Using the PC Parallel Port in Digital Systems Lab Exercises

Cecil E. Beeson, P.E., Assistant Professor

University of Cincinnati Clermont

Introduction

This paper will describe a multi-phase project to utilize the functionality and accessibility of the PC parallel port to augment traditional college electronics lab exercises. The opportunity for students to use the ubiquitous PC to exercise and control digital circuits designed in a college lab can be a valuable learning experience.

At Clermont students are introduced to PC fundamentals early in their college studies. As these students pass through the Electrical Engineering Technology (EET) program, they construct their circuits and typically view the PC only as an analytical tool to solve engineering problems. Of course, in the real world, PCs are capable of controlling all types of devices.

After reading about parallel port programming (PPP) in a magazine a few months ago, the author of this article did some research and decided to introduce students in a Digital Systems course to this fascinating concept. The response from the students was impressive.

Value to EET Students

The value of using PCs in this manner is significant. EET students can grow in, minimally, five dimensions of technology:

- Continue to develop skills through traditional studies in the EET area
- Become more mature users of PCs by becoming more informed of PC capabilities and structure through the integration of PCs into the lab environment
- Expand their knowledge of software development because of the need to write programs to control the PC and external electronic devices
- Become more aware of user interface issues through careful design of input fields and screens displays
- Acquire additional technical test skills because of interactions between PC and external electronic circuitry

Although not exclusively tied to PPP, two additional benefits can be realized. Students will be required to perform research and since most information is available over the Internet, their knowledge of web technology should increase. Also they will be required to prepare a written report; communication skills will be improved too.

A Humble Beginning

To illustrate the power of PPP, two relatively simple circuits (one with output only and one with inputs) were constructed on breadboards. These circuits were chosen to illustrate the basic principles of parallel port input/output capabilities and associated software. Both projects were designed and demonstrated by the instructor.

For an output circuit configuration 8 LEDs were connected to the parallel port and software was written to exercise the LEDs in various sequences and combinations. *Figure 1A* shows the circuit diagram[1]. *Figures 1B, IC* and *1D* show three of several programs that were used to exercise the LEDs. Program #1, written in MSW Logo[2], counts from 0 through 255 and outputs the binary equivalent numbers on the LEDs. Program #2, written in QBasic, is similar to program #1 except that each count is delayed by approximate 0.25 seconds. Program #3, written in C++, accepts a decimal number from the keyboard and displays the corresponding binary number on the LEDs.

To illustrate input to the parallel port, eight DIP switches were used to control whether or not the corresponding LEDs should be illuminated or extinguished. *Figure 2A* shows the circuit diagram for one technique to input 8-bit quantities from an external device. Another technique will be mentioned later in this paper. *Figure 2B* shows a Logo program that polled the switches and decided which LEDs should illuminated or extinguished. Please note the various comments that have been inserted into the program.

Inputting a byte through a standard parallel port is somewhat challenging. Because some of the pins are open collector (OC), they must be brought high before being used as inputs. In the program shown in *Figure 2B*, the Logo statement – outportb 890 4 – performs this function (where 890 is the decimal equivalent of the hex address for the status register, 037A, and decimal 4 is equivalent to 0100 in binary). Bits D3, D1 and D0 are hardware inverted and are pulled high by outputting a zero (0) and bit D2 is pulled high by a logic one (1); hence the 0100 output. Gathering the byte of data from the switches is interesting. The lower 4 bits of the switch input are accessed through the control port and the upper 4 bits are available through the status port. After inputting both 4-bit quantities and masking unwanted bits, the two nibbles are ORed together to form the final byte of information.

Computer programs, MSW Logo, QBasic and C++, were written for most of the configurations. Logo and QBasic were selected because of their relative simplicity, ease of use and the availability of instructions to permit direct access to hardware ports. A very short tutorial on MSW Logo is contained in *Appendix A*. In addition QBasic is the language that most EET students use in their first required computer information systems class. Although inherently more complex and flexible than Logo and QBasic, C++ was used because it is so popular in "real world" applications and technology students should be exposed to it.

