Versatile Real Time Microprocessor Development System

Hossam Eldin M. Abdelbaki
Department of Electrical Engineering, Alexandria University, Alexandria, 21544. Egypt (email: ahossam @ cs.ucf.edu).

Abstract--This paper describes the design and use of an EPROM emulator and programmer as a low cost development tool. The design is based around a very powerful flash risc microcontroller. In conjunction with cross assemblers and linkers, the emulator allows executable code to be developed using a host computer, and to be directly executed by the target system. The design is flexible in that it can be used as a development tool for target systems based on a variety of microprocessors. Also it can be used as an interface to a host computer or as a standalone unit if the host is not available. These capabilities can eliminate the need to burn EPROMs, erase EPROMs if the code is incorrect, and repeat the sequence until the code is fully debugged.

Index terms--Atmel flash microcontrollers, EPROM emulator, EPROM programmer, Microprocessor developing system.

I. INTRODUCTION

Low cost software development tools such as cross assemblers and linkers [1] for various microprocessors are readily available to create executable modules. Although simulators [2] can be used to test the software without relying on the actual microprocessor, they are useful in debugging logical errors but cannot test the true environment of the target system. This is particularly true for programs that are time-critical, or that involve interrupts and I/O operations where the I/O devices employed by the target system may not be available in the host system. In the final phase of a project design, the program must be integrated with the target system for testing and debugging.

To evaluate any microprocessor system, not only must the actual chip be bought but a developing board must also be available. One factor almost all evaluation boards have in common is the monitor program that is resident on the board, this represents an extra hidden cost in the design of a simple prototype board. This is true for minimal systems such as student generated projects where typically switches or hexadecimal keypads are used for input and LED or seven segment display units are used for output, and resident monitor is not available [3].

![Figure 1: Block diagram of the proposed EPROM Emulator / Programmer](image)

This project, as shown in Figure 1, describes the design and use of EPROM emulator and programmer as a low cost development tool. The design is based on one of the state of the art flash microcontrollers, AT90S8515 from Atmel [4]. In conjunction...
with cross assemblers and linkers, it allows executable code to be developed using a host computer, and to be directly executed by the target system. The project is flexible in that it can be used as a development tool for target systems based on a variety of microprocessors. Also it can be used as an interface to a host computer or as a standalone unit if the host is not available. These capabilities can eliminate the need to burn EPROMs, erase EPROMs if the code is incorrect, and repeat the sequence until the code is fully debugged. The main processing components in the project are the AVR risc microcontrollers AT90S8515 and AT90S1200 \[5, 6, 7\]. The idea of the emulation depends on having a RAM, which will be called emulation memory, that can be accessed either by the processing microcontroller or the target microprocessor. By enabling the microcontroller side, the object programs can be downloaded into the emulation memory, either by the host or the keypad, then, the microprocessor side is enabled so that the target microprocessor supplies the address and data signals for the emulation memory and executes the written program. If the program is working properly, the microcontroller side is enabled and the previously written program is burned into an EPROM by the programmer supported in the unit. On the other hand, if the program needs modifications, it is amended and downloaded again into the emulation memory and the process continues until the final working software is developed.

The unit has been successfully tested on a variety of microprocessors and microcontrollers (which accepts an external program memory) such as 8085, 8088, Z80, 6809, and 8031.

The paper is organized as follows. Section II gives the hardware modules, Section III presents the software developed for the project, Section IV suggests some possible future modifications for the project.

II. Hardware Description

The project can be divided into 11 hardware modules. For each module, a block diagram is shown.

1) Microcontroller module:

This is the main processing module in the project, it is based on the AT90S8515 microcontroller. The microcontroller ports are used extensively in handling the emulation memory, controlling the multiplexed display, and serial communication with the host or the keypad. PORTA and PORTC are connected to the memory interface module, memory control module, and to memory module. PORTB and PORTD are connected to the display and to the serial communication module. X1 is a 4MHz clock input for the microcontroller.

2) Serial communication module:
The serial computer interface uses a MAX232 converter, which converts the serial RS232 levels to and from the computer to standard TTL for the emulator. The module is connected with the host via the signals PC_TX (PC transmit signal) and PC_RX (PC receive signal). It converts these signals to standard TTL signals named CON_TX and CON_RX. The common of JMP1 is connected to UC1_RX pin which is the UART receive input for AT90S8515. This pin can be connected to either CON_TX or UC2_TX which is the UART transmit pin for AT90S1200. The first connection is to enable the host to communicate with the unit and the second connection is used when the host computer is not available and the keypad is used to enter the programs to the emulator. JMP2 connects the CON_RX signal to UC1_TX so that the host computer can receive the response of the emulator. For manual mode, the keypad discard any response from the emulator circuit. On the other hand, CON_RX can be connected to UC2_TX when it is required to test the commands sent by the manual keypad module.

3) Memory module:

This module consists of a 64k byte static ram and a 64K byte EPROM. The static RAM will be called the emulation memory which can be read or written to by the main microcontroller, on the other hand, the microprocessor accesses this memory as a read only program memory. The emulation memory is used for downloading programs without needing to burn an EPROM. Once the developer becomes satisfied with his code, he can dump it to the EPROM. The buses of both memories can be accessed either by the main microcontroller or by the target microprocessor as will be seen later. The address and data buses are supplied from the memory interface module and the control signals are supplied from the memory control module.

4) Memory interface module:

This module switches between the microcontroller side, where it is possible to download object codes into the emulation memory, and the microprocessor side, where the target microprocessor can execute the downloaded programs. The main microcontroller supplies two important control signals, ENUC and ENUP, to do the switching tasks. When ENUC = 0 and ENUP = 1,
the microcontroller side is enabled for program download mode. In this case, the target microprocessor is put in hold state by connecting the hold pin of the processor to ENUC if it is active low or connecting its hold to ENUP if it is active high. In the same time the processor buses are isolated from the memory bus and the microcontroller signals are connected to the memory. If ENUC = 1 and ENUP = 0, the microcontroller side is isolated from the shared memory and the microprocessor is enabled to execute the program in the emulation memory.

Switching to microