

# GENERAL PURPOSE PROGRAMMER/EMULATOR FOR TARGET MICROPROCESSOR BASED SYSTEMS

**Hossam El-Din M. Abdou**, *Student Member IEEE* and  
**Said E. El-Khamy**, *Senior Member IEEE*

Department of Electrical Engineering, Faculty of Engineering, Alexandria University  
Alexandria 21544, Alexandria, Egypt. E-mail:ahossam@alex.eun.eg

## ABSTRACT

A new EPROM programmer and emulator circuit is described In this paper. This circuit is general purpose and has many useful features and applications, among which are; I- It can be used with any 8 bit microprocessor. Ii- It contains RAM and EPROM with variable capacities. Iii- It can be controlled via an IBM compatible PC. Iv- The RAM and EPROM can be used at the same time. V- The output signals can be connected to the external system using various methods (flat cable, D type connector or socket connector). The presented programmer/emulator circuit can be used for debugging microprocessor system with highly reduced costs and time.

## I. INTRODUCTION

With the price of a microprocessor being quoted in pounds rather than tens or hundreds of pounds, it is possible to make the mistake of assuming that this figure represents the major element of cost in any system employing such a device. This is not true since considerable man-hours, together with many additional circuits and equipment are required to make the device work. The finished product therefore costs many times more than the integrated circuit itself. Despite this, the microprocessor-based system is still highly cost

effective, with possible economic applications pervading industry, commerce and home. It has influenced every aspect of modern technological society; from toys to car, sewing machines to factory automation; and even in the design of digital computer itself.

In designing microprocessor based systems [1]-[7], both software and hardware must be interacted with each other. In traditional design, the software ( assembly language program that will control the microprocessor) is designed and then the hardware circuit is built. The linkage between the software program and the hardware circuit is the EPROM on which the program is dumped. It is clear that any error in the program, due to method of dumping or due to bad program code design will not make the target microprocessor system work.

Software debuggers can be used to trace the program but many debuggers are based on the INTEL microprocessors and semantic errors are difficult to be traced. Of course every time the program is dumped into the EPROM, The EPROM must be erased and this costs much time and money.

## II. THE NEW PROPOSED DEBUGGING TOOL

In our proposed system, the prototype of the defined microprocessor is built and then the program is written in a text file using any text

editor after that the program is dumped to the general programmer/emulator which on the other hand is connected to the target microprocessor system.

The program is dumped into the RAM of the programmer and after that the control is transferred from the PC to the target processor. The address decoder must be designed such that the target microprocessor deals with the RAM in the general programmer as the EPROM of the target system. By this way, the EPROM can be emulated by using RAM.

This technique makes it possible and easy to test and debug microprocessor systems by an efficient method. If the assembly language program does not operate well on the target processor system, the programmer is switched to the PC mode and another assembly language program is dumped to the RAM of the general programmer after that the control is given to the target processor. Repeating these steps enables the designer to make a physical and real time debugging and testing of the microprocessor based systems.

In addition, the general programmer can be used as an additional peripheral to the target system for example if the target system has an EPROM only and the application requires another EPROM or an external RAM then the general programmer can supply its EPROM or RAM to support the target processor system.

### **III. SYSTEM BLOCK DIAGRAM**

In this section the main block diagram of the general programmer/emulator and the PC interface block are described.

#### **A. Main Block Diagram**

The main block diagram of the general programmer/emulator is shown in Fig. 1. The following paragraphs describe the blocks of figure 1:

1- The static RAM and the EPROM are shared between the host PC and target processor system.

2- the data bus is 8 bits and the address bus is 16 bits wide.

3- The control unit generates all the signals needed to the RAM or the EPROM such as read signals, write signals, programming pulse for the EPROM, transferring control form the PC mode to the microprocessor mode, etc.

4- There are 6 switches which manually control some of the functions of the general programmer/emulator. A summary of the function of each switch is given below.