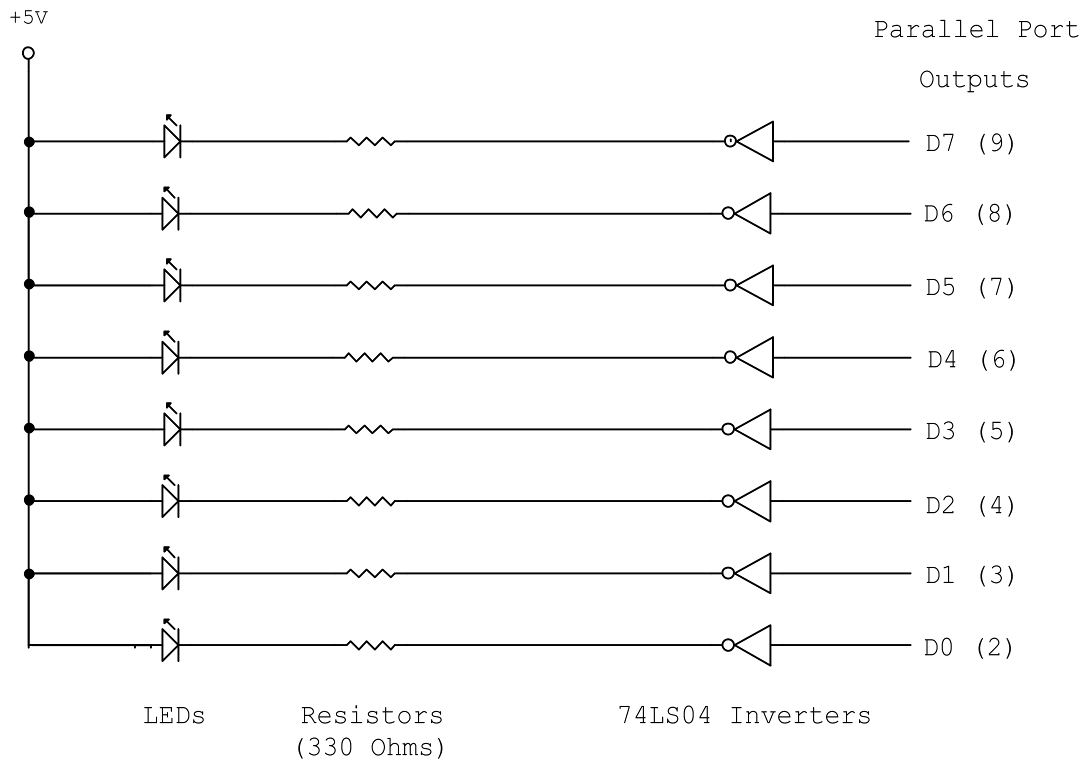


Figure 1A **– Circuit to Display Parallel Port Outputs**

```
; Display 0 – 255 on LEDs

to flasher
        make "x 0
        repeat 256 [
                show :x
                outportb 888 :x
                wait 15
                make "x :x + 1
                ]
        outportb 888 0
end
```

Figure 1B **- Program #1 (Logo)**

```
FOR i = 0 TO 256
        ts = TIMER
       OUT &H378, i
    here:
       te = TIMER
       td = te - ts
      IF td < .25 GOTO here
NEXT i
OUT &H378, 0
```

Figure 1C **- Program #2 (QBasic)**

```
/ Illuminate selected LEDs (8) with an input #

#include <conio.h>
#include <iostream.h>

int _outp(unsigned short port, int databyte);

#define DATA 0x378
#define STATUS 0x379
#define CONTROL 0x37a

int inval = 0;

void main(void)
{
        while(inval < 256)
        {
                cout << "Enter a decimal #\n";
                cin >> inval;
                _outp(DATA, inval);
        }
}
```

Figure 1D **- Program #3 (C++)**

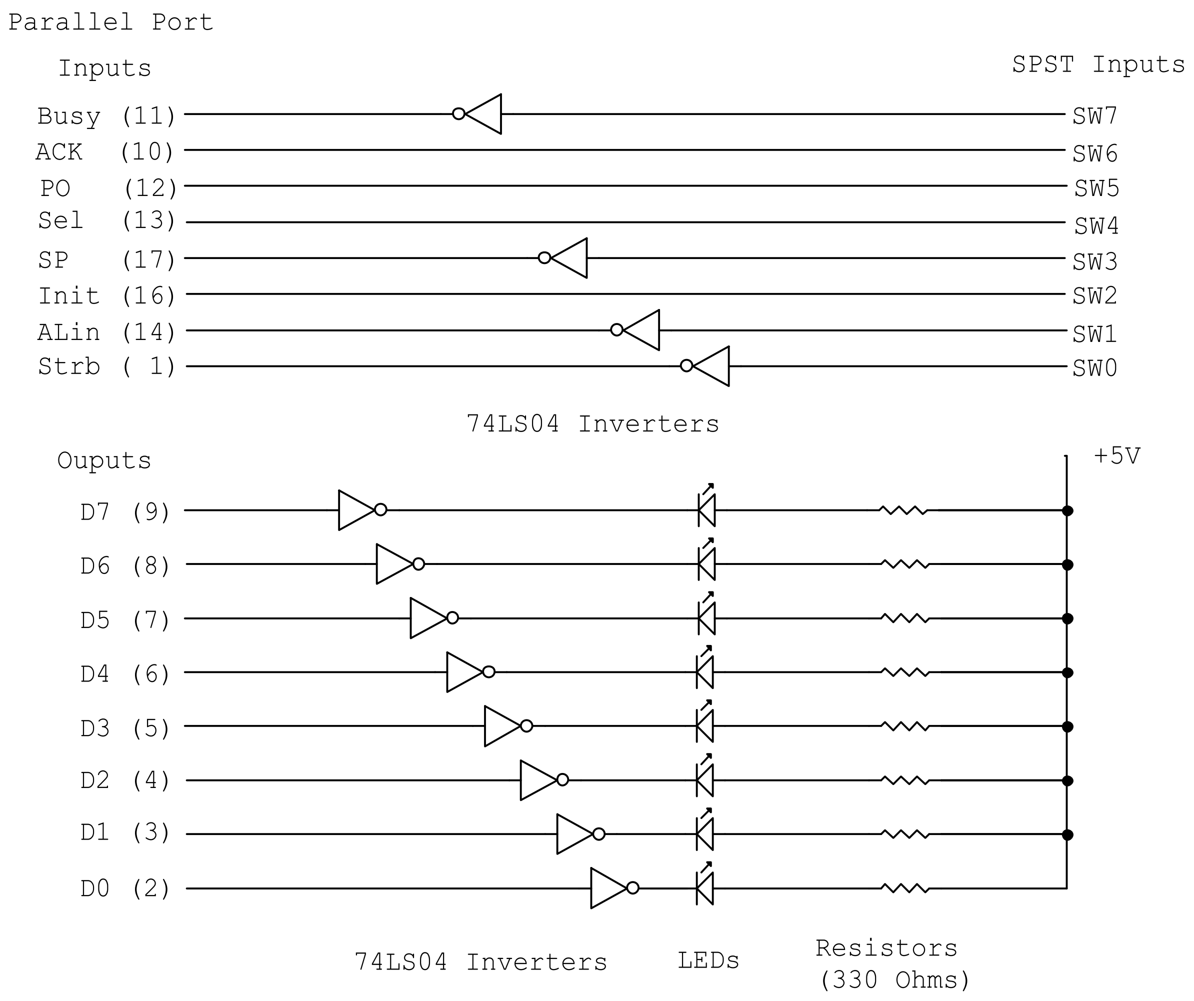


Figure 2A **– Displaying Switch Inputs on LEDs**

```
to read8m1
        cs
        forever [
                outportb 890 4                  ; Bring OC pins high
                wait 60                         ; Delay program operation for 1 sec
                make "low inportb 890           ; Input D3 – D0 via control port
                make "high inportb 889          ; Input D7 – D4 via status port
                make "low bitand :low 15        ; Mask upper nibble of low order byte
                make "high bitand :high 240     ; Mask lower nibble of high order byte
                make "byte bitor :high :low     ; Combine lower and upper nibbles to form byte
                outportb 888 :byte
                ]
end
```

Figure 2B **– Logo Program for Switch Input and LED Output**

The demo circuits and a software tutorial were discussed in class. Most of the students were unaware of the PC parallel port interface capability. Some of the students formed into small groups and proceeded to experiment with the circuitry and software.

One group automated a process to test AND, NAND, OR and Exclusive OR chips. All possible combinations of logical inputs were generated and routed through the chips and the final operational status of the chip was indicated by a green LED (chip functioning normally) or a red LED (chip defective). The group used both Logo and QBasic for the testing process.

A Few Technicalities of the Parallel Port

A full report on parallel port technology would be immense and beyond the scope of this paper. Many sources are available for learning and reference[3-7]. Although several forms of parallel ports are available, for reasons of simplicity for this project, each PC was assumed to have a standard port and the data bits were not bi-directional. Other possibilities include extended capability ports (ECP) and extended parallel ports (EPP).

The parallel port is located at the rear of a PC (*Figure 3A*)[8] and has a 25-pin female socket (*Figure 3B*)[8]. In addition there are three 8-bit registers: data, status and control. For most PCs the data register is located at address 0378H. The status register has an address of 0379H and the control register's address is 037AH. To determine the assignments for a specific machine, one could use the DOS debug utility to examine the contents of memory address 0040:0008H, which contains the address of LPT1, or the parallel port. For example:

```
C:\>debug
-d 0040:0008 L2
0040:0000                   78  03                      x.
-q
C:\>
```

In Intel nomenclature the address would read 0378H.

Table 1 lists important attributes of a parallel port.

Another technique to input an 8-bit quantity (from switches in this experiment) is shown in *Figure 4A*. A multiplexer is used to select the most significant nibble first and then the least significant nibble. The two nibbles are then inverted appropriately and reformed into a single 8-bit item. This technique avoids problems with OC technology but it is slower because two reads and one write are required to input a complete byte of data. To illustrate additional Logo commands, the status of each switch was shown visually on a monitor screen by selectively painting circles with red to designate an open switch and green to designate a closed switch. A picture of the screen is shown in *Figure 4B*. In this picture switches 0, 4, 6 and 7 (data bits D0, D4, D6 and D7 are green) are closed; the other switches are open. A Logo program is shown in *Figure 4C*. This procedure was not initially implemented because of the introduction of mutiplexer hardware and more complex commands to create the screen display.

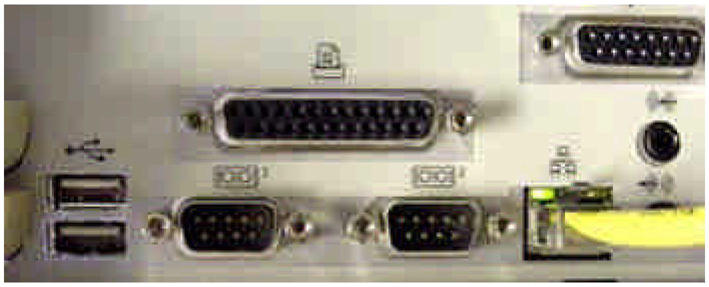


Figure 3A – **View of Parallel Port Socket**

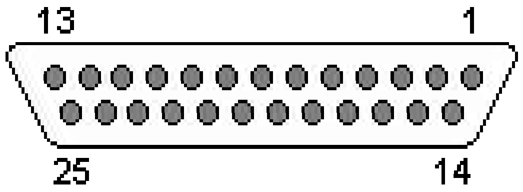


Figure 3B – **Pin Arrangement for DB25 Port**

Pin #	Description	Abbreviated Description	In/Out	Inverted Logic	Open Collector	Register Assignment
1	Strobe	Strb	In/Out	Yes	Yes	Control
2	Data 0	D0	Out			Data
3	Data 1	D1	Out			Data
4	Data 2	D2	Out			Data
5	Data 3	D3	Out			Data
6	Data 4	D4	Out			Data
7	Data 5	D5	Out			Data
8	Data 6	D6	Out			Data
9	Data 7	D7	Out			Data
10	Acknowledge	ACK	In			Status
11	Busy		In	Yes		Status
12	Paper Out	PO	In			Status
13	Select	Sel	In			Status
14	Auto Feed	ALin	In/Out	Yes	Yes	Control
15	Error		In			Status
16	Initialize	Init	In/Out		Yes	Control
17	Select Printer	SP	In/Out	Yes	Yes	Control
18 - 25	Ground		NA	NA	NA	NA

Table 1 – **Selected Data on Parallel Port Connections**

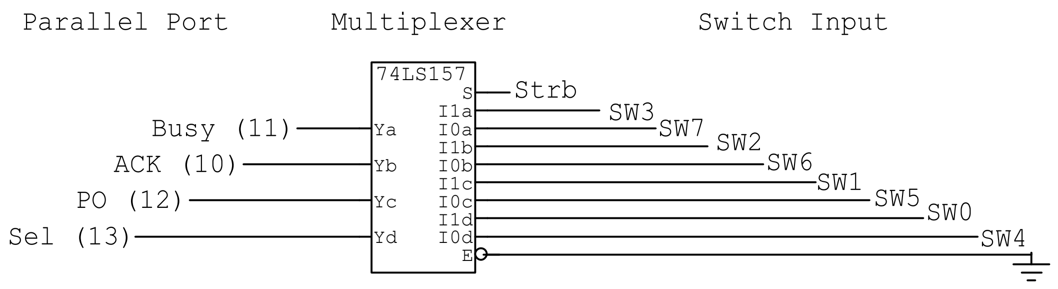
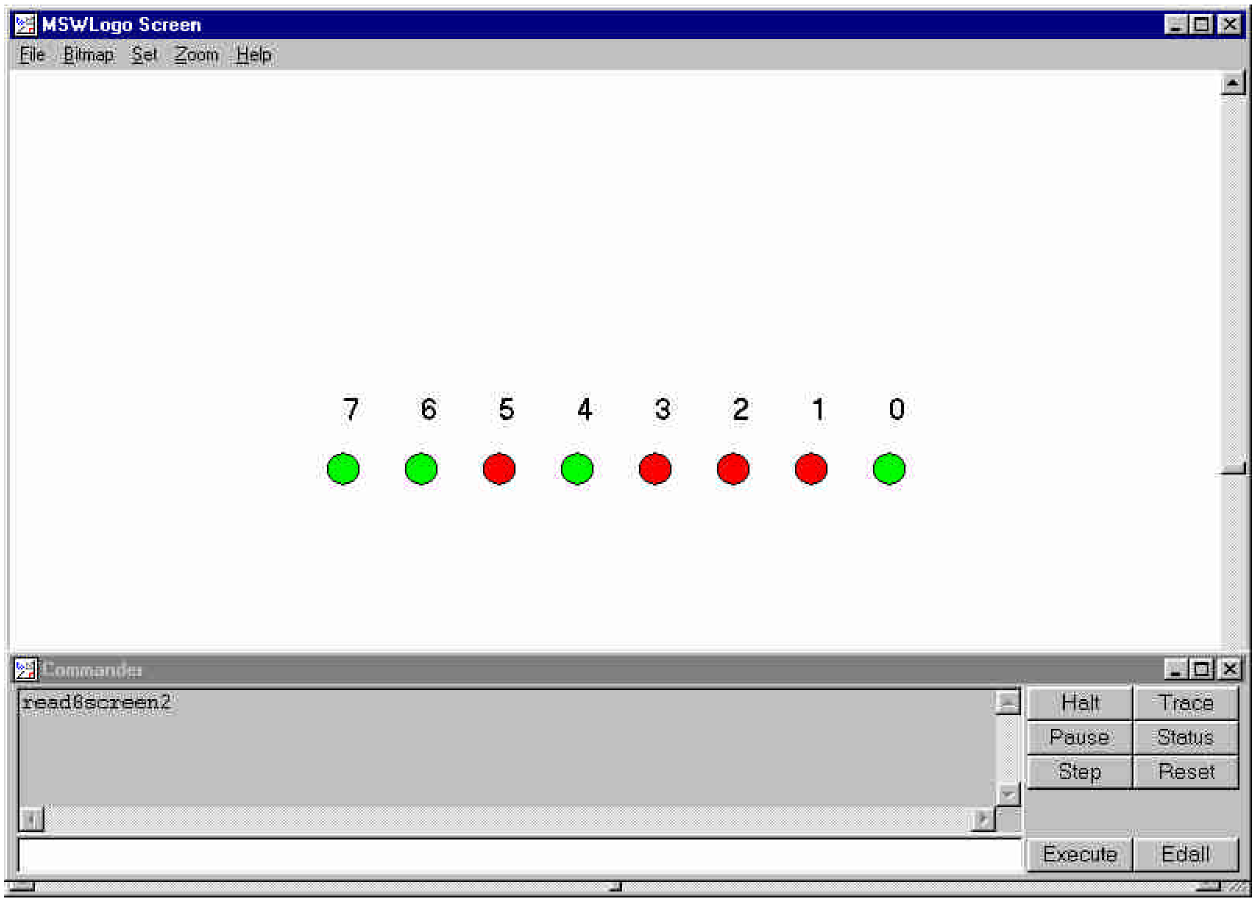


Figure 4A **– Switch Input Using a Multiplexer**



```
read8screen2
```

Figure 4B **– Screen Display of Switch Inputs**

```
to off
      setfc [255 0 0]
      fill
end

to on
      setfc [0 255 0]
      fill
end

to read8screen2
      cs
      ht
      make "led 7
      pu
      make "x -175
      make "y 50
      repeat 8 [
            setxy :x :y
            pd
            label :led
            pu
            make "y :y - 50
            setxy :x :y
            pd
            circle 10
            pu
            make "x :x + 50
            make "y :y + 50
            make "led :led - 1]
      forever [
            outportb 888 0
            wait 6
            make "low inportb 889          ; Input lower nibble
            wait 6
            make "low bitand :low 240
            make "low ashift :low -4
            make "low bitxor :low 8
            outportb 888 1
            wait 6
            make "high inportb 889         ; Input high nibble
            wait 6
            make "high bitand :high 240
            make "high bitxor :high 128
            make "byte :low
            make "byte bitor :high :byte   ; Form data byte
            make "x 175
            make "y 0
            setxy :x :y
            make "mask 1
            repeat 8 [
                  make "testit bitand :byte :mask
                  ifelse :testit > 0 [on] [off]
                  make "x :x - 50
                  setxy :x :y
                  make "mask ashift :mask 1]]
end
```

Figure 4C **– Logo Program to Input Switch Data by Multiplexer and Display on Screen**

What's Next?

Starting with the Winter quarter, January 2003, the PPP project will be introduced to an intermediate level digital systems course. Projects currently in the planning stages include:

- Integrating a PC with counter circuits
- Generating various types of waveforms
- Interfacing with PLDs (programmable logic devices)
- Operating digital subsystems consisting of decoders, encoders, multiplexers, logic function generators, tristate registers and memory devices
- Assignment of challenging team projects involving many types of digital devices will be assigned and appropriate written and oral reports required

During the Spring 2003 quarter the most advanced digital systems course will be offered. This course contains more rigorous coverage of microprocessors and microcontrollers in the design process. Possible projects could include:

- Circuitry involving more advanced sensing devices
- Implementation of wireless communications between a base, or control, station and stationary external devices
- Wireless communications with mobile devices
- Incorporation of GPS technology
- Integration of interrupt capability of the parallel port into circuit design and operation
- Interfacing external microcontroller circuits
- Extensive use of PC and microcontroller assembler language programs
- Explore use other computer languages such as Java and Visual Basic
- Use of optoelectronic devices
- Control of DC motors through use of H-bridges and the PWM (pulse width modulation) technique
- Controlling circuits via the Internet
- Increased emphasis on screen design and user interface principles

A Few Final Thoughts

The parallel port concept was introduced in a digital systems course during the Spring 2002 quarter. Students were generally surprised to learn that a PC parallel port could be used to provide an interface with external circuits designed in a college digital systems course. Apparently several were under the impression that this interface could only be implemented with sophisticated equipment and highly trained technical personnel. They were also delighted for an opportunity to exercise their computer programming skills in a very new environment rather beyond a traditional computer information systems class. After analyzing early results, this project will probably become a permanent part of the digital systems program.

