DESIGN, DEVELOPMENT AND TROUBLE-SHOOTING OF MICROPROCESSOR BASED SYSTEMS USING A MINIMAL TOOLS ENVIRONMENT

Khawaha M. Yahya * and Tariq M. Jadoon*

ABSTRACT

This write up attempts to study the various aspects of the design, development and trouble-shooting of microprocessor based systems using a "minimal tool" environment. An 8085 based single board computer was designed for this purpose.

INTRODUCTION

The best way to perform the study was to go through the entire design, development and trouble-shooting phases of a microprocessor based system and at each phase identify the typical problems and propose methods for their elimination. A microprocessor based system for a particular application or a general purpose computer system which could be modified to implement diverse functions. It could enter, execute and debug user programs, provide control of various output devices, or be used as a developmental tool for designing processor based systems. Of the two, the control system application has to be tailored to a particular control problem and thus every specific application is unique. On the other hand a computer system has to execute user programs, which are of a diverse nature, and thus is general. In performing such a study, it is far more desirable to work on a computer system then a control application as the former encompasses all the spheres of a microprocessor based system. While performing the study, a prototype single board microcomputer was designed, developed and debugged so that we could have an insight to the various aspects of the developmental phases.

OBJECTIVE

There are whole sets of very good reasons for constructing your own computer system. Buying an already assembled unit and setting it on the bench or assembling a kit does not give you an insight about computers because all the thinking about how things work has already been done for you. Designing a microprocessor based system requires expensive tools like development system with emulator pods etc. that are not available at our schools and beyond the pocket of individuals.

This write up attempts to identify the various aspects of the design, building and troubleshooting a microprocessor based system using inexpensive tools.

THE BASIC APPROACH

A system as large as a computer requires many decisions. The most immediate and important decision being the general approach to be followed. A major goal in all designs is simplicity of construction and test.

The system should be flexible. It should grow, to a general purpose computer capable of calculations and data processing. In addition, it should be capable of such functions as control of external devices and should have the flexibility for future expansions. May the expansions be in devices to be controlled or in the data handling capability of the system. Processor speed is of prime importance in number crunching applications. Higher the processor speed, higher the cost of the system. An inexpensive system has to sacrifice processor speed.

The boil out is simplicity. Keeping the functions separate simplifies debugging. Using components, which are readily available and are general purpose

* Department of Electrical and Electronic Engineering, NWFP, University of Engineering and Technology, Peshawar
rather than dedicated ones reduces the cost. Using TTL is advisable.

SELECTING THE MICROPROCESSOR

It is very difficult to show that one processor is really better than another but this superiority is only in that application. Generally processors with a large instruction repertoire, require more support chips and moreover a very small number of instructions carry out most programming.

The final choice of 8085 microprocessor is based on our debugging & software environment, the application and the availability of Intel's SDK-85, a minimum kit which includes the chip, some memory, basic input/output, and the necessary control elements on a single board.

THE HARDWARE/SOFTWARE TRADE OFF

The system is built around the microprocessor. The memory comprising RAM & ROM is connected to the 8085 system bus. I/O devices, LED display and Keyboard are connected to the I/O bus, implemented by using the Programmable Input Output chip.

Microprocessor interfacing - connecting a variety of peripherals to the processor is no longer the traditional art requiring the design and implementation of elaborate control electronics. Modern interfacing techniques may be implemented either by hardware or by software or a combination of both. It is always up to the designer to strike a reasonable compromise between the efficiency of the hardware and the lower component count of software implementation. The following hardware-software trade-offs have been made in our design.

a. In our system we have employed a delay subroutine (software) DBOUNCE to debounce the keyboard keys, instead of using NAND gates (hardware), as the number of keys is large.

b. We have used a multiplexed LED display that has to be refreshed (software). This might be a disadvantage as the processor time is utilized but it is saving on the latches and data lines (hardware) that would be used if the display were not multiplexed. In addition, the problem of shifting digits left on pressing a key on the keyboard would be a perpetual problem as one port would have to be reserved for one LED if we were not multiplexing the output.

c. For interfacing the keyboard we have used an 8255, a general purpose Programmable Peripheral Interface (PPI), compatible with most microprocessors. In comparison, special purpose, but complex controllers like the 8279 could have been used for the keyboard and display but we kept the design simple. A non-encoded keyboard is used, continuously scanned by the monitor program for a key closure. The monitor program keeps the microprocessor continuously busy. This polling technique is a trade off for a more complex algorithm required for a coded keyboard.

HARDWARE DESIGN

The project is built around the 8085 Microprocessor; the 8085 multiplexes its data bus with the lower order 8 bits of the address bus. The bus structure dictates the need for an external bus latch 74LS373. The buffer IC's 74LS244 & 74LS245 drive the system buses. The two decoders 74LS138 provide enable and control (MEMR~, MEMW~, IOR~ IOW~) for the memory chips 2716 (2-K ROM) & 6116 (2-K RAM), programmable peripheral interface 8255 and the multipurpose device 8155 Timer.

The non-encoded 24-Keyboard is arranged in a matrix format (4x6), and is interfaced to the microprocessor unit through the 8255. The port Po (PoP) of the 8255 is an output port for the rows and port Pc (PoP) is an input port for the columns.

The six 7-Segment LED displays are interfaced with the microprocessor unit using the 8255. They form a multiplexed display system in common cathode configuration. The arrangement uses two ports. Port Pa (PoP) provides the data to be displayed buffered through a 74LS244 while Port Pb (PoP) is used to select a particular 7-Segment display. Port B is buffered through a 75492.
The system requires a +5V power supply capable of sourcing 2A. The microprocessor unit & memory components require about 500 mA, and the remaining current is supplied to the 7-segment display.

MEMORY DESIGN

Each memory I/O chip is enabled by a signal from the 74LS1398 decoder. The microprocessor always jumps to address 0000H on a system reset. The lowest memory is reserved for the ROM monitor program.

The next 6k memory space has been left unoccupied for future expansion. The 2K RAM is connected at address 2000H. The 256 bytes of scratchpad memory at address 3000H occupies 256B space (3000H-30FFH). The decoding is such that (3100H-37FFFH) is foldback space, and can not be used for expansion.

SOFTWARE DESIGN

The ROM monitor was developed concurrently with the hardware. The monitor is modular and highly structured. It can be broadly divided into four groups.

Group 1:- Start up and initialization routines
Group 2:- Command routines
Group 3:- Utilities
Group 4:- Monitor Tables

The start up and initialization routines comprise of jump addresses for the various interrupts, code to initialize the peripherals, a routine to save the registers and status, a program to send the start up prompt to the LED display and a command recognizer routine which awaits a command from the keyboard and jumps to service it.

The command routines allow the user to
1. Examine and modify registers
2. Execute the user program
3. Single step through any routine
4. Substitute memory

The command routines frequently call one of around twenty utility routines which perform diverse functions e.g. clear the display, display an error message, output characters to the display, read the keyboard and so on.

Key closures are identified by grounding all the keyboard rows and then checking for grounded columns i.e. column-to-row switch closure. The program debounces the key closure by waiting long enough for a clean contact to be made.

The particular key closed is identified by grounding single rows and observing whether a closure is found. The key identifications are in a table TKBD in memory.

The output is a multiplexed LED display. A REFRESH utility, refreshes each display by sending characters from the output buffer to the LED's many times a second so that the LED display appears to be continues.

Single stepping though the program is accomplished by Trapping the processor after each program instruction; saving the status and then jumping back to the monitor so that command routines might be serviced. The trapping is performed by the TIMER OUT of the 8155.

BUILDING THE BOARD

There are a variety of commercially available experiment boards for developing circuits. Breadboards require no soldering and allow ease of interconnecting pins by simply plugging wires in holes. Protoboards, On the other hand, give a more robust outlook and all problems of a faulty contact are eliminated. We have used Breadboards as plugging in a wire is much simpler and far more flexible than soldering a joint.

The development process is greatly simplified by taking small steps at a time instead of hasty leaps. Our first step is to interface the development kit with a Breadboard. The SDK-85 has all the system buses available for expansion. The idea is to use the processor of the kit in our system so that we have access to its internal registers, can execute one instruction at a
time, substitute memory, output signals on the buses and write and test small routines using the kit's 256 bytes of memory.

The kit's expansion buffers are enabled from address 8000h. This is where we start building. A system bus is implemented on the board by connecting wires in parallel constituting the address bus, the data bus and required signals of the control bus. The next step is to fix a decoder on the to board get the CS- (chip select) signals. The memory map of our development system is evaluated to a base address of 8000h. The individual allotment of memory space is retained. Thus for this the A15 of the address bus is connected to the active enable GI (pin 6) of the 74LS138 so that it is activated above address 8000h. The other address lines are connected to the decoder select lines according to the requirement. The interface is then tested by a continuous loop writing to a port. An oscilloscope is used to check whether the select pin is activated by the loop. We now have a decoder and a system bus on our board.

From here on we interface one device at a time and test whether the interface is successful by writing short routines to output or input test data. The next chip we interfaced is PIO, the 8255 PPI. It is tested by sending the control word to its control word register and then sending a byte to be output on one of its output ports. As the output ports are latched they can be tested with a multimeter. Once the PIO is successfully interfaced we proceed to interface the keyboard. The keys are soldered onto a board and thoroughly tested whether pressing a key shortens the row column matrix and do we get a ground at the columns by grounding a row upon depressing a key. The keyboard rows and columns are then connected to the PPI ports. The code for testing a key release or closure is entered into the RAM of the kit modifying the port addresses accordingly. A routine found handy for displaying results on the kit's display is UPDDT (036Eh) which on calling, outputs Register A to the data field. In due course of time we thus successfully interfaced the keyboard.

The number of routine increase, necessitating the use of RAM to store them. This requires the control signals MEMR and MEMW. So use the 74LS138 decoder to decode the RD-, WR- and IO/M- to get the MEMR & MEMW control signals and then interface the 6116 RAM to the system buses. The interface is tested by trying to substitute memory in the kit at the RAM locations. The RAM is very sensitive to voltage fluctuations and it is essential that the ground is common to the kit.

As the code length of our routines for interfacing increases it becomes necessary for some method of storage to exist so that we do not have to enter the complete code every time the kit is powered up. Battery back up seems to be the solution. Four penlight batteries are enough to provide back up for a day or two but the dynamic RAMS tend to be leaky and a slight aging of the battery causes data bytes to fault at random. Moreover, the battery has to be completely isolated from the system otherwise it will try to drive it and drain instantly. While in isolation the kit cannot access the RAM unless we have a common ground and as the ground is not very well distributed on a breadboard leakage is far rapid than expected.

By far the best backup available is an EPROM. The idea is to write down as many routines as possible with lots of NOP's (No operation) between instructions and routines and burn it in on an EPROM programmer. While testing simply copy the routines required into RAM. RAM gives us flexibility, note down all changes required and when these become substantial burn in a new EPROM.

THE MULTIPLEXED LED display is interfaced to the PPI in a manner similar to that of the keyboard.

Finally another board with the processor, bus buffers and latches is built and interfaced concurrently with the kit's system bus. Both are used alternately until a time comes when the kit can be completely isolated from the breadboards and we have a working computer system.

POWER SUPPLY

Power supply, the heart of any system, is often the most overlooked element of a system. If the
Power supply is not properly specified, the system does not work reliably.

Power supply performance is measured by the following parameters:
- voltage current ratings
- regulation
- efficiency

Power-supply engineering is a much harder skill to learn than digital circuit design or Programming. To know the voltages the circuit will require, involves looking up the data sheets. But how much current will the whole system draw? Again, the specification sheets give the minimum, typical and maximum current drain or power dissipation. All the typical figures are simply added to get a rough estimate. The current rating of the power supply should be twice the typical system load current.

Power supplies are not perfect and cannot deliver exactly 5.00 volts under all load conditions. If the load varies, one would like to know how much the voltage will change. If the load is constant and the input line voltage from the wall socket varies, the output in variation over temperature and time is the load regulation tolerance. In addition to tolerance, the overshoot and stability measurements are extremely important.

In driving standard TTL, the circuit, will not tolerate an overshoot of greater than 8 volts. Most commercial power supplies have little or no overshoots. In contrast most home-built power supplies tend to be unstable. This is due to the layout and construction techniques.

The question always asked is "should I build or buy?" Buying is expensive but is the result of hundreds of thousands of man-hours in power-supply engineering. If one builds the supply, the transformer, diodes and capacitors must be all chosen according to certain design rules and formulas. The regulator is matched to the transformer diode, capacitor combination so that stability and efficiency problems do not occur. For any thing less than 3A at 5V simple monolithic voltage regulators such as the popular LM309 is available. One problem typical to breadboards is the distributed supply voltage. This can be minimized by supplying at a central point.

