Proceedings of 2013 2nd International Conference on Advances in Electrical Engineering (ICAEE 2013)
19-21 December, 2013, Dhaka, Bangladesh

Development of a 16-Bit Microprocessor Learning System using Intel 8086 Architecture

Golam Mostafa
Department of Electrical and Electronic Engineering, Ahsanullah University of Science and Technology (AUST)
141-142 Love Road, Tejgaon Industrial Area, Dhaka-1208, Bangladesh, Email: krdcbdgm@yahoo.com

ABSTRACT
For the period 1971-1975, the industrial design engineers primarily studied the 4004, 8008 and 8080 microprocessors for the purposes of building control circuits and consumer products. The introduction of the superior 8-bit 8085 microprocessor in 1976 drew the attention of the academicians who were charmed at the architecture and functionality of this chip and they shortly included it as a course in the curriculum. As a result, the need for 8085 microprocessor Learning/Training System was a genuine demand and Intel released the SDK-85 Learning Kit without detailed design documentation. In 1978, Intel introduced the 16-bit 8086 microprocessor, which also found its place as a standard course at the educational institutions. Intel released the SDK-86 Learning Kit for 8086 but again without detailed design information. The SDKs helped people learning the 8085/8086 architecture and programming but not knowing the design methodology of a microprocessor learning system. The design techniques remained within the realm of the business companies. This paper presents an account of independently developed and tested design guidelines for an 8086 trainer, which an interested reader may use to build his own 8086 Microprocessor Learning System.

Keywords-MicroTalk-8086 Trainer, Learning System, SDK-86, Monitor Program,

I. Introduction

In the year 1971, Intel introduced the 4004 as the first rudimentary microprocessor to serve dedicated applications in the manufacturing of calculators for the Busicom Company of Japan. As the field of the MPU applications was expanding, so was the acceleration in the development of the new MPU chips.

In 1976, Intel introduced the 8085, which marked the beginning of the first general purpose, fully programmable and single supply microprocessor chip. The 8085 was endowed with considerable complexity level, which convinced the Intel Engineers believing that the chip was to be studied and understood by the design engineers/academicians and accordingly they released the 'System Design Kit, SDK-85' for learning the 8085. The academic institutions shortly adopted the microprocessor in their curriculum.

In 1978, Intel launched the 8086, which was superior to 8085 in many respects but at the expense of simplicity. The 8085 contained only 6,500 transistors, whereas the 8086 contained 29,000 transistors to support features like 16-Bit processing, multiuser platform, 1MByte memory access, 64Kbyte IO ports addressing, built-in debugging logic and parallel processing with 8087/8089 co-processors. Though complex, the 8086 was still to be studied and understood in order to design and develop reliable and robust systems for which an 8086 Microprocessor Learning System was in need and accordingly the Intel released the 'System Design Kit, SDK-86 (Fig. 15)'. The SDK-86 Kit was very expensive with a tag price of U$800/- in 1993. Educational institutions also adopted the 8086 in their syllabuses and the need of 8086 Trainers was in line to carry out the laboratory works.

Besides the SDK-86 Kit, many companies built and released 8086 trainers for educational and training purposes but without detailed design information. Since, these trainers were expensive they remained out-of-reach of the common students and institutions of developing countries particularly of Bangladesh. This scenario has motivated the author to undertake the technical challenging job of the design, development, and prototyping of a low cost 8086 Learning System.

This paper contains the tricks, tools, methodology, procedures, formula and resources for building an 8086 Learning System. An interested reader may follow all these guidelines and can build an 8086 trainer, which he can use for his own learning, purposes.

II. Expected Hardware and Software Functional Features of an 8086 Trainer

A microprocessor learning system is a machine, which cannot discharge the dynamic behavior of a class room teacher. However, a designer must employ all possible efforts to incorporate features into its design so that it stimulates the students to prepare questions in the form of program codes, submit them to the trainer and get the answers. This means that the trainer should be user-friendly, interactive, and well documented.

Before the design begins, the author of this paper, being a designer of the proposed 8086 trainer (let us call it MicroTalk-8086) wishes to declare that the final trainer should have the following hardware and software features.

A. Hardware Functional Features

The following description accounts the hardware resources and their functions of the trainer, which can be located on the Component Layout Diagram of Fig. 1.

- Hexadecimal keypad and its associated controller (U17:8279, U20:LS138): The keypad will enable the user entering various commands and data.
- Seven-segment display unit and its associated controller (DP0-DP15,U17:8279, U18-U19:LS138): The display unit will permit the user to see the *(i)* 20-bit address of a memory location and its 8-bit content, *(ii)* 16-bit content of two consecutive memory locations, *(iv)* 8-bit/16-bit content of CPU registers, and *(vi)* multi-digit result of computation.
- LED-based display unit and its associated controller (LED0-LED15, U16:8255): This unit will show the results of calculation and register in bit form.
- Serial Communication Controller and its associated controller (U15:8251,U2:LS74,U3:LS90,U21:232): These components form the 'Serial Communication Interface' for exchanging code/data with IBMPC.

- Interrupt Priority Controller (U14:8259): The chip handles multi-level maskable interrupts over the INTR and INTA/ pins of the 8086.
- Connectors (J1-J5): J1, J3-J4 and breadboard will enable developing interfacing circuits leading to system design. J2 will connect +5V to the logic board. J5 connector will accommodate serial cable for communication with the IBMPC.

B. Software Functional Features

The following bulleted lists have accommodated the software functional features of the trainer, which essentially form the Monitor Program (a ROM resident Computer Operating System) of the trainer.

- Keyboard Commands (EXA, EXB, EXW, CHG): To open a memory location and check/change its 8-bit contents or 16-bit contents of 2 consecutive memory locations.
- Keyboard Commands (FRW, BKW): To check the content of the next or the previous memory/register location.
- Keyboard Commands (AL, AX, IP): To check the contents of all the 8-bit, 16-bit general purpose and index registers.
- Keyboard Commands (FLR, FB): To check the content of the flag register in hex and bit forms.
- Keyboard Command (DOP): To execute a user program residing in EEPROM/RAM.
- Keyboard Command (PC, S/S): To execute one instruction at a time (known as trace or debugging).
- Keyboard Command (DNL): This is a 'Hot Key' and is dedicated to initialize all the relevant hardware of the 8086 trainer for establishing serial communication with the IBMPC.
- Additional EEPROM-based subroutines to facilitate system design.
- Graphical User Interface (GUI) (Appendix-D): This GUI interface will be installed in the IBMPC and will allow users to download Intel-hex formatted code/data into the RAM space of the trainer.

III. Functional Descriptions of Hardware Components and their Development

A. Components Layout Diagram of the Proposed 8086 Trainer

The hardware components of Fig. 1 form various functional subsystems of the 8086 trainer.

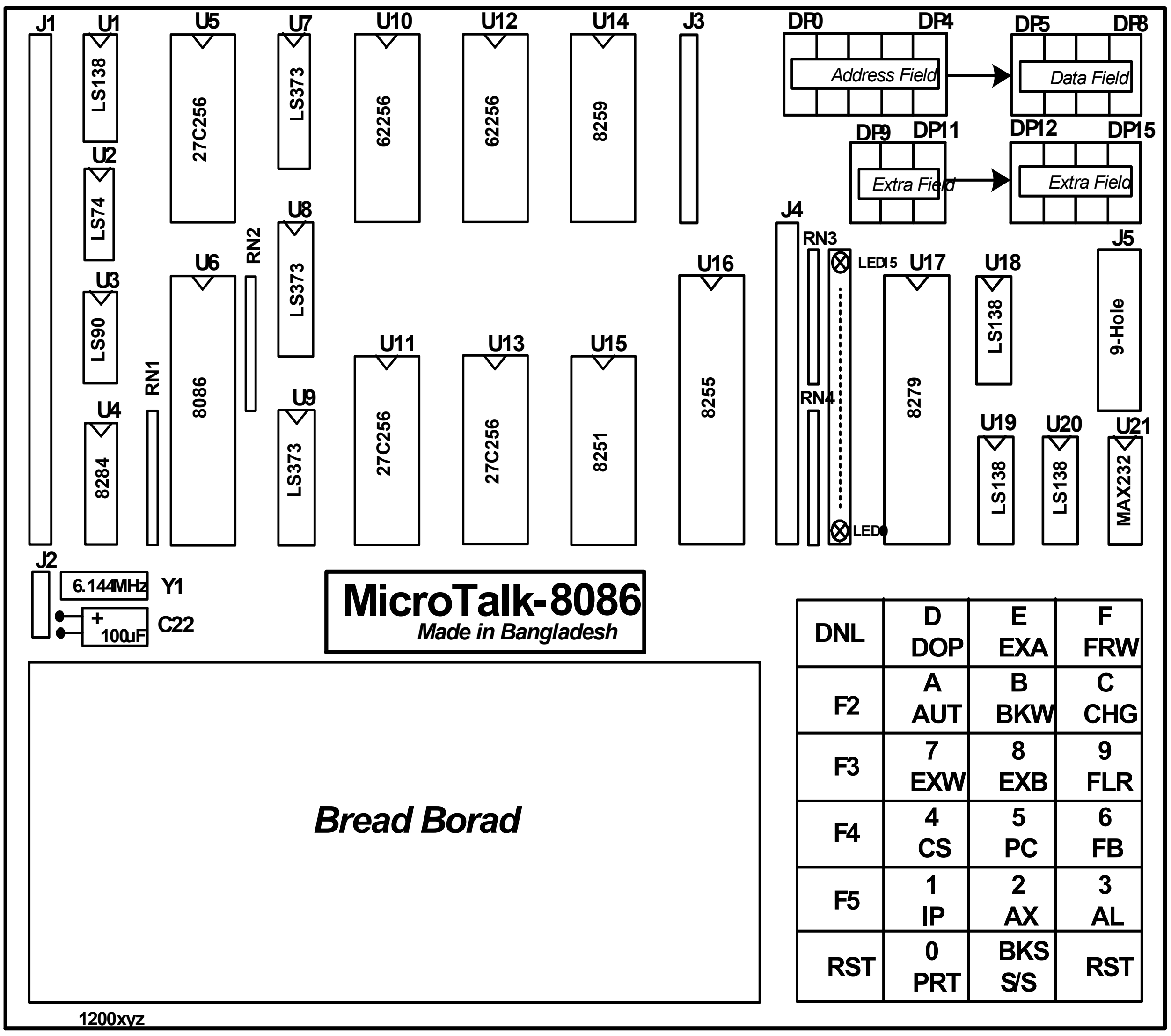


Fig. 1 Componets Layout of MicroTalk-8086 Trainer

B. Organizational Diagram (Hardware Block Diagram) of the Proposed 8086 trainer

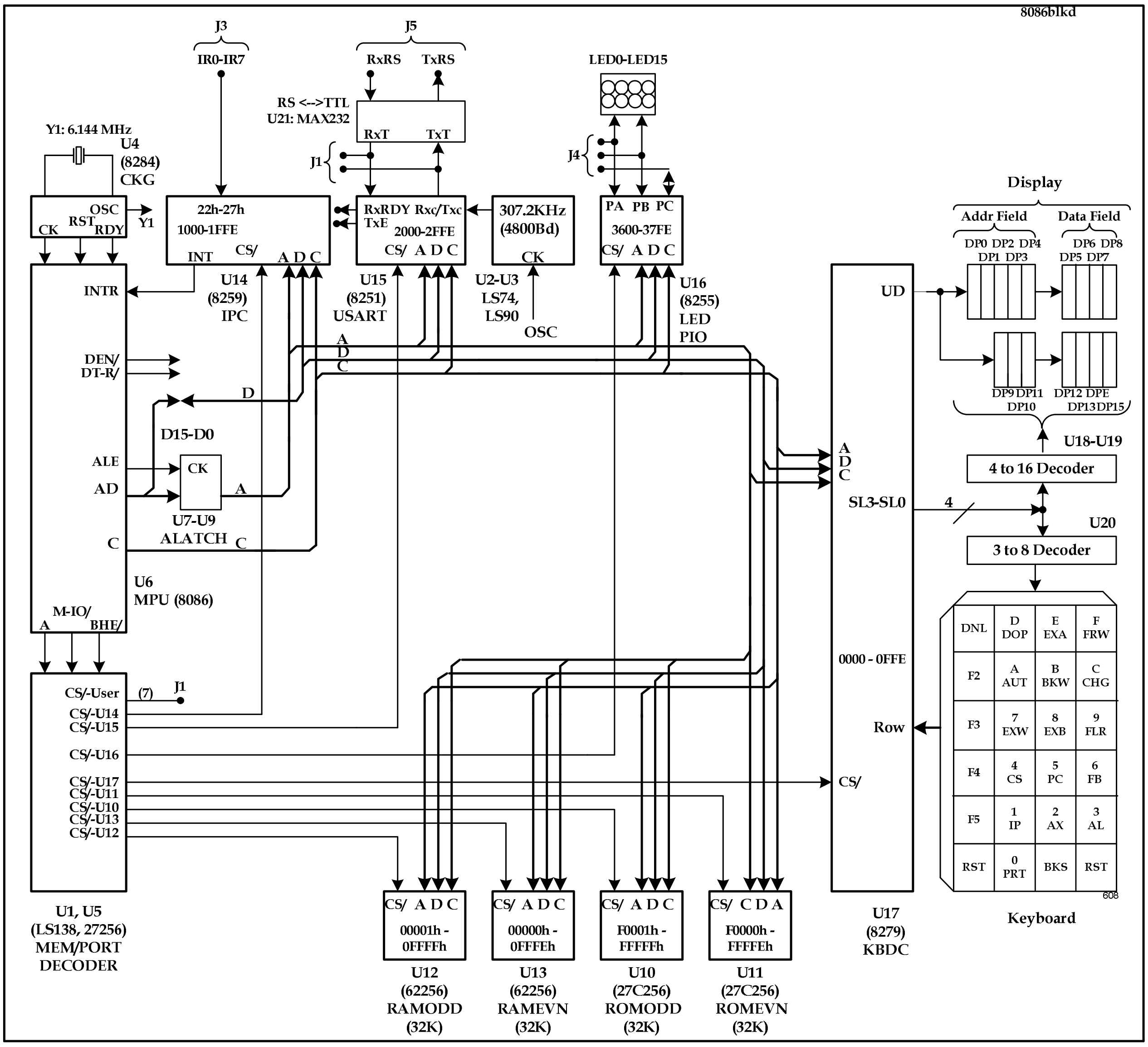


Fig. 2 Organizational (Hardware Block Diagram) Diagram of MicroTalk-8086 Trainer

C. Parts List of the MicroTalk-8086 Trainer (Fig. 1)

Sn	Subsystem	Components			Functions
		Circuit Desig.	Type No	Package	
1	CPU	U6	8086	40-pin DIP	16-Bit Microprocessor
		U7-U9	3x74LS373	20-pin DIP	Auxiliary Data Latch
		U4	8284	18-pin DIP	Clock/Reset Signal Generator
		Y1	Crystal	6.144MHz	External Crystal for 8284
		C22	Polar	100μF/16V	Capacitor for Reset Circuit, Resistor in RN2
		RN1	8x4.7k	9-pin SIP	Pull Down Resistors for few signals of 8086
		RN2	8x4.7k	9-pin DIP	Pull Up Resistors for few signals of 8086
		J1	Female	2x40-pin SIP	Edge Connector to interface user circuit with 8086
2	Memory/Port Decoder	U5	27C256	28-pin DIP	EPROM as Decoder
		U1	74LS138	16-pin DIP	3-to-8 Decoder
3	Memory	U10-U11	2x27C256	28-pin DIP	EPROM as Odd Bank and Even Bank
		U12-U13	2x62256	28-pin DIP	RAM as Odd Bank and Even Bank
4	Keyboard/Display	U17	8279	40-pin DIP	Keyboard/Display Controller
		U18-U19	2x74LS138	16-pin DIP	Decoders for 16-no CC-type 7-Seg Display Devices
		U20	74LS138	16-pin DIP	Decoder for Keypad
		DP0-DP15	CC-type	10-pin DIP	CC-type Type 7-Segment Display Devices
		K00 –K53	Push	6-pin DIP	Spring return type PCB Mountable Switches
5	Serial Communication	U15	8251A	28-pin DIP	Sync/Async Serial Communication Controller
		U2	74LS74	14-pin DIP	D Flip-flop used as divide by two counter
		U3	74LS90	14-pin DIP	Decade Counter
		U21	MAX232	16-pin DIP	RS232 ↔ TTL Converter
		J5	DB-9	9-pin	Female Connector to hold Serial Comm. Cable

6	Interrupt Priority Controller	U14	8259	28-pin DIP	Interrupt Priority Controller
		J3	Female	2x14-pin SIP	Edge Connector for External Interrupt Signals
7	Parallel IO	U16	8255	40-pin DIP	Parallel Input/Output Controller
		RN3, RN4	2x8x1k Res	2x9-pin SIP	Current Limiting Resistors for LED0-LED15
		LED0-LED15	Normal	LED	Light Emitting Diodes to Monitor IO Signals
		J4	Female	2x24-pin DIP	Edge Connector for Hookup Wiring with 8255
8	Other	Bread Board	-	-	To Prototype User Circuits
		J2	Female	-	To Connect External +5V and 0V

D. CPU Subsystem

- U6: 8086 Microprocessor which, we will study.
- U7-U9: Latches
  U8, U9 will demultiplex $A_{15} - A_{00}$ signals from the multiplexed AD_{15}-AD_{00} bus. U7 will demultiplex A_{19}-A_{16} and BHE/ signals from the multiplexed signals A_{19}/S7 – A_{16}/S3 signals.