References

1. Circuit diagrams drawn with CircuitMaker®, Protel Technology, Provo, UT

2. MSW Logo software and tutorial/reference manual have been made freely available by George Mills at website, www.softronix.com

3. Parallel Port Complete, Jan Axelson, Lakeview Research, ISBN 0-9650819-1-5

4. Interfacing the Standard Parallel Port, Craig Peacock, www.senet.com.au/~cpeacock

5. Use of a PC Printer Port for Control and Data Acquisition, Peter H. Anderson, Department of Electrical Engineering, Morgan State University

6. IBM Parallel Port FAQ/Tutorial, Zhahai Stewart, home.rmi.net/~hisys/parport.html

7. Controlling the World with Your PC, Paul Bergsman, HighText Publications, ISBN 1-878707-15-9

8. Pictures copied from materials at website: www.howstuffworks.com

9. Logo Foundation: el.media.mit.edu/logo-foundation/index.html

Appendix A

And Now a Few Words about MSW Logo Language

Since Logo programs are featured so prominently in this article and some readers may not be familiar with the language, perhaps a few words would be appropriate. We'll use **Program #1** (*Figure 1A*) as a simple example:

```
Line 1:  ; Display 0 – 255 on LEDs
Line 2:
Line 3:  to flasher
Line 4:            make "x 0
Line 5:            repeat 256 [
Line 6:            show :x
Line 7:            outportb 888 :x
Line 8:            wait 15
Line 9:            make "x :x + 1
Line 10:           ]
Line 11:           outportb 888 0
Line 12: end
```

First, line numbers have been added to provide references to specific commands; they are not part of the language.

Before launching into the syntax and format of Logo, you should know the overall function of this little program. It generates all numbers from zero through 255 and outputs the binary equivalent on 8 LEDs, returns to zero and halts. These 256 binary combinations are possible since $2^8 = 256$, or one byte can accommodate 256 different numbers (0 through 255).

Line 1 is a comment line. The semicolon causes Logo to disregard all characters from that point until a new line character is detected. Careful insertion of comments can make a program much easier to read and understand. Line 2 is merely a blank line and can be used to logically separate parts of a program. Blank lines are ignored by Logo.

Logo programs are comprised of one or more procedures. In this program the procedure name is flasher (line 3) and it is a user-defined name. There are many library procedures provided by Logo.

Execution of line 4 causes a variable, x, to be assigned a value of zero (0). This command is functionally equivalent to a more readable form found in other languages: x = 0.

Line 5 causes a series of commands to be executed 256 times. The statements that are executed are located between the opening and closing brackets, [ ].

Line 6 displays the current value of x in the lower, or commander, portion of the MSW Logo screen. The colon is necessary so that the value of x will be displayed. Without the colon, the character x, not its value, would be visible.

The command, outportb, in line 7 outputs the current binary value of x to port number 888, which is the decimal number for the parallel port address. Most languages permit the use of hexadecimal numbers but Logo does not. Decimal 888 is more recognizable as 0378H to practitioners in the PC field.

The wait 15 statement on line 8 temporarily suspends program operation for approximately 0.25 seconds. The value of 15 is a measure of the number of $\frac{1}{60}$ths of a second; 15 times this number is approximately 0.25 seconds. Because of the many other concurrent tasks required of the PC, this wait time is not exact. This statement was inserted to allow the external circuitry to stabilize before continuing.

The value of x is incremented by one in line 9. In other languages, e.g., C++, a value can be incremented by using: $x = x + 1$.

The bracket in line 10 is the closing symbol for the repeat statement in line 5.

When the 256-loop iteration cycle has been completed, line 11 is executed. This command merely assures that the 8 external LEDs are extinguished.

The end statement on line 12 is a required statement to indicate to Logo that program coding is complete.

This is a very brief and high-level look at MSW Logo. There are also constructs for Windows type programs – radio buttons, command buttons, list boxes, message boxes, to name just a few. A Logo program and a screen layout to illustrate the use of command buttons are shown in *Figures 5A* and *5B*, respectively. This program is much more complex than earlier Logo programs listed in this article. The circuit used to test this program was shown previously in this paper in *Figure 1A*.

To create a Logo program file, one can either use the Logo edit function or use some other ASCII editor and then load the resulting file into Logo. If the latter technique is used, the file must be saved with an extension of .lgo.

