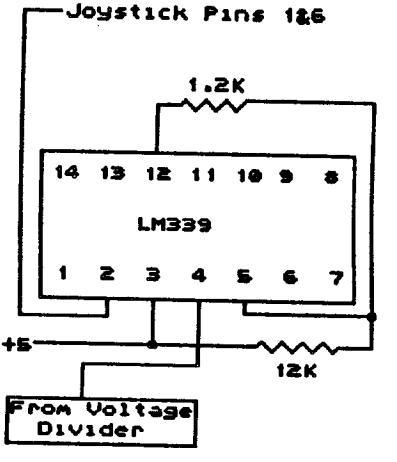


Figure 9.2 - Comparator

The output of the comparator must now be presented to the computer. This is done through the joystick port in the demo program. Since the pixel fires for such a short time, a fast sampling rate is needed. Thus, a machine language subroutine is essential. The operating system has a vertical counter and a horizontal counter for the purpose of a light pen. The schematic of the entire circuit is seen in Figure 9.3.

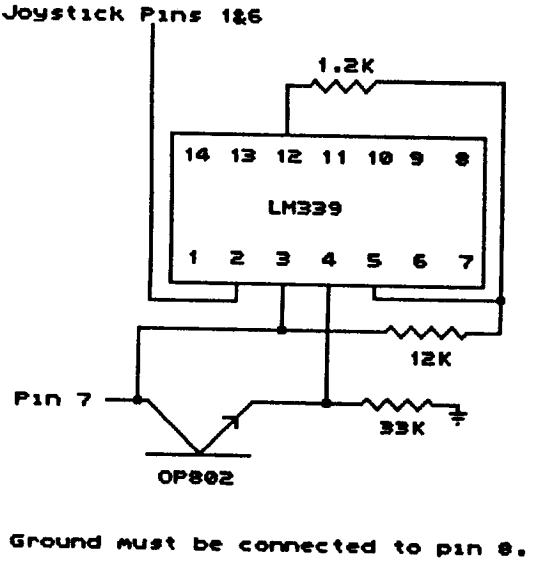


Figure 9.3 - Entire Lightpen Circuit

The software, called LGHTMUSC, begins by setting up a screen display of four vertical bars. Each vertical bar corresponds to one of the sound channels. Above the bars are on/off switches.

When the light pen is placed on the white square that turns the channel on, an indicator will show the change in the status of the sound channel. At first, no note will be sounding. Once the channel is on, move the light pen down to the vertical bar. You will see a circle follow the position of the light pen on the vertical bar. You will also

hear the pitch go higher and lower according to the relative height of the light pen.

All four channels can be turned on at the same time. If you choose, you can temporarily turn any of the channels off. Before you hit the break key to stop the program, you must have the pen up to the screen. Otherwise, it will stay inside of the subroutine waiting for a pixel to fire.

10. Device Control

When writing data to the joystick ports, the bits assume either a high or a low state. These states can be used as on/off switches to control external devices. In this chapter, various types of circuits which turn devices on and off are discussed.

If you need to review the procedure to set the ports for output, consult the joystick chapter. A complete description of how to configure the data bits for output can be seen there.

LEDs

The turning on and off of an LED has already been mentioned in this text. This procedure is easy to do and useful when writing data to the ports since you can see the state of the bits with the LEDs. TTL ICs cannot provide enough current to drive the LEDs, but they will illuminate very dimly if you rely on the current from an IC's output. In order to get as much brightness as possible, they are most effective when connected with the cathode to the IC's output and with the anode to a positive voltage through a voltage dropping resistor. The effect is that when the IC's output goes low, the LEDs cathode is connected to ground. The IC output can sink enough current, even though it cannot source enough. The 10-30 milliamps, depending on

the LED, will have no problem going to ground through the IC output. The anode should not be connected directly to +5 volts as this will burn it out. Use Ohms' Law for a specific calculation of the correct resistor. A 1K resistor will almost always work well.

To power the interfacing logic, you can use an external power supply, but it will probably be easier to use the 5 volts from the joystick port. If you use an external power supply, make sure that everything has a common ground.

Now, you can simply write data to the port according to the procedure previously outlined. The LED will light up when the output of the IC goes low. To correctly reflect the data, you should use an inverting buffer. This will make the IC's output go low when the data is high. The LED will then light up when the data in that particular bit is high.

Figure 10.1 (on the following page) is a diagram of 8 LEDs run from the parallel port using two 7404 inverters. The 5 volt power comes from pin 7 of the joystick port while the ground from the joystick port is pin 8. Each 7404 has six gates, so two of the ICs are necessary.

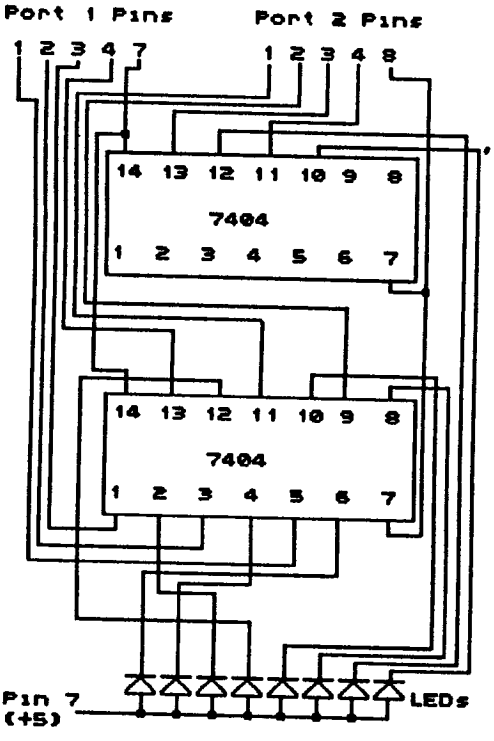


Figure 10.1 - LED Control

If you were so inclined, you could devise an intricate LED display. Using the joystick port allows you to have eight different, independent channels. Later on in this book, this very technique is used for controlling a Christmas light display. This LED display could also be synchronized with music from the sound chip.

Transistor Drivers

If you need to power a DC device which requires a higher voltage and current than a TTL IC can provide, you may consider driving it with a transistor. A transistor that will provide high current is a TIP102. It is a power darlington in a TO-220 case, and according to manufacturer's specs, should be able to provide up to 8 amps. If high current is anticipated, use a heat sink to protect against heat damage. An external power supply is required. It can be from 5 to 15 volts depending on the device to be powered. Figure 10.2 is the schematic for this device.

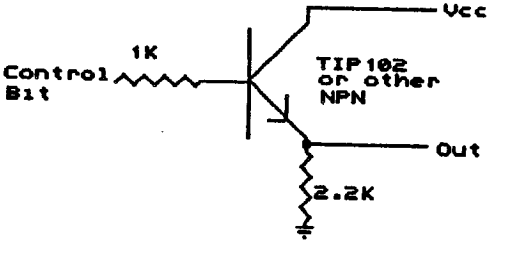


Figure 10.2 - Transistor Driver

You can use almost any NPN transistor in place of the TIP102, but check the data sheet to see how much current they can handle. A good medium- current alternative is the TIP31A. It should be able to handle 1

amp of current. It comes in a TO-220 package so that a heat sink can be attached.