E. Memory/Port Decoding Subsystem

- U5: 27C256 Based 1st Stage Memory-port Decoder: This is an EEPROM based decoder whose storage locations permanently hold pre-defined data (Fig. 4) in order to uniquely select the on-board RAMs, ROMs and IO controllers (8279, 8259 and 8251) for the specified address boundaries.
- U1: 74LS138 Based 2nd Stage Decoder: This decoder selects the on-board PIO (U15:8255) and seven more externally connected IO controllers.
- Decoder Schematic Diagram

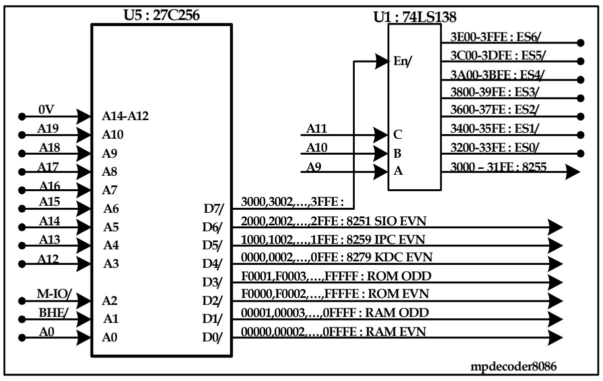
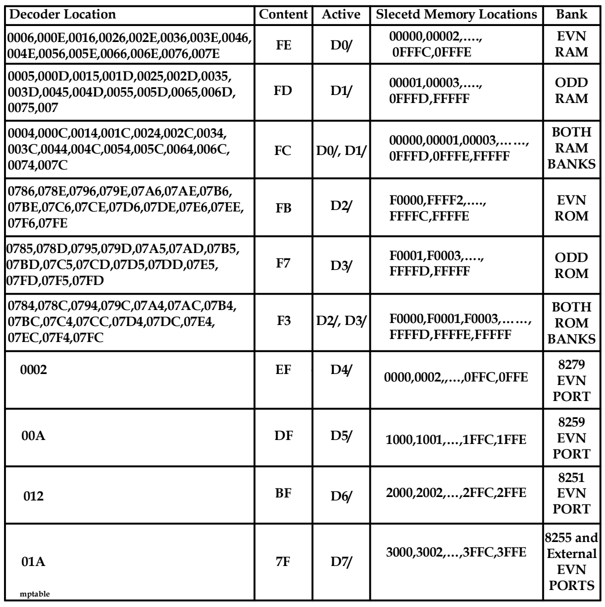


Fig. 3 Schematic Diagram of Memory/Port Decoder

- Truth Table of the 1st Stage ROM Based Decoder



Decoder Location	Content	Active	Slecetd Memory Locations	Bank
0006,000E,0016,0026,002E,0036,003E,0046, 004E,0056,005E,0066,006E,0076,007E	FE	D0/	00000,00002,...., 0FFFC,0FFFE	EVN RAM
0005,000D,0015,001D,0025,002D,0035, 003D,0045,004D,0055,005D,0065,006D, 0075,007	FD	D1/	00001,00003,...., 0FFFD,FFFFF	ODD RAM
0004,000C,0014,001C,0024,002C,0034, 003C,0044,004C,0054,005C,0064,006C, 0074,007C	FC	D0/, D1/	00000,00001,00003,......., 0FFFD,0FFFE,FFFFF	BOTH RAM BANKS
0786,078E,0796,079E,07A6,07AE,07B6, 07BE,07C6,07CE,07D6,07DE,07E6,07EE, 07F6,07FE	FB	D2/	F0000,FFFF2,...., FFFFC,FFFFE	EVN ROM
0785,078D,0795,079D,07A5,07AD,07B5, 07BD,07C5,07CD,07D5,07DD,07E5, 07FD,07F5,07FD	F7	D3/	F0001,F0003,...., FFFFD,FFFFF	ODD ROM
0784,078C,0794,079C,07A4,07AC,07B4, 07BC,07C4,07CC,07D4,07DC,07E4, 07EC,07F4,07FC	F3	D2/, D3/	F0000,F0001,F0003,......., FFFFD,FFFFE,FFFFF	BOTH ROM BANKS
0002	EF	D4/	0000,0002,,...,0FFC,0FFE	8279 EVN PORT
00A	DF	D5/	1000,1001,...,1FFC,1FFE	8259 EVN PORT
012	BF	D6/	2000,2002,...,2FFC,2FFE	8251 EVN PORT
01A	7F	D7/	3000,3002,...,3FFC,3FFE	8255 and External EVN PORTS

mptable

Fig. 4 Truth Table for the 1st Stage Memory/Port Decoder

F. Memory Subsystem

- Schematic Diagram

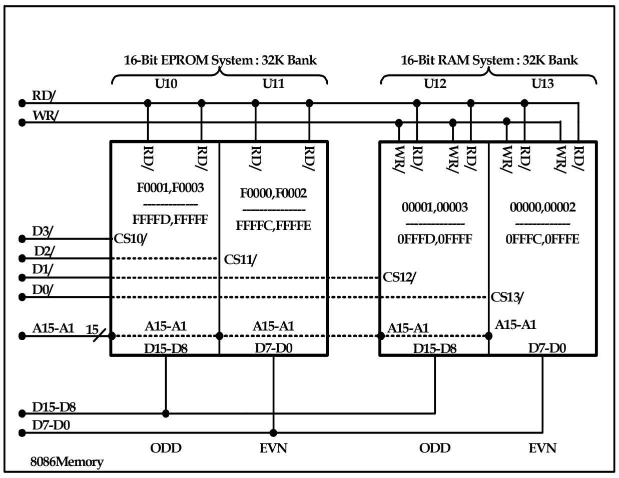


Fig. 5 Schematic Diagram for RAM and EEPROM Subsystem

- Memory Space Organization of MicroTalk-8086

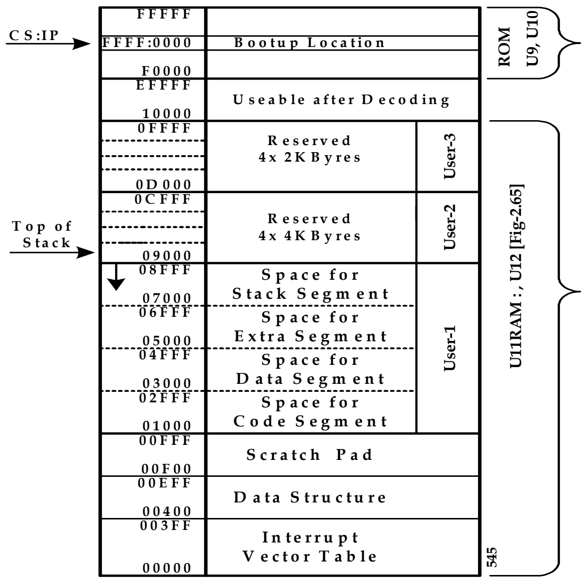


Fig. 6 Memory Space Map of MicroTalk-8086

G. Schematic Diagram of Serial IO Interface

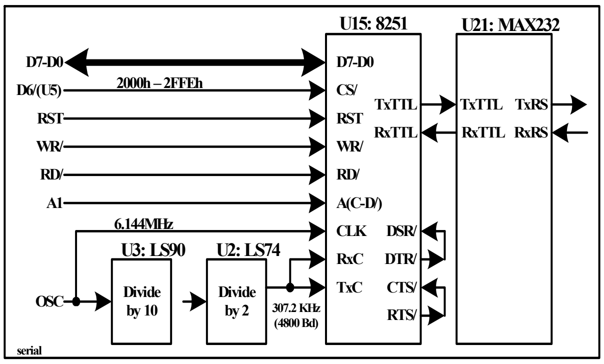


Fig. 7 Schematic Diagram for Serial Communication Subsystem

H. Schematic Diagram of Keyboard/Display Controller

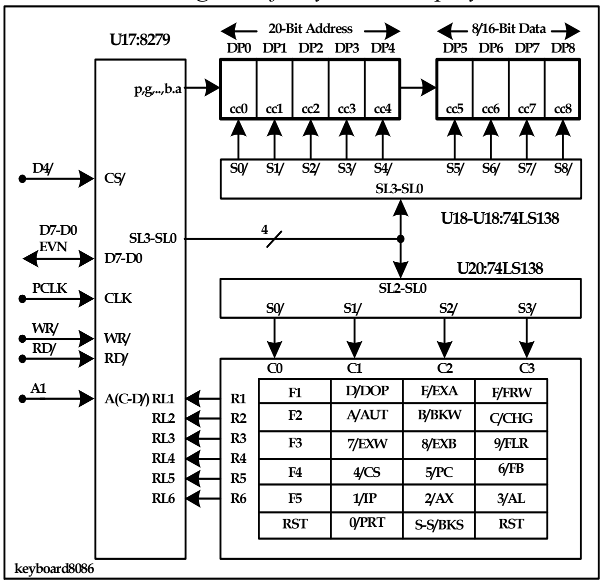


Fig. 8 Schematic Diagram for Keyboard/Display Subsystem

IV. Functional Descriptions of Software Components and their Development

The 8086 trainer must contain a ROM-resident program called ‘Monitor Program (MP)’. This MP allows a user to issue command and enter data from the local hex keypad as well to see data on the display. The MP is a complex program, which mainly consists of the following components. A designer must acquire clear understanding on the functions and interlocking of these software components to translate them into program.

- EXA (Examination) Command Key
  Pressing this key will open the address field. In response, the user will enter 20-bit address of a memory location in hex form. At the end of address entry, the MP displays the message ‘XXXXX Ad’ and then waits to receive EXB or EXW commands for showing the content of one memory location or of two consecutive memory locations.
- EXB (Byte Examination)
  The EXB command tells the MPU to show on the data field the 8-bit content of a memory location whose address is seen on the address field.
- EXW (Word Examination)
  The EXW commands the MPU to show on the data field the 16-bit content of two consecutive memory locations starting from the one whose address is seen on the address field.
- FRW (Forward) Command Key
  Pressing this key will increment the address field. The user will see the 8-bit content of the next memory location or the 16-bit content of the next two consecutive memory locations.
- BKW (Backward) Command Key
  Pressing this key will decrement the address field. The user will see the 8-bit content of the previous memory location or 16-bit content of two previous consecutive memory locations.
- CHG (Change) Command Key
  Pressing this key will open the data field. In response, the user enters new 8-bit/16-bit data for the current memory/register location.
- AUT (Automatic Increment of Address Field)
  This key opens the address field. Once the address entry is done, the data field is automatically opened. After entry of the 8-bit data in the data field, the next memory location is automatically opened.
- DOP (Do/Execute a Program) Command Key
  This command key opens the address field. The user enters the beginning address of the application program. Once the address entry is done, the MPU jumps at the beginning of the user program and executes it.
- PC (User Program Counter) Command Key
  This assists to execute one instruction at a time with the help of the S/S (Single Stepping) Command Key. Pressing PC key will open the address field. The user enters the address of the instruction to be single stepped. Pressing the S/S will execute the instruction pointed by the user PC.
- S/S (Single Stepping) Command Key
  This key tells the MPU to execute one instruction at a time known as debugging or trace. At the end of the execution of current instruction, the next instruction becomes ready for execution. In the mean time, the user may check contents of the registers if they contain the expected values or not. This way we may detect/correct faulty instructions.
- AL (AL-register Check Command Key)
  This command is active in S/S mode. It allows checking and editing the contents of all the 8-bit registers (AL, AH, BL, BH, CL, CH, DL and DH) with the help of the FRW, BKW and CHG keys.
- AX (AX-register Check Command Key)
  This key is active in S/S mode. The AX command allows checking and editing the contents of all the general-purpose 16-bit registers (AX, BX, CX and DX) with the help of FRW, BKW and CHG keys.
- IP (IP-register Check Command Key)
  This key is active in S/S mode. It allows *(i)* checking the contents of the 16-bit pointer registers IP, SI, DI, BP and SP and *(ii)* editing the contents of the pointer registers SI, DI and BP with the help of the FRW, BKW and CHG keys.
- CS (CS-register Command Key)
  This key is active in S/S mode. The CS key allows only checking the contents of the 16-bit segment registers (CS, DS, ES and SS) with the help of the FRW and BKW command keys.
- FLR (Flag Register Command Key)
  This command key is active in S/S mode. The FLR command allows checking the 16-bit content of the flag register in hexadecimal form.
- FB (Flag Bit Checking Command Key)
  This command key is only active in S/S mode. The FB command key enables checking the contents of the active nine flag bits of the flag register of 8086.
- PRT (Port Register Checking Command Key)
  This command is active in S/S mode and it allows checking the contents of the variable ports only.
- BSK (Back Space) Command Key
  Pressing this command key allows the MP to erase an incorrectly entered digit either in the address or data field.

V. Development Strategy for Hardware Circuits and Software Routines

There are plenty of things to design, develop and test in both the hardware and software sides. We have undertaken a highly ambitious job, which aims at developing every bit of hardware and software from zero base line. The hardware and the associated software codes are experimental and they may not respond in a way the designer has wanted. The designer does not know whether it is the piece of hardware or the software or the both that is not working! To overcome this difficulty, he has to begin the design with a simple and elementary circuit (basic startup circuit, Fig. 9) and the associated codes. The designer must employ all possible trials-and-errors efforts to make this startup circuit working based on which we will develop the remaining hardware and software (Section-III, IV) of the trainer following the incremental method.