**TESTING**

What do you do when it does not work? What went wrong and why? The debugging process, also known as testing or troubleshooting is an integral part of any system design. Murphy the high priest of blunder said it all:

"In any field of scientific endeavor, if something can go wrong it will, if it can't go wrong it definitely will. And when you think the worst is over, it is not."

When faced with a misbehaving system, there are a number of techniques available to the designer for identifying and correcting problems.

The tools necessary to identify and locate these problems are a multimeter, logic probe, logic analyzer and signature analyzer, oscilloscope, emulator, simulator and in-circuit emulator.

Most of the tools cost several hundreds of thousands of rupees and are not available or accessible to most designers. EPROM programmers may be rented but in-circuit emulator can not be. If we cut on any of the above tools the testing time increases manifold. So what can be done? The best thing is to make a compromise between the equipment and testing time. Following a logical technique usually simplifies troubleshooting. The most commonly encountered errors in a system are:

1. A wiring fault - a short or open circuit
2. Component failure - including wrong value components
3. Software bugs
4. Noise or interference - either internal or external.

- Wiring faults are detected by a resistance check or "buzz test". Check each wire and make sure it goes to the right pin and no other. Do not be confident that the schematic is without fault until the system works.

- Components such as resistors, capacitors, inductors, transformers, transistors, diodes, integrated circuits and connectors may all experience failures. Resistors crack open. Capacitors leak out their electrolyte "nothing is perfect."
Software can be at fault. If you have never tested a routine you can’t say whether it is perfect. Software problems or “bugs” are often the hardest to identify. The system may work for months and then mysteriously halt due to some unanticipated flag condition.

Noise is everywhere. Any length of wire becomes an antenna.

1. When ICs switch they generate small current changes in their power requirements. If too many circuits switch at once, the power supply voltage may change enough to affect other parts of the circuit. We thus usually have bypass capacitors across each IC.

2. The power supply should be properly designed with noise filters. If a glitch happens at a crucial moment, data are lost and the machine fails.

3. Simple problems usually prevent the system from working at all. Intermittent failures are most often due to connector or bad solder joint problems. These should be checked first, before assuming anything else is at fault.

**DESIGN PROBLEMS**

You thought you knew what you wanted - but you didn’t. Yes, we all make mistakes, so we might as well admit it. Design errors are generally due to:-

- **IMPROPER USE:** Passing too much current through a resistor will cause it to burn up. Applying too much voltage to a capacitor will cause it to short. The "too much" problem is the most common. For example too many loads on a single output line may cause errors depending on temperature variation.

- **IMPROPER SPECIFICATION:** If we believe that a component has a fan-out of 20 while it can only drive 10 is improper specification. It simply was not noticed in the data specification sheet. It is absolutely necessary to thoroughly go through the data sheet before using a particular IC. Look at the specification sheets even if you have a circuit described in a book, may be the author did not test it.

**CONCLUSIONS**

- After going through all the development stages we ended up with a prototype single board computer on a breadboard. Almost all the hardware testing and troubleshooting was done using nothing more than a VOM (volt ohm meter). The software was tested and debugged using Intel's SDK-85. Conceptually this provide something like an in circuit emulator. The difference being that we have not emulated the microprocessor. Instead we have substituted the prototype system's microprocessor with that of the SDK's 8085. This provides almost complete control of the microprocessor as we can examine and modify register, setup software breakpoints and single step through our routines. All these options are almost indispensable debugging techniques. The only hang up compared to an in circuit emulator is that everything has to be done in Hex code and we miss symbolic debugging and trace-back. But comparing the price tag of a system development kit and the prototyping equipment generally used for developing microprocessor based systems it is an obvious choice.

- The hardware connection details are presented in figures 1,2 & 3. For further details you can refer to the report presented by the authors to the Board of Advanced Studies & Research, NWFP UET, Peshawar.

**REFERENCES**

1. Intel's 8085 user's manual
2. Intel's Memory component hand book
3. Intel's micro controller hand book
5. Albert P. Malvino, "Digital Computer Electronics".
6. Rodney Zaks, "Microprocessor Interfacing".
7. Hewlett Packard, "Practical Microprocessor".
8. SDK-8085 user's manual.