Two excellent sources of information about Logo are listed in the References section[2,9] of this paper.

```
to ledbuttons
      cs
      ht
      outportb 888 0

      windowcreate "main "mw "ledbuttons 0 0 100 250 []

      staticcreate "mw "ms7 [LED 7] 10  22 20 20
      staticcreate "mw "ms6 [LED 6] 10  47 20 20
      staticcreate "mw "ms5 [LED 5] 10  72 20 20
      staticcreate "mw "ms4 [LED 4] 10  97 20 20
      staticcreate "mw "ms3 [LED 3] 10 122 20 20
      staticcreate "mw "ms2 [LED 2] 10 147 20 20
      staticcreate "mw "ms1 [LED 1] 10 172 20 20
      staticcreate "mw "ms0 [LED 0] 10 197 20 20

      buttoncreate "mw "SevenOn "ON 40  20 15 15 [make "LED bitor :LED 128
outportb 888 :LED]
      buttoncreate "mw "SixOn   "ON 40  45 15 15 [make "LED bitor :LED  64
outportb 888 :LED]
      buttoncreate "mw "FiveOn  "ON 40  70 15 15 [make "LED bitor :LED  32
outportb 888 :LED]
      buttoncreate "mw "FourOn  "ON 40  95 15 15 [make "LED bitor :LED  16
outportb 888 :LED]
      buttoncreate "mw "ThreeOn "ON 40 120 15 15 [make "LED bitor :LED   8
outportb 888 :LED]
      buttoncreate "mw "TwoOn   "ON 40 145 15 15 [make "LED bitor :LED   4
outportb 888 :LED]
      buttoncreate "mw "OneOn   "ON 40 170 15 15 [make "LED bitor :LED   2
outportb 888 :LED]
      buttoncreate "mw "ZeroOn  "ON 40 195 15 15 [make "LED bitor :LED   1
outportb 888 :LED]

      buttoncreate "mw "SevenOff "OFF 65  20 15 15 [make "LED bitand :LED 127
outportb 888 :LED]
      buttoncreate "mw "SixOff   "OFF 65  45 15 15 [make "LED bitand :LED 191
outportb 888 :LED]
      buttoncreate "mw "FiveOff  "OFF 65  70 15 15 [make "LED bitand :LED 223
outportb 888 :LED]
      buttoncreate "mw "FourOff  "OFF 65  95 15 15 [make "LED bitand :LED 239
outportb 888 :LED]
      buttoncreate "mw "ThreeOff "OFF 65 120 15 15 [make "LED bitand :LED 247
outportb 888 :LED]
      buttoncreate "mw "TwoOff   "OFF 65 145 15 15 [make "LED bitand :LED 251
outportb 888 :LED]
      buttoncreate "mw "OneOff   "OFF 65 170 15 15 [make "LED bitand :LED 253
outportb 888 :LED]
      buttoncreate "mw "ZeroOff  "OFF 65 195 15 15 [make "LED bitand :LED 254
outportb 888 :LED]

      buttoncreate "mw "Quit "Quit 40 220 40 15 [outportb 888 0 windowdelete
"mw]
end

Make "led 160
```

***Figure 5A* – Logo Program to Illustrate Use of Command Buttons**

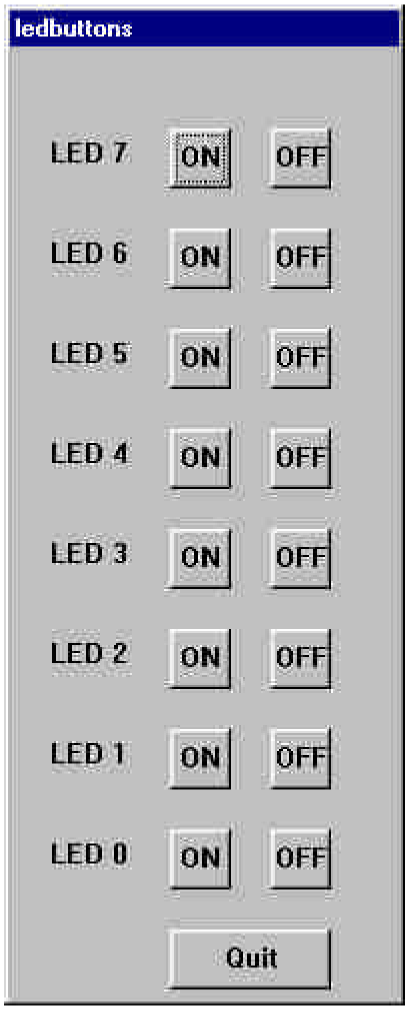


Figure 5B **– Command Button Layout**