To realize a complex system of this kind (the 8086-based trainer), we have to follow the 'Small Start Strategy (SSS)' approach, which states that:

- Connect together only those hardware components (Startup Circuit), which are minimally required to 'start up' the 8086. The MPU will announce its successful 'start up' by continuously blinking LED0 connected at PA0-pin of the 8255 of Fig. 9, 10.
- Prepare only that much program code as much we need just to blink LED0 continuously.
- After testing the 'startup' circuit, we can add with it another incremental amount of hardware and the corresponding activating software codes. Continue like this until the complete system emerges.

VI. 3RD Party Instruments Required to Develop Hardware and Software Resources of Trainer.

- IBMPC with WINX Operating System: To develop machine codes for the 8086 assembly instructions and download application codes into RAM space.
- Wire Wrapping Accessories (Appendix-A): These accessories would be required to connect the components by wire wrapping (if need arises).
- 8086 Assembler: To create assembly source file, list file and Intel-Hex file for various software routines, subroutines of the Monitor Program, Interfacing Experiments and 8086-based System Design.
- TOP2005 Programmer: To fuse the machine codes of the 8086 assembly programs into the ODD and EVN banks of the EPROM memories of the trainer.
- USB ↔ RS232 Smart Conversion Cable: To interface the RS232 signal of the serial subsystem of the trainer with those IBMPC, which are equipped with only USB-ports.
- Optionally MicroTalk-8086 or SDK-86 trainer: (Appendix-C):
  The trainer will help testing and debugging of the experimental hardware circuits and software routines under development.

VII. Testing of the Startup Circuit

Let us carry out the following steps to test the Startup Circuit of Fig. 9.

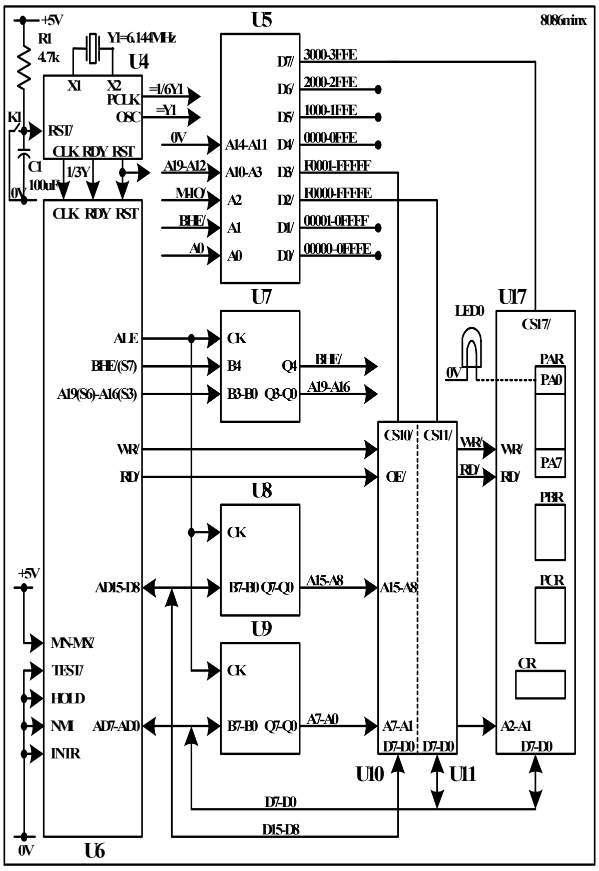


Fig. 9 Block Diagram for the Startup Circuit of MicroTalk-8086

1. Take a piece of paper and draw complete schematic diagram for the Startup Circuit of Fig. 9.
2. Place the ICs, Reset Circuit and LED0 on the breadboard (Fig. 10) as per circuit diagram of Fig. 9. Connect all the components together using short jumper wires. We hope that the startup circuit will work. In contrary, we have to build the startup circuit on gridboard using wire-wrapping (Fig. 13).

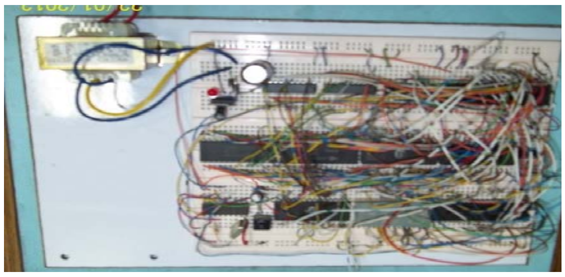


Fig. 10 Breadboard Setup to Test Startup Circuit of Fig. 9

3. Let us convert the following text/pseudo codes into 8086 assembly codes using IBMPC, DOS Screen and MASM assembler. This program blinks LED0 continuously.
   a. At powerup, 8086 begins at location FFFF0h (FFFF:0000, Fig. 6).
      From location FFFF0h, let us make a jump at location F0000h (Fig. 6).
   b. The program execution begins at location F0000h. Since, we do not have RAM in the startup circuit, there is no need to initialize (even no scope) to call subroutines.
      Initialize 8255 (U17) to make its Port-A lines working as output in order to drive LED0.
   c. Ignite the LED0.

Insert Time Delay by program loop. We cannot call a Time Delay Subroutine due to lack of Stack Space in our startup circuit.

d. Extinguish the LED0.
   Insert Time Delay by program loop.
e. Goto Step-c to blink the LED0 again.

4. Convert the Text Codes of Step-3 into 8086 assembly Codes as follows:

```
LA:     FFFF0     -  jmp     F0000h
LB:     ;----- inialize 8255 PIO Controller-------------------
        F0000     -  mov     dx, 3206h    ; points CR
                     mov     al, 80h
                     out     dx, al
LC:     ;----- ignite LED0 -------------------------------------
                  -  mov     dx, 3000h    ; points PAR
                     mov     al, 01h
                     out     dx, al
        ;----- insert Time Delay by Program Loop----------
                  -  mov     cx, 0FFFFh
        HERE1:       loop    HERE1
LD: ;---- extinguish LED0---------------------------------------
                  -  mov     dx, 3000h
                     mov     al, 00h
                     out     dx, al
        ;---- insert Time Dlay by Program Loop------------
                  -  mov     cx, 0FFFFh
        HERE2:       loop    HERE2
LE: ;---- blink LED0 again--------------------------------------
                  -  jmp     LC
```

5. Let us use IBMPC, DOS Screen and MASM assembler to find Binary Codes as follows for the Assembly Codes of Step-4.

```
LA:    FFFF0   -  EA 00 00 00 F0  : jmp    F0000h
       ;---------------------------------------------------------------
LB:    F0000   -  BA 06 30        :  mov   dx, 3006h
                  B0 80              mov   al, 80h
                  EE                 out   dx, al
LC:    F0006   -  BA  00 30       : mov    dx, 3000h
                  B0  01            mov    al, 00h
                  EE                out    dx, al
LC1:   F000C   -  B9 FF FF        : mov    cx, 0FFFFh
LC2:   F000F      E2 FE             loop   LC2
LD:    F0011   -  BA 00 30        : mov    dx, 3000h
                  B0 00             mov    al, 00h
                  EE                out    dx, al
LD1:   F0017   -  B9 FF FF        : mov    cx, 0FFFFh
LD2:   F001A      E2 FE             loop   LD2
LE:    F001C   -  EA 06 00 00 F0  :  jmp   F0006h (LC)
```

The readers might have noticed the hand coding of the '*jmp F0000h*' instruction at label LA. This is a system level privileged instruction and only the trusted people can use it. This instruction greatly facilitates easy writing and testing of program codes at the development phase of a system. At the absence of this instruction, we must use the instruction '*jmp DWORD PTR [bx]*', which requires that the target location (F0000 = F000:0000) must be present in a memory-based table, creation of which would appear as a cumbersome job at this preliminary stage of system design. Therefore, inclusion of the *jmp direct_address* instruction is a novel idea of the Intel engineers!

6. Now, from the binary codes of Step-5, let us pick up the underlined code bytes and record them as follows. These codes will go into the Even Bank of EEP ROM (U11, Fig. 9).

```
FFFF0 -    EA
FFFF2 -    00
FFFF4 -    F0
;-------------------------------------------------------------
F0000 -    BA 30
F0004 -    80
F0006 -    BA 30 01 B9 FF FE 00 B0 EE
F0018 -    FF E2 EA 00 F0
;-------------------------------------------------------------------
```

- Let us fuse the binary codes of Step-6 into EEPROM of Even Bank (U11, Fig. 9).
  The Procedures:
  a. Let us find the address of the location of U11 into which we will fuse the code byte EA. The location address is: 7FF8h. How?

  The system address of the location into which EA will be stored is: FFFF0h, where:
  FFFF0 = 1111 1111 1111 1111 0000

  The CPU uses the address lines A_{15}-A_1 (Fig. 5) to access the locations of U11.

  Therefore, the A_{15}-A_1 (1111 1111 1111 000 = 111 1111 1111 1000 = 7FF8h) bits of FFFF0h is the location of U11 that must hold EA. The next location 7FF9h will hold code byte 00h and so on.

  Similarly A_{15}-A_1 (0000 0000 0000 000 = 000 0000 0000 0000 = 0000h) bits of F000h is the location of U11 that must hold code byte BA.

  Now we have the following code mapping for U11:
  7FF8 – EA 00 F0
  0000 – BA 30 80 BA 30 01 B9 FF FE 00 B0 EE
  000C – FF E2 EA 00 F0

  b. Take out U10 and U11 from the breadboard and erase them using Ultra Violet Eraser.

  c. Let us read contents of U11 by a ROM Programmer. The screen should show FFs.

  d. Edit the relevant locations of the screen with the code bytes of Step-6. After the edition, let us save the file as 86EVN.bin.

  e. Now, let us program the 86EVN.bin file into U11 and then place it on the breadboard.

7. Now, from the binary codes of Step-5, let us pick up the non-underlined code bytes and record them as follows. These codes will go into the Odd Bank of EEPROM (U10, Fig. 9).

```
FFFF0 -    00
FFFF2 -    00
;-------------------------------------------------------------
F0000 -    06 B0
F0004 -    EE
F0006 -    00 B0 EE FF E2 BA 30 00
F0016 -    B9 FF FE 06 00
;-----------------------------------------------------------------
```

- Let us fuse the binary codes of Step-7 into EEPROM of Odd Bank (U10). The procedures are same as for the Even Bank Programming. After programming, let us place U10 on the breadboard. The code mapping of U10 is:

```
7FF8 –  00 00
0000 –  06 B0 EE 00 B0 EE FF E2 BA 30 00 B9
000C –  FF FE 06 00
```

8. Apply +5V to circuit of the breadboard. Activate the reset switch. The LED0 should start blinking.

VIII. Development of Monitor Program

What is a Monitor Program (MP)?

After powering up the 8086 trainer, we see the message '8086 CPU' on the display window. Now, we press the E/EXA key, the message _ _ _ _ _Ad appears on the display. How does the CPU know the meaning of the symbol 'E/EXA'? How does it distinguish between 'E for Digit' and 'EXA for Memory Examination

Command'? The answer to this question will lead us to define the meaning of a monitor program, its essential features and the design aspects. This short conference paper cannot accommodate all the details of the design of a Monitor Program. The readers are referred to [4] for comprehensive description of the design and listing of the MP of the 8086 trainer.

Looking at the trainer board, we find that there are two EPROM chips. If we replace these two chips by another two EPROMs just bought from shop, the events cited above will never occur. This indicates that the original two EPROMs contain 'Something' which has guided the CPU to output binary data into the display buffer of 8279 (U17, Fig. 8) corresponding to the message '_ _ _ _ _ Ad'. This 'Something (called MP)'is a collection of complex data structure consisting of many routines and subroutines that interpret the meaning of the command EXA and takes action accordingly. The Flow Chart of Fig. 11 depicts the minimum components of the 8086 MP. We will be able to add with it other routines like 'Single Stepping', 'Auto Forward', 'Back Space', Register Check and etc.

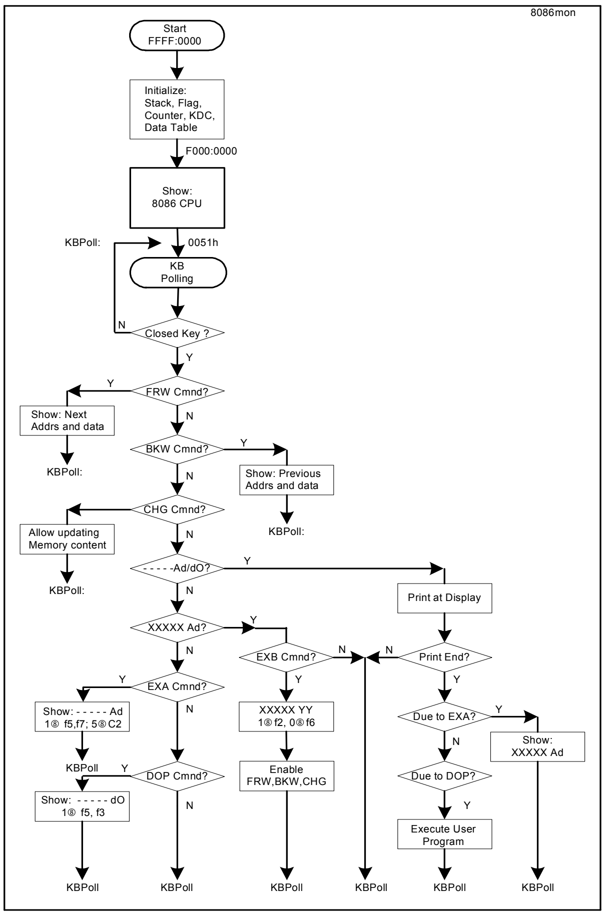


Fig. 11 Basic Monitor Program of the 8086 Trainer

IX. Conclusion

8086 Microprocessor Learning Systems are highly demanded because of its widespread use in the educational institutions and industries. It is also the core of all the Intel advanced architectures. Unfortunately, the design methodology of the trainer is not available in the literature. The author has aimed not to build a better trainer but to disseminate the design tricks by building an 8086 trainer himself (Fig. 13, 14, 16, and 17).

References

[1] Intel Corporation, *Microprocessors,* Literature Sales, P.O. Box 7441, Mt. Prospect, IL60056-7641, USA, 1990.

[2] D. V. Hall, *Microprocessors and Interfacing,* McGraw-Hill Book Company, Singapore, 2nd Printing, 1987.

[3] G. Mostafa, *8086 Microprocessor Interfacing and System Design,* Karighar R&D Center, 2009.

[4] G. Mostafa, *Development of a Low Cost 16-Bit Microprocessor Trainer,* M.Sc. (Computer) Thesis, Bangladesh University of Engineering and Technology, 1998. krdcbdgm@yahoo.com

Appendix-A: Wire-wrapping Accessories

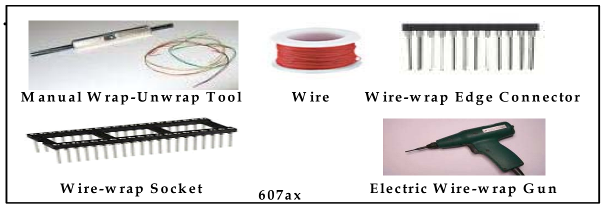


Fig. 12 Wire-wrap Accessories

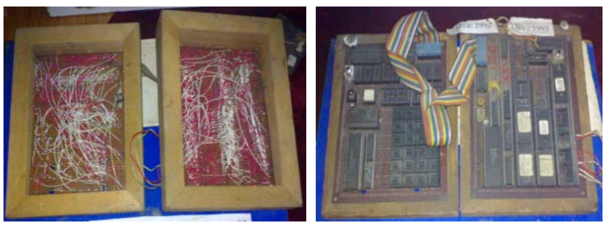


Fig. 13 Wire-wrap and Component Sides of MicroTalk-8086

Appendix-B: Inexpensive manual PTHed 8086 Trainer

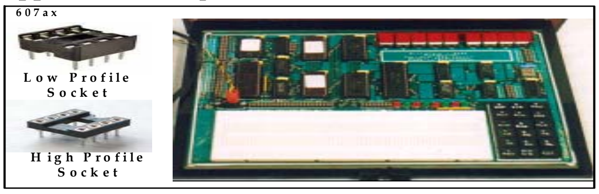


Fig. 14 MicroTalk-8086 on Manual PTHed Double Layered PCB

Appendix-C: Third Party's 8086 Trainers

Fig. 15 Intel's SDK-86 Trainer [1]

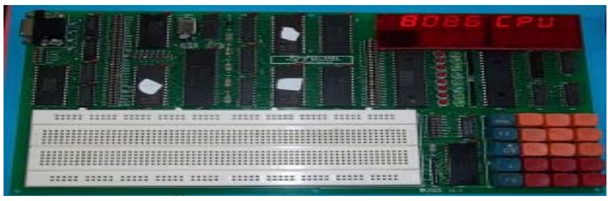
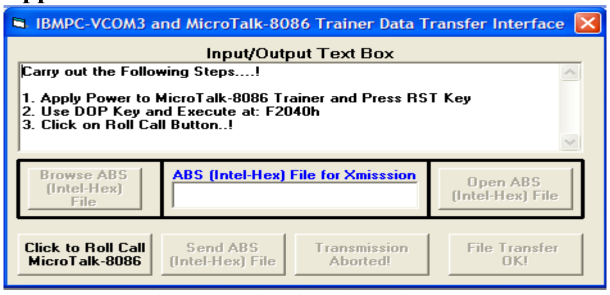


Fig. 16 MicroTalk-8086 Trainer on auto PTH-ed PCB

Appendix-D: GUI Interface for Trainer and IBMPC


IBMPC-VCOM3 and MicroTalk-8086 Trainer Data Transfer Interface

Input/Output Text Box

Carry out the Following Steps....!

1. Apply Power to MicroTalk-8086 Trainer and Press RST Key
2. Use DOP Key and Execute at: F2040h
3. Click on Roll Call Button..!

Browse ABS (Intel-Hex) File
ABS (Intel-Hex) File for Xmisssion
Open ABS (Intel-Hex) File
Click to Roll Call MicroTalk-8086
Send ABS (Intel-Hex) File
Transmission Aborted!
File Transfer OK!

Fig. 17 Graphical User Interface for MicroTalk-8086 [3] and IBMPC