**SW1:** reserved

**SW2:** choose EPROM or RAM.

**SW3:** choose 32K static RAM or 8K static RAM.

**SW4:** choose 16K EPROM or 8K EPROM.

**SW5:** EPROM programming mode or EPROM reading mode.

**SW6:** Power ON or power OFF.

#### **B. Parallel Port Block Diagram**

Figure 2 illustrates the block diagram of the parallel port interface. The parallel port of the host PC is connected to three serial to parallel converters so that the 8 bit data and the 16 bit address can be generated. Also it is connected to a parallel to serial converter so that data can be read into the host PC. For simplicity, some of the details are not shown in the block diagram such as the isolation between the input and output.

Figure 3 is a photograph of the general programmer circuit

### **IV. SOFTWARE DESIGN**

The programmer is controlled via a software program written in C++ language. The main functions included in the program are illustrated in the following hypothetical code.

```

Main_function ( )
{
Ask_for_file_name ( );
Read_file_contentts ( );
Extract_address_data_pairs ( );
Enable_computer_side ( );
Write_data_to_programmer ( );
Verify_written_data ( );
Enable_microprocessor_side ( );
Reset_microprocessor();
}

```

This function accepts a valid file name containing the machine code program that will run on the target microprocessor. The file is in text (readable) format and mainly must contain the address-data pairs required to be written to RAM (or EPROM) of the programming circuit. Also the file can contain comments that makes the program more clear. Any comment line must be preceded by any of the following marks, //, /\*, or; marks. For example if the file contains the following three lines:

```

// MOTOROLLA 68xx command
  /* line number 5 */
  0005 4C ; INC A

```

then the code will skip the first line (since it begins with //), the second line (since it begins with /\*) and the last part of the third line (since it begins with; ). The valuable information is the address (0005 ) and the contents (4C). Thus the above segment means that the instruction 4C (increment the accumulator) will be written in the memory location 0005. After extracting the address - data pairs from the file the computer program enables the microprocessor side and the instructions are written in the specified memory locations. The written instructions are then read from the external RAM and compared to the written instructions. After verification the microprocessor side is enabled and the C program send a RESET signal to the microprocessor so that the assembly language program is executed on the target system.

If the machine code program is required to be modified, the control is transferred to the PC side and the above steps of dumping the program and verifying it are repeated. After testing the machine code, the final version of the code can be programmed into an EPROM simply by switching SW2 to the EPROM mode and SW5 to EPROM programming then the same routines of transferring and verifying the data are repeated. After the code is written to the EPROM, it can be put in the target microprocessor system and the target system can work alone (independent of the host PC).

## V. RESULTS AND PROGRAMMING EXAMPLES

The programmer is tested for many 8 bit microprocessors such as 6802, 6502, Z80, 8085 and 8088 (of course other processors such as 6800, 6809 or 8080 can be used). Two text files (LISTING 1 and LISTING 2) are presented as programming examples. The first file contains a machine code program for MOTOROLLA 6802 microprocessor and the second file contains a machine code program for ZILOG Z80 processor. The two programs do the same function that is they count in hexadecimal from 00h to FFh.

We note that there is flexibility in writing the program addresses and data thus for example line 15 in LISTING #2 can be written as 000E f3, 000e F3, 000e f3 or e f3 (the program is not case sensitive).

## VI. CONCLUSION

A general circuit for programming various types of 8 bit microprocessors is proposed. This circuit gives us the flexibility and robustness of designing, debugging, modifying and testing of microprocessor based systems.

Although this circuit is used for programming 8 bit microprocessors, the same

design ideas can be used for programming 16 bit and 32 bit microprocessors. Currently I am working in designing more general programmer for programming and testing 8, 16, 24, 32 and 64 bit microprocessors thus the software for based systems containing processor such 8086 to Pentium can be tested in a robust way. It is obvious that the proposed programming circuit can be used in control systems. Also it can be a valuable tool for implementing fuzzy rule based systems [8].