You can use the transistor drivers to control stepper motors. Most stepper motors will have four stator coils. These must be pulsed in order for the motor to move. The combination of the stators that are pulsed determines the direction and magnitude of the motion.

Most stepper motors will have different specifications, so you will have to check the specific motor's specifications that you plan to use.

Relays

A 5 volt reed relay is very flexible because it will handle any voltage from 0-120 volts. At 120 volts, it is capable of carrying 1 amp of current. Using the 5 volt reed relay, many 120 volt devices can be safely powered. You can calculate the wattage of a light bulb for an application of 120 volts that draws 1 amp. Using the formula P = E x I or P = 120 volts x 1 amp, you can see that even the small 5 volt reed relay will handle up to a 120 watt light bulb.

With interfacing logic, the relay is best controlled in the same manner in which the LEDs were driven. When consulting the spec sheet of the relay, you will notice that at one end there are three leads. The two leads across from each other are for the primary

coil. Each of these leads should be connected to the output of the interfacing IC. The third lead should be connected to 5 volts without a dropping resistor. When the output of the IC goes low, one side of the relay's primary goes to ground, and since the other lead is at +5 volts, the circuit is completed. The relay secondary is now open and will conduct. Remember that the output of the IC gate must be low to enable the relay. Either use an inverter or remember that a low output turns it on when you write your software. Figure 10.3 is a diagram of reed relays controlled by an inverting buffer. It would be wise to use an external power source, but it is all right to use the computer's internal power supply for up to 4 or 5 reed relays.

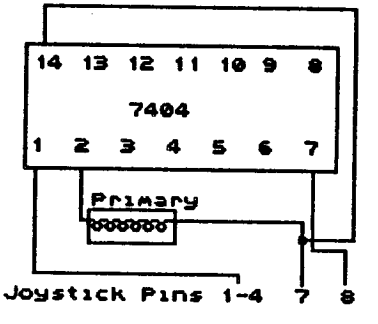


Figure 10.3 - Relay

If the reed relay does not have the current capabilites that you need, you can use a 5 volt miniature PC relay. Radio Shack markets a 5 volt mini relay that draws 72 milliamps for the primary coil. The secondaries can then handle 3 amps at 120 volts AC. The above circuit which ran the reed relays will also work for these relays.

You would need an external power source though, because more than three or four of these relays would overburden the internal power supply's capabilities.

You may find that even 3 amps at 120 volts is not enough. Relays that can handle higher amperage usually need 12 volts and more current than the logic gates can handle. For these relays, you will need to use a transistor driver. The drivers discussed earlier which used the TIP102 or the TIP31A would both be more than adequate. These drivers would then provide the power for the high current relays. Of course, this setup would require an external 12 volt power supply. One other alternative would be for the reed relay to control the higher amperage relay.

Opto Triacs

Opto Triacs, and many other types of opto isolators, can be used to control external devices. An opto triac will drive a low current, 120 volt AC device. Figure 10.4 is a

schematic of such a configuration.

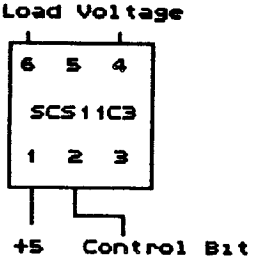


Figure 10.4 - Optotriac

An opto triac could also control one of the relays in the previous section. Figure 10.5 is the schematic of a circuit in which an opto triac controls a high current relay.

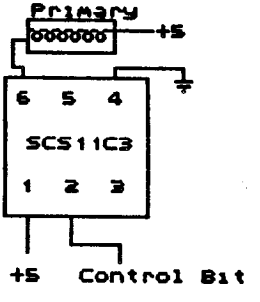


Figure 10.5 - Triac & Relay

Solid State IC Drivers

There are several ICs that will drive higher current, higher voltage devices. The chips illustrated require no external components.

Figure 10.6 is the schematic using a dual peripheral driver, a 75446, to drive a single device.

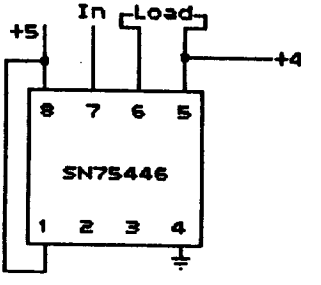


Figure 10.6 - Driver

The 75446 can also drive two separate devices as in the schematic of Figure 10.7.

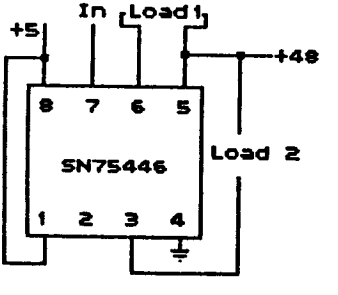


Figure 10.7 - Dual Driver

The DH0017 is a high voltage, high current driver which is another alternative to the above driver and, at 50 volts, can

handle a bit more current. The schematic for using this IC is is seen in Figure 10.8.

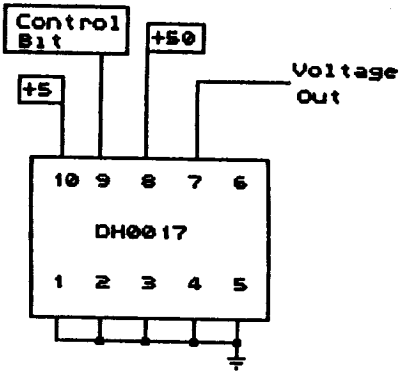


Figure 10.8 - DH0017 Driver

An application of the previous control circuits is to have the computer control various household devices at certain times. While developing your hardware, you can use one of the programs that write data to a port. After the hardware is developed, you will want to write software specifically for that purpose.

11. Decoders

One disadvantage of controlling external devices from the joystick ports is that each binary bit is a single control line going out. Decoders will take 4 bit binary data from the computer and affect 16 decoded bits one at a time. This will expand your control of external devices. One problem with using the circuit illustrated is that the state of only one control line at a time can be controlled. There is a solution to this problem which will be dealt with later in this chapter.

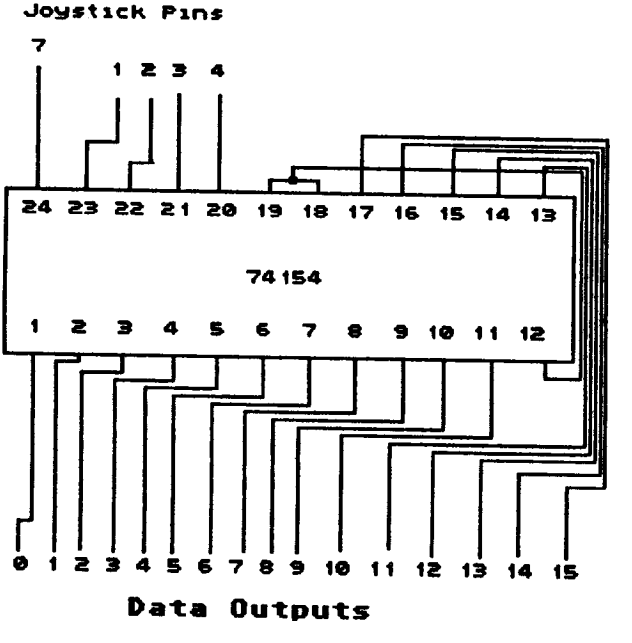


Figure 11.1 - Decoder

The first decoder configuration uses a 74LS154, a single 4 to 16 line decoder. This chip can also be used as a demultiplexer, so certain IC pins must be considered for use as a decoder. Pins 1-17 contain the outgoing data. Pins 20- 23 contain the incoming binary data. Pin 24 is +5 volts and pin 12 is ground. Pins 18 and 19 must be held low for the decoder to work. These pins are used differently if this IC is used as a demultiplexer. Figure 11.1 is the schematic for this circuit.

If you want to control the 16 outputs of the 74LS154 and keep them either on or off, independent of the other bits, you will need additional logic. A good solution is to use T flip- flops where the output of the flip-flop is toggled back and forth between states. The toggle is enabled in this configuration when the data out of the decoder goes low. The first low will set the output of the T flip-flop low. After a high and then a low are given as data, the flip-flop will toggle to high. The IC used in the circuit is a quadruple JK flip-flop set up in a T flip-flop format. Its part number is 74LS276. Figure 11.2 is the schematic for this circuit.

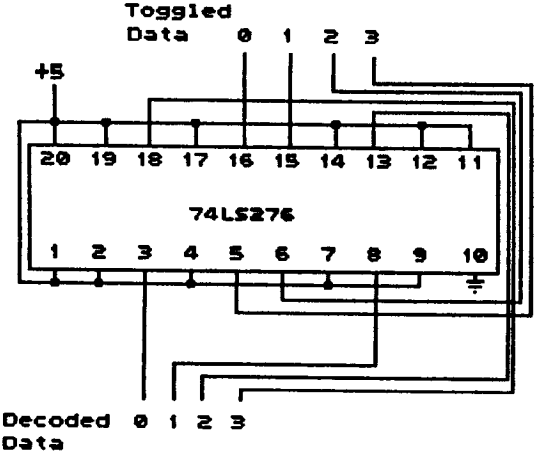


Figure 11.2 - T Flip-flop

Before you use these decoders in more complex circuits, you may want to build them on a solderless breadboard and use LEDs to see the state of the outputs. This is highly recommended as debugging a smaller circuit component is easier than debugging a larger circuit. A program for use with the 4 to 16 decoder, with or without the T flip-flop, is called DECODE. It runs from joystick port one. Of course, you could use both ports for a total of eight bits of decoded logic.

12. Serial Data

Computers communicate using either serial data transfer or parallel data transfer. Parallel communication is faster than serial communication because all of the bits can be sent simultaneously. Serial data is sent one bit at a time, so this would be obviously slower. Serial transfer is used though, for several practical reasons. The major reason serial data is advantageous is that the transfer medium requires far fewer connections. For the type of data transfer illustrated in this chapter, only two signals must be presented to the serial data decoder. These two lines are for the clock and for the data.

The project in this chapter demonstrates synchronous serial data. Some words of explanation are in order for this topic. When data is sent over the data line, a method for synchronizing the decoding of each bit must be implemented. The decoder needs to know when to interpret the data signal as the next bit and when to interpret the data signal as still being the same bit. Synchronous serial data is different from asynchronous serial data in one major way. Asynchronous decoders sample the data sixteen times for every bit and a counter keeps track of when a data bit begins and ends. Both the transmitter's and the receiver's synchronizing clocks must be very close to the same speeds. This means that the rate of

data transfer is slower because the asynchronous decoder samples sixteen times every data bit and also has to read start and stop bits.

Synchronous serial data decoders use a common clock (common to transmitter and receiver) to keep track of the data bits. This makes for a faster rate of transfer because there are no start and stop bits to contend with. The synchronous circuit is simpler in that the decoder does not require a separate clock.

The particular shift register used in the circuit to decode the serial data coming from the computer is a 74LS164. Attached to the 8 parallel data output pins of the IC are eight LEDs. The LEDs will show the data reflected in each of the eight data bits. Since the LEDs are attached with the cathode to the gate of the IC, the LED will light up when the output of the IC's gate is low. Because of this fact, the software will invert the data before sending it out. Data entered at the keyboard will be inverted, then sent out over the port. The lit LED can then be interpreted as a one.

Eight data bits must be sent to the shift register. The least significant bit will be sent first. The most significant bit will be sent last. As the clock goes from a low state to a high state, the shift register looks for a data bit. It then gathers whatever data is present

on the data line and places it in its first holding latch. As it accepts the data into this location, all of the other data bits in the shift register are shifted over one place. Figure 12.1 illustrates this process.

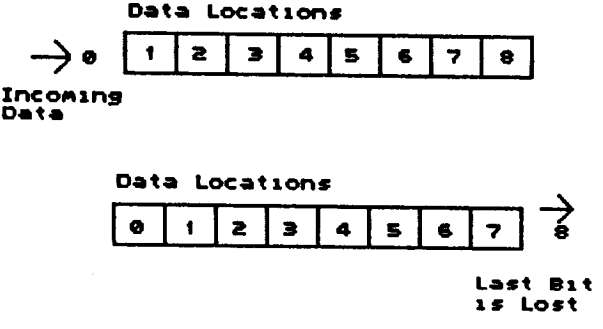


Figure 12.1 - Data Shifting

Every time that the clock goes high, a data bit is pushed into one end of the shift register. This is how eight bits of data are received. Figure 12.2 illustrates a typical data transfer.

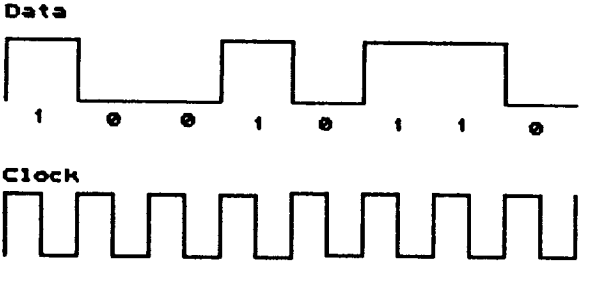


Figure 12.2 - Timing Diagram

Parts List

1 - 74LS164 Shift Register
8 - LEDs 8 1K Resistors
1 - 9-Pin Connector

To get the serial data from the computer, two bits of the joystick port are used. Bit one of the port is used for the clock and bit two of the port is used for the data. Figure 12.3 is the schematic diagram of the circuit.

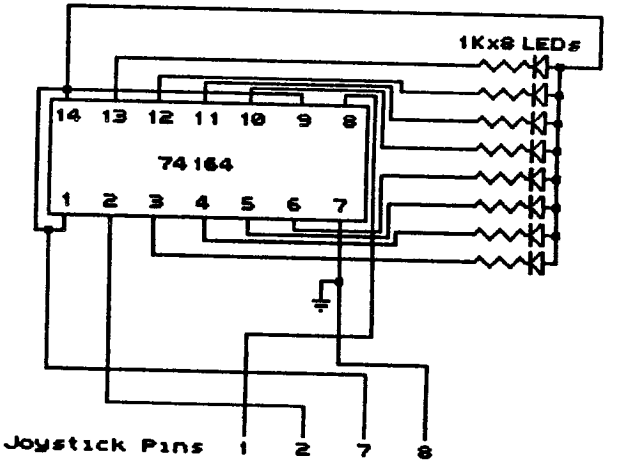


Figure 12.3 - Shift Register Schematic

The demonstration program called SHIFT8 accepts an input from 0-255 and then sends it out over the two bits of the joystick port in a serial manner. Another demonstration program called SHIFT16

accepts a number from 0-65535, and then sends it out over the two bits of the joystick port in a serial manner also.

Another use for this project is to transfer the data over infrared beams. Figure 12.4 is the schematic for this circuit. Run the program INFRARED to use this circuit.

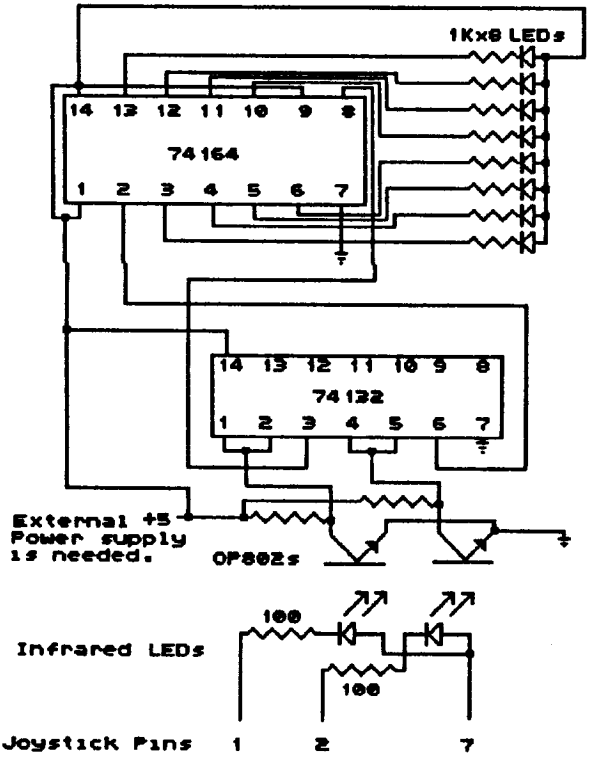


Figure 12.4 -Shift Register with Infrared

13. Data Selectors

There may be a time when your need for more places to input data exceeds what is available. Data selectors may be used to accommodate you if this situation arises.

The principle of the data selector in this chapter is simple. It acts as a set of switches so that you can change from one set of available data to another. There are two methods of data selection possible. One is an external clock that switches at regular intervals and the other is controlled directly by the computer. You can experiment with the computer-controlled selector by using a controlling bit to switch data banks. We will only illustrate the use of a data selector controlled by an external clock.

The circuit switches between two 4 bit sets of data. These are presented to the joystick port which uses an external clock to perform the switching. The computer knows which data set it is seeing because the trigger sees the clock state. When the clock is high, the trigger sees this and knows how to interpret the data. When the clock is low, it knows that the alternate set of data is being seen. The schematic for this circuit is seen in Figure 13.1.

Parts List

1 - 9-Pin Male Socket
1 - 4011 Quad NAND Gate
2 - 4066 Quad Bilateral Switches
3 - 14-Pin Dip Sockets
1 - 0.47 microfarad electrolytic capacitor
1 - 100K resistor
1 - 1M resistor
2 - 9-Pin Female Plugs
Hookup Wire Breadboard or
circuit board Project box

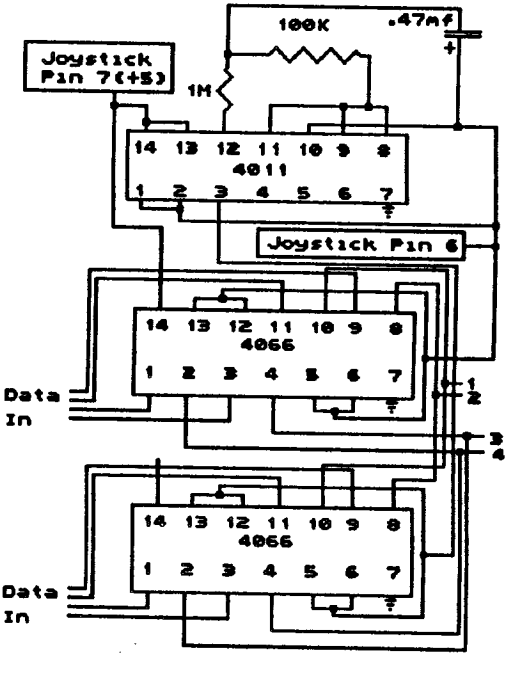


Figure 13.1 - Data Selector

When the device is plugged into joystick port one, connect two joysticks (or any other data altering devices) to see the data on both data banks. You can then see the status of both joysticks if you read the port. A demonstration program is included called OVERLY2. This program will show the status of both sets of data. To use this program, you must first rename it to AUTORUN.SYS. After rebooting the computer, the overlay will appear at the top of the screen. You will see if the device is toggling or not because it will say TOGGLE! followed by a yes or a no on the overlay.

The overlay will work independant of your BASIC programming. This is because it is updated through a vertical blank interrupt. You can turn the overlay off by pressing the [SELECT] console key. To turn the overlay back on again, press the [START] console key.

14. Analog Data

You may be wondering what the difference between analog and digital data is. Digital data is a set of ones and zeros, a set of transistors in either a high or a low state. The low state approaches zero volts, and the high state approaches five volts. These ones and zeros form the digits of a binary number. A set of four of these digits comprise a nibble, a set of eight of these digits comprise a byte, and a set of sixteen of these digits comprise a word.

An analog signal may be at zero or five volts, but it may also be anywhere in between those two voltage levels. Of course, analog signals can be at any voltage level above five volts, but for our discussion we will limit ourselves to a range of zero to five volts. The computer cannot interpret analog voltage levels. For this reason, analog to digital converters are used. The analog voltage is converted to a digital number and the microprocessor can then use it in the way that the software calls for.

Every project to this point in this book has been a digital project. The information dealing with these projects spoke of bits that were either on or off. Now, we need to open the door to projects that use analog data.

Every joystick port has two pins devoted to analog data. These pins might be better known to you as the paddle input pins. Inside the computer are built-in analog to digital

converters that interpret this data and transform it into a digital form that the computer can understand. The paddles that can be connected to these pins are nothing more than potentiometers. By turning the paddle knob, the resistance is changed as well as the resulting voltage. The internal analog to digital converter then translates this voltage into a digital number and allows the microprocessor to use it accordingly.

For your first experiment, it would be a good idea to connect a simple potentiometer to the first paddle pin of joystick port one. This will be pin five. You can then run a demonstration program called POTMETER and see how it can be interpreted. Figure 14.1 has a schematic of the potentiometer that you will use. You can also try the BASIC listing of Figure 14.2 to see the values of the analog data being read.

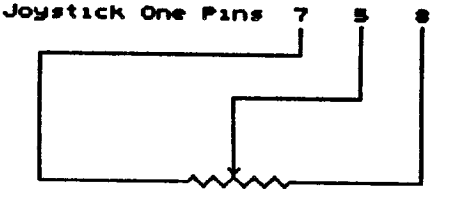


Figure 14.1 - Potentiometer

```
10 X=PADDLE(0)
20 POSITION 20,11
30 PRINT X;"   "
40 GOTO 10
```

Figure 14.2 - BASIC Listing

There is a small problem with the Atari operating system that prevents values of 229-255 from being returned. This also prevents voltage values of 0-1.5 volts (note that this upper value of 1.5 volts is approximate and depends upon your actual computer) from being read. This can be a problem when you need to read the lower voltage levels. There is a way to fix this with several external parts. Figure 14.3 (on the following page) is the schematic of this fix.

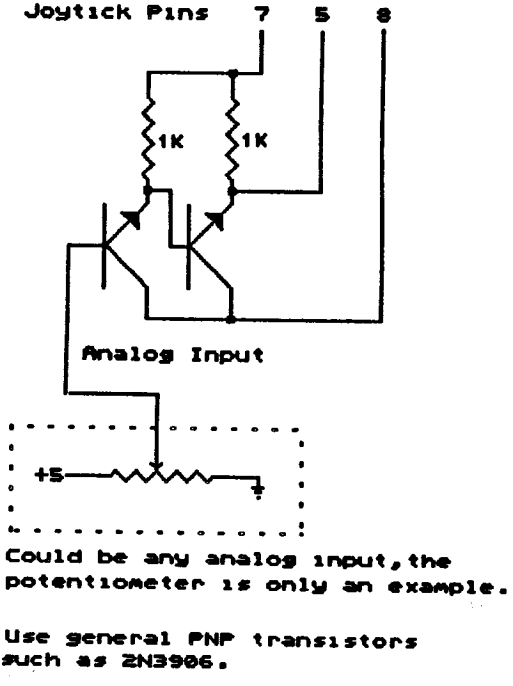


Figure 14.3 - Paddle "Fix"

15. Synthesizer Add-Ons

The eight bit Atari computers have a sound chip that is adequate for most applications. This chapter will provide ways of adding more capabilities to the already present sound chip. These additions can also be enabled in a remote location. The remote use of these new sound generators might be your most useful application of these ideas.

The first project is a simple oscillator. The IC used to produce the frequency is a 555 timer. Two 4066 quad bilateral switches are used to change the frequency. Figure 15.1 is the schematic of the circuit.

Parts List

1 - 9 Pin Male Socket
1 - 555 Timer
2 - 4066 Quad Bilateral Switches
10 - 10K Resistors
10 - 4.7K Resistors
1 - 0.1 microfarad disk capacitor
1 - 1K Resistor

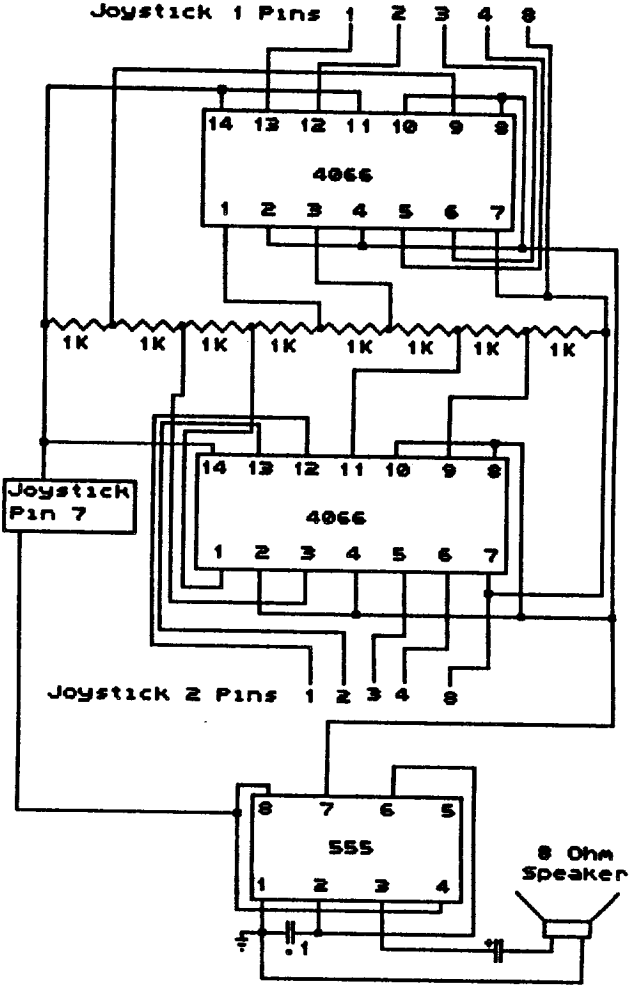


Figure 15.1 - Tone Generator

There is no demonstration software for the previous circuit, but try the listing in Figure 15.2. Any input of the keyboard will alter the frequency that is being produced. Press [ESCAPE] to end the program.

```
10 OPEN #2,4,0,"K"
20 POKE 54018,56
30 POKE 54016,255
40 POKE 54018,50
50 GET #2,X
60 POKE 54016,255-X
70 IF X<>27 THEN 50
80 POKE 54018,56
90 POKE 54016,0
100 POKE 54018,60
110 END
```

Figure 15.2 - BASIC Listing

The previous circuit can be set to a frequency and then left alone. If you want to generate your own waveforms, the following circuit can be used but will have to be continuously updated.

The way the waveform generator works is that a voltage level is produced according to values poked to the joystick register. These values set the bits which then control the 4066 quad bilateral switches. By alternating the voltage levels at a very fast

rate, almost any waveform can be created. Figure 15.3 is the schematic for this circuit.

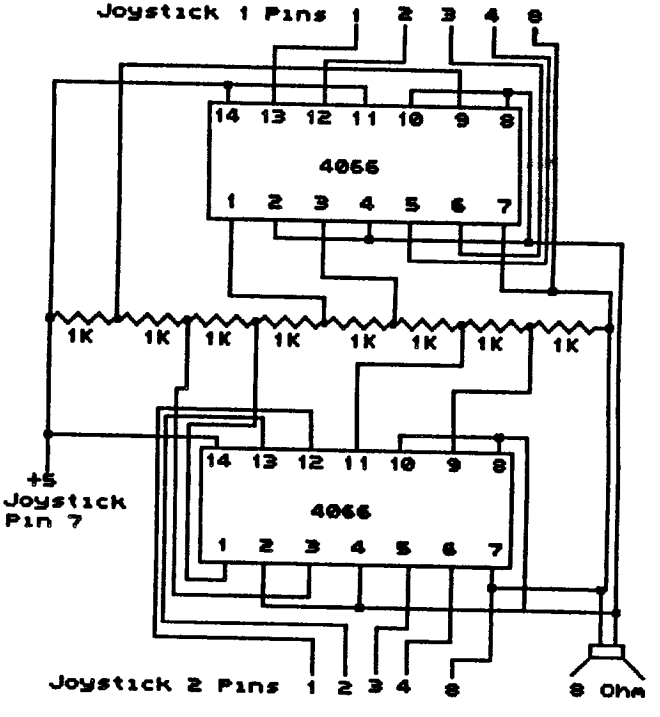


Figure 15.3 - Waveform Generator

The demonstration program for the waveform project is called WAVEFORM. When you run the program, you type in the values of the waveform that you want. You will type in values from 0-255. The greater

the value, the larger the voltage. The smaller the value, the lower the voltage. To produce a waveform, you must alternate between different voltage levels.

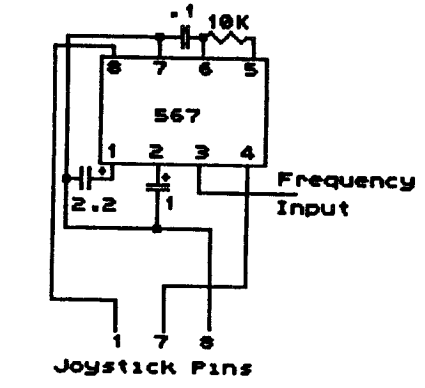
16. Tone Decoders

Tone Decoders

This chapter will discuss two different circuits that will recognize certain frequencies or tones. The first one will utilize preset channels which have outputs that the computer will read. For this circuit, it will be best to build the optional frequency generator. The second design uses a frequency-to-voltage converter. This is then fed into the paddle input allowing the computer to differentiate between the incoming voltage levels.

For the first design, a 567 tone decoder IC was used. This chip, set up in the configuration we used, will respond to a specific preset frequency. When the frequency is detected, the output of the IC, which goes to the joystick port data bit, goes low. The band width of the frequency is very narrow but can be adjusted somewhat with the low-pa capacitor. This is described in the schematic of Figure 16.1.

To input frequency to the 567, you may want to amplify an audio signal. To do this, try a simple microphone. If there is not enough gain, try the schematic in Figure 16.2.



567 Output goes low with
correct frequency.

Figure 16.1 - Tone Decoder

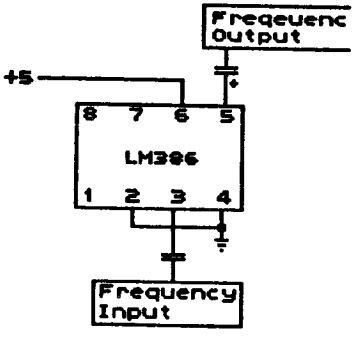


Figure 16.2 - Amplifier

For either of the tone decoder circuits, you can build a frequency generator. The frequency generator can then be carried remotely and pressed to sound desired audio signals. These signals could then be interpreted by the computer, and you could have it perform a task of some sort. Figure 16.3 is the schematic for the frequency generator circuit.

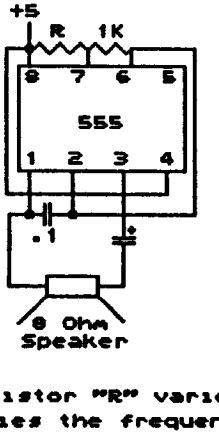


Figure 16.3 - Frequency Generator

The second main type of tone decoding in this chapter is one which uses a frequency-to-voltage converter. The voltage is then fed into the paddle pin and the computer makes the appropriate interpretation. Figure 16.4 is the schematic of this design.

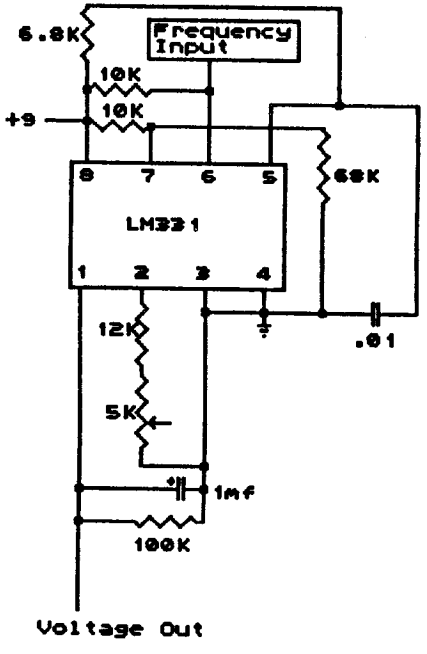


Figure 16.4 - Tone Decoder

The demonstration software provided is a program called FREQUNCY. It is meant to be used with the frequency to voltage converter. Run the program with the hardware attached and then whistle into the microphone or use the frequency generator illustrated in Figure 16.3.

17. Networking

The project in this chapter is very easy to build and has endless applications. The construction only consists of a pair of cables. The programming techniques are easy in BASIC. The machine language subroutine takes care of the actual communication protocol, so you do not have to worry about that. The networking system could be used in many commercial ventures where remote data must be retrieved. A demonstration bulletin board system could also be developed so that modems were not necessary.

The medium for data transfer is joystick ports one and two. Connected between the two ports of each computer are seven small wires. One problem with the joystick ports is that there are only eight bits with which to communicate. Considering that a byte is eight bits, it seems that it is easy enough to send the byte from one computer to another. That is not the case since there are other aspects of parallel data communication to consider.

When computers communicate either internally or externally, they do what is called "handshaking." This handshaking varies from one computer to another and from one type of external device to another. The main idea, though, is that certain bits are used to indicate that data is ready to send, that data is ready to receive, and many other bits of information.

It seems that eight bit parallel communication is possible. The reason that this is apparent is because there are eight data bits available. One thing to remember is that we must be able to alert the other computer that data is waiting. Without this capability, the remote computer, where the data is intended to go, will not know to get the data at the ports.

Additionally, the receiving computer must be able to signal that it has gotten the data. The reason for the necessity of the data received signal can be best illustrated with an example. Suppose that the sending computer presents data to its port with the destination being the remote computer. How long does it keep that data there in order to insure that the receiveing computer has gotten the data? If it has to wait for a time period, then there is a potential waste of time involved.

For the reason described, handshaking is an essential part of parallel data communications. That becomes a problem with our Atari eight bit computer in that we only have a total of eight bits to both write the data and to do the handshaking. To solve this problem, the data can be broken up into nibbles which are four bits each. The nibbles can be sent separately as part of a specific communications protocol leaving four bits available for handshaking. For our system,

we only used two handshaking bits. The other two are in reserve for a multiple computer networking system.

The communication protocol is as follows. Bits five and six of one computer feed bits six and five of the other computer respectively. They are purposefully crossed. Upon initialization of the program, bit six of each computer goes low. In this state, the remote computer knows that there is no data waiting.

When one of the computers is ready to send a byte, it sets bit six of its port to high. At the same time, it puts the upper nibble of the byte on the lower four bits of the register. This high is then seen in bit five of the receiving computer. When the receiving computer sees this, it takes the data from the lower four bits of the register. The receiving computer then sets its own bit six to high so that it can signal that it has received the first nibble. When the sending computer sees that its own bit five is high, it needs to send the other nibble. Don't forget that this is from bit six of the other computer since the wires are crossed. It puts the lower nibble in the lower four bits of the register, and at the same time sets its own bit six to low. When the receiving computer sees its own bit five go low, it knows that the other nibble is waiting. It reads this from the lower four bits and then processes it. Once it has done

this, it sets its own bit six low again. This signals the sending computer that the transfer of that byte is complete.

Figure 17.1 is a diagram of this procedure.

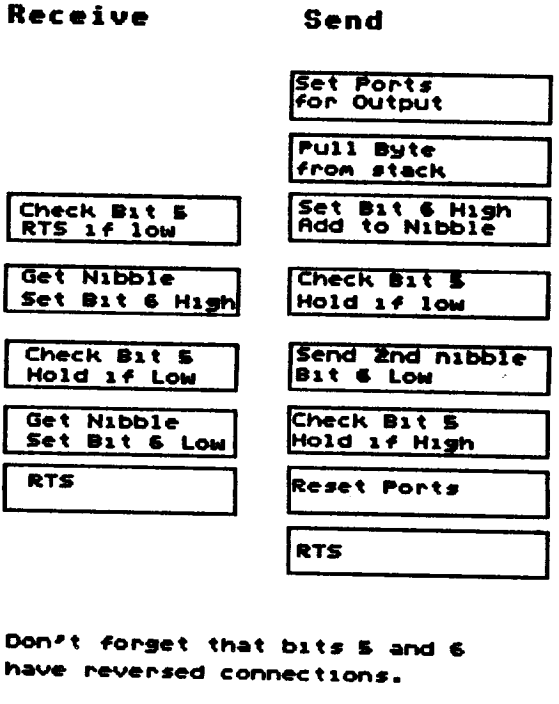


Figure 17.1 - Communication for Networking System

One thing that the receiving computer does while it is getting the data is to look for a backslash character, followed by another byte, and lastly followed by a carriage

return. This indicates that a special function is to follow.

The machine language subroutine does several things in order to communicate with the BASIC program. When it receives a byte of data, it puts it into location 1789. Location 204 is then set to one in order to signal the BASIC program that a byte needs to be processed. Once the BASIC program has read the byte, it must set location 205 to zero so that it will know not to read that byte again.

The second thing that the subroutine keeps track of is the special function prompt. Anytime that the remote types a "/", followed by another character, followed by a carriage return, it indicates a special function is to follow. The subroutine then sets location 205 to one. When this location is seen to be a one, the BASIC program can execute a special function. The character of the special function is found in location 1790. When our BASIC program sees the flag set, it first resets location 205 to zero, and then does a GOTO to the line number corresponding to the ATASCII value of the special character times 100.

Following is a chart of the special function built into the BASIC program at the present time, but many more could be added.

Character	Function
B	Program the remote computer in the BASIC immediate mode or change a line of the BASIC program.
D	Do a drive directory.
E	Extended BASIC functions. It differs from the first BASIC function in that it clears the screen when done. The other type could possibly crash.
R	Read a file on disk.

Figure 17.2 is the wiring diagram for the cable(s).

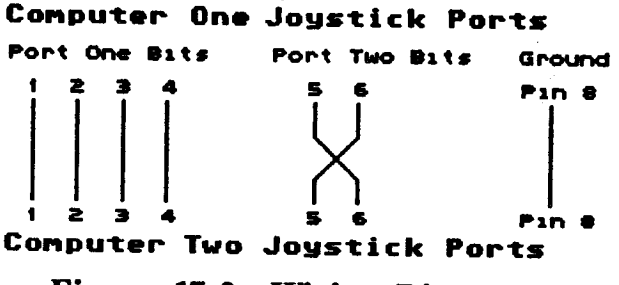


Figure 17.2 - Wiring Diagrams

When you run the program called TALK, make sure that you don't hit [START] until both computers say to do so. The source code for this subroutine is on the disk and is called TALK.SRC.

If you get ambitious, try multiple computers. We gave a demo of this at a Dade Atari Users Group meeting where we had four computers networked. The hardware used was a set of 4066 quad bilateral switches. These switches were controlled by a single computer using the two remaining bits of the joystick port. The controlling logic used was a 74LS164 shift register. Make sure that if you try doing this that only two computers are linked at the same time.

18. Display Lighting

The project in this chapter has been implemented by an acquaintance of ours who uses it for a Christmas light display. Besides that particular usage, one could set up a restaurant sign or accent a store window display. The circuit has eight independently programmable lighting channels. These channels can be turned on and off to produce many different and unique patterns.

The hardware itself is simple to understand but gets tedious when building. The reason that it is so tedious to build is that there are many connections to be made. If you are careful and patient though, your project will turn out well.

The best way to contruct this circuit is on a perforated breadboard. For all of the ICs, you should use sockets. The relays can also be plugged into sockets if you wish. We chose to solder them directly into the circuit.

For each channel, there are three components outside of the computer. The first is an inverting buffer. This prevents any damage to the computer. The second component is a 5 volt reed relay. This relay is then used to enable the high current relay which turns the lights on or off. The large relay will handle three amps. If you need more than that, use a relay with a higher amperage rating. With the three amp capability, you can power up to 360 watts of

lights.

An external power supply was used because the computer itself cannot power the 16 relays that are being used. The external power supply should be twelve volts with a current capability of about 1 amp. We then used a 5 volt regulator so that we could have both 5 and 12 volts.

Since the circuit in its entirety is very extensive, only a single channel is illustrated in Figure 18.1. Duplicate this 8 times to complete the project. Each channel will then control a set of lights.

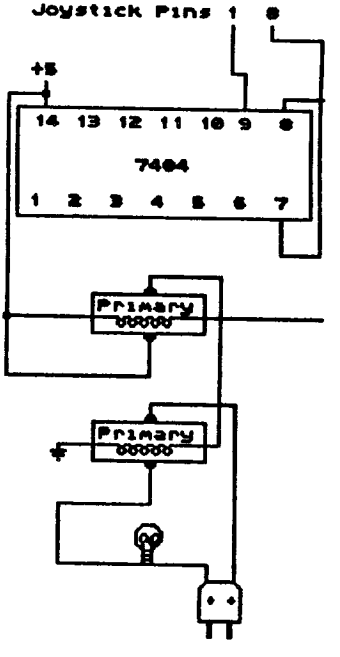


Figure 18.1- Lighting Display Channel

The demonstration program that is included is called LIGHTS. When you run it you will notice a screen display which indicates the channel(s) that are on. This screen display slows down the program. To speed up the program, you can disable the screen display by simply pressing [SELECT]. Pressing [START] will enable it.

The patterns are read from the data statements beginning at line 1000. To change the patterns, change these data statements. Each event is composed of three consecutive data bytes. The first byte is the sum of all bits which will be turned on. The second byte is the duration of the event in fourths of a second. The third byte is either a 1 or a 0. If it is a 1, there is more data to come in the pattern. If it is a 0, the end of the pattern has occurred.

The number of times that each pattern is repeated can also be changed. The data at line 10000 contains this information. Change the data to alter the number of times that each pattern is repeated.

Appendix A:

PC Board Services

KIT CIRCUITS
Box 235
Clawson, MI 48017

EXPRESS CIRCUITS
314 Cothren Street
P.O. Box 58
Wilkesboto, NC 28697

T.O.R.C.C.C
Box 47148
Chicago, IL 60647

Appendix B:

Parts Sources

Active Electronics
P.O. Box 9100
Westborough, MA 01581

All Electronics Corp.
905 S. Vermont Avenue
P.O. Box 20406
Los Angeles, CA 90006

Bigfoot Systems
2708 East Lake Street
Minneapolis, MN

Copper Electronics
4200 Produce Road
Louisville, KY 40218

Diamondback Electronics
P.O. Box 12095, Dept. 102
Sarasota, FL 33578

DIGI-Key
Highway 32 South
Thief River Falls, MN 56701

Heathkit
Dept. 021-892
Benton Harbor, MI 49022

Hitachi America Ltd.
2210 O'Toole Avenue
San Jose, CA 95131

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002

JDR Microdevices
1224 S. Bascom Avenue
San Jose, CA 95128

Mark V Electronics
248 East Main Street
Suite 100
Alhambra, CA 91801

Mouser Electronics
2401 HWY 287 North
Mansfield, TX 76063

R.F. Electronics
1056 N. State College Blvd.
Dept. R
Anaheim, CA 92806

Appendix C:

Programs

DECODE - Used to demonstrate the decoder circuits.

DETECT - Used to show the state of external detectors.

DLDISPLY - Shows display list information.

ENCODE - Used to show the state of encoded external logic.

FREQUNCY - Decodes the frequency with the external hardware.

INFRARED - Used with the infrared version of the serial data project.

LGHTMUSC - Used to demonstrate the light pen.

LIGHTS - Lighting display program.

LOGIC - Logic gate and truth table demonstration.

OVERLY1 - Overlay showing port information.

OVERLY2 - Overlay showing port information.

OVERMAKE - Overlay maker utility.

PENDULUM - Calculates the length of the pendulum string and the period and frequency.

POTMETER - Demonstrates the use of a potentiometer.

SHIFT8 - Sends an 8-bit number serially to the shift register project.

SHIFT16 - Sends a 16 bit number serially to the shift register project.

SPDACCEL - Calculates the speed and acceleration of a moving object.

SPEED - Times and calcluates the speed of a moving object.

STOPWTCH - Stopwatch demonstration.

SWITCH - Switch demonstration.

TALK - Networking program.

USRMAKER - Utility to adapt machine language subroutines to BASIC programs.

WAVEFORM - Used to create waveforms.

Source Codes:

OVERLY1.SRC - Source code for the first overlay program.

OVERLY2.SRC - Source code for the second overlay program.

SHIFT.SRC - Source code for the communication routine for both SHIFT programs.

SH8.SRC - Source code for the SHIFT8 display.

SH16.SRC - Source code for the SHIFT16 display.

TALK.SRC - Source code for the networking communication routine.

Index

"Your Atari 8 Bit Comes Alive!"

Eighteen chapters of exciting hardware projects to build for your eight bit Atari. The novice to the electronics expert can learn from and enjoy these projects as the instructions are step-by-step. Background information in each chapter allows the reader to fully understand the projects.

Demonstration Software Is Included!

A disk of working programs is included in BASIC. The source code for those programs that use machine language subroutines is also included.

The equipment needed is any 8 bit Atari computer, a disk drive, and a monitor or a television.

Here is what is included in this comprehensive book:

- The introductory ELECTRONICS chapter goes into great detail as a learning experience for the novice and as a review for the electronics expert.
- The LIGHT PEN chapter shows how to build and use a light pen.
- The NETWORKING chapter shows how to network two or more Atari 8 Bit computers.
- The SWITCH chapter involves a demonstration of logic gates and includes truth tables, electronics schematic symbols, and interpretations of cases.
- The SERIAL DATA chapter demonstrates the use of serial data and provides an excellent demonstration of its implementation. This project can also be altered to communicate over infrared light beams.
- The FREQUENCY chapter gives several examples of decoding frequencies which are in the ambient surroundings.
- The SYNTHESIZER ADD-ONS chapter illustrates several devices that will extend the sound capabilities of your computer.
- The LIGHTING DISPLAY chapter shows how to control display lights for setting up restaurant signs, store window displays, or even Christmas light displays.
- Other chapters discuss event detectors, data encoders and decoders, 120 volt circuit controllers, and much more.

Complete with disk of working programs!

IMPORTANT!

Computer Spectrum, Inc. was formed for the creation of unique quality products for personal computers. Please fill out and return this card, so that we may keep you informed of any updates or new products as they become available. This card will also allow you to provide us with information on products that you would like to see created by Computer Spectrum.

If you need additional space, please feel free to attach additional pages, or contact us through our BBS, at (305) 251-1925. Thank you for purchasing this product, and for your time in completing this questionnaire.We look forward to helping you.

Sincerely,

David B. Leinecker

David B. Leinecker, V.P., Marketing
Computer Spectrum, Inc.

Place of Purchase
Name _____
Address _____
City _____
State _____ Zip _____
Phone

Name _____
Address _____
City _____
State _____ Zip _____ Age _____
Phone _____ Date _____
Product will be used on: _____ computer system with _____ K memory.
Primary Use of Computer? _____

How do you rate this book? (Hi) 10 9 8 7 6 5 4 3 2 1 (Lo) (Circle one)
Why? _____

_____

How do you rate the demo programs? 10 9 8 7 6 5 4 3 2 1 (Circle one)
Why? _____

What was your favorite project? _____

_____

How did you find out about this product? ( ) Magazine Ad ( ) Magazine Review
( ) Saw in store ( ) User Group/Club ( ) Computer Spectrum BBS ( ) Other BBS
( ) Friend ( ) Other

Please indicate any of the following magazines which you read regularly (at least 3 of the last 5 issues)?
( ) Antic ( ) Analog ( ) Compute! ( ) STart ( ) ST-Log ( ) Compute!'s Atari ST
( ) Atari JOURNAL ( ) Current Notes ( ) Family Computing ( ) Byte ( ) PC World
( ) Other (please indicate)

What types of products for the personal computer would you like to see more of?
( ) Arcade Games ( ) Educational Games ( ) Hardware Project Books ( ) SAT Prep
( ) Others
If you checked Hardware, what would you like to see? _____

_____

Do you belong to a users group, and if so, who is your contact person? ( ) Yes ( ) No
Name _____ Address _____
City, State _____ Zip _____ Phone _____
Would you like to receive new product announcements? ( ) Yes ( ) No

Important!!!

Please Fold on this line

From:

_____

_____

_____

Place Postage Here

Computer Spectrum, Inc.
P. O. Box 162606
Miami, FL 33116

Attention: Customer Support

Please Staple Here