### REFERENCES

[1] M. Rafiquzzaman, *Microprocessors and Microcomputer-Based System Design*, (Second Edition), CRC Press, 1995.  
 [2] D. V. Hall, *Microprocessors and Interfacing: Programming and Hardware*, McGraw-Hill, 1996.

[3] D. V. Hall, *Microprocessors and Digital Systems*, McGraw-Hill, 1983.  
 [4] B. B. Brey, *The Intel Microprocessors*, Maccmillan, 1994.  
 [5] M. Sargent and R. L. Shomaker, *The IBM PC form the Inside out*, Addison-Wesley, 1986.  
 [6] Y. C. Liu and G. A. Gibson, *Microcomputer Systems: the 8086/8088 Family Architecture, Programming, and Design*, Prentice Hall, 1986.  
 [7] [http://infopad.eecs.berkeley.edu/CIC/archive/cpu\\_history.html](http://infopad.eecs.berkeley.edu/CIC/archive/cpu_history.html). (This site contains information about all microprocessors known to mankind since the birth of computer engineering).  
 [8] H. Surmann and A. P. Ungerling, *Fuzzy Rule Based Systems on General Purpose Processors*, IEEE Micro, Aug. 1995, pp 50 - 48.

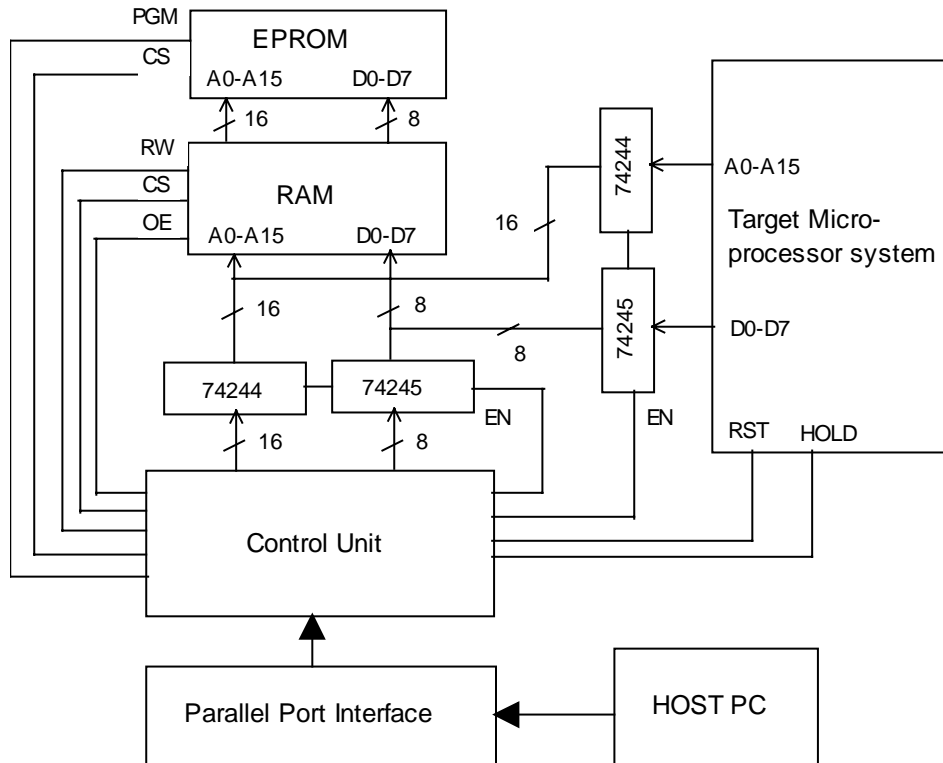


Fig. 1 Main block diagram of the general programmer/emulator

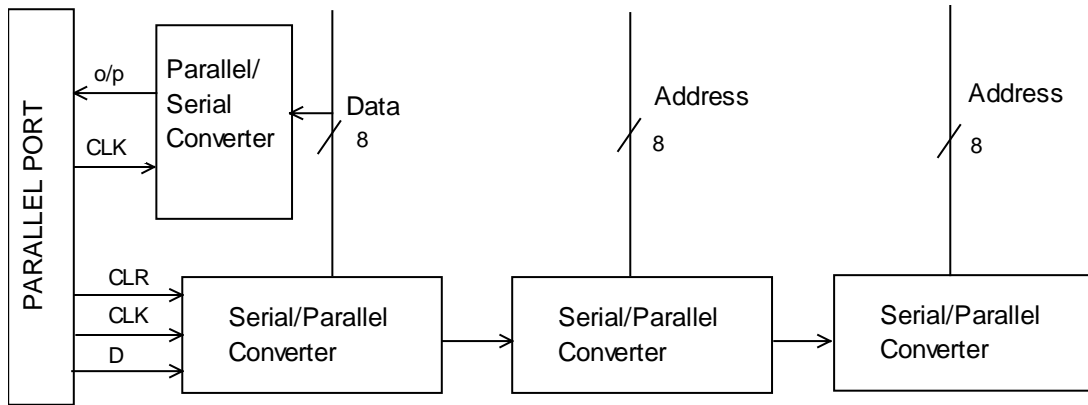


Fig. 2 Block diagram of the parallel port interface

Fig. 3 Photograph of the programmer circuit

**LISTING 1:6802 hexadecimal counter program**

```
// file #1 counter1.hex
```

```

/* Processor MOTOROLA 6802 */
/* this program output the numbers from 00 hex to FF
   hex to */
/* a display whose address is at lines 3 and 4 (8000)
*/
/* the delay between each count is stored in lines 7
and 8 (FFFF)*/
/*****
*****/

/*LINE #*/
0000 86 ;LDA A 00 ;1
0001 00 ;2
//----- load accumulator A immediate with 00
0002 B7 ;STA A 80000 ;3
0003 80 ;4
0004 00 ;5
//----- store contents of A to port 8000 (display)
0005 4C ;INC A ;6
//----- Increment the accumulator A
0006 CE ;LDX FFFF ;7
0007 FF ;8
0008 FF ;9
//----- Load index register with FFFF (for time
//delay)
0009 09 ;DEX ;10
//----- Decrement the index register
000A 26 ;BNE FD ;11
000B FD ;12
//----- If result is not zero then go to 0009
//(relative)
000C 20 ;BRA F4 ;13
000D F4 ;14
//----- branch back to address 0002
/* End of program */
OFFE F0 ;15
OFFF 00 ;16
//----- Restart vector
// program starts from address F000

```

```

// file #2 counter2.hex
/* Processor ZILOG Z80 */
/* this program output the numbers from 00 hex to FF
   hex to */
/* a display whose address is at line 3 (0E)
*/
/* the delay between each count is stored obtained via
registers B and C */
/*****
*****/

/*LINE #*/
0000 3E ;LDA , 00 ;1
0001 00 ;2
//----- Load accumulator A immediate with 00
0002 D3 ;OUT (0E) , A ;3
0003 0E ;4
//----- output the contents of A to port 0E
//(display)
0004 06 ;LD B , 80 ;5
0005 80 ;6
//----- Load B register with 80
0006 0C ;INC C ;7
//----- Increment C register
0007 20 ;JR NZ , 0006 ;8
0008 FD ;9
//----- Jump to location 0006
0009 05 ;DEC B ;10
//----- Decrement the B register
000A 20 ;JR NZ,0006 ;11
000B FA ;12
//----- Jump to location 0006
000C 3C ;INC A ;13
//----- Increment the accumulator
000D 20 ;JR NZ , 0002 ;14
000E F3 ;15
//----- Jump to location 0002
/* End of program */

```

**LISTING 2:Z80 hexadecimal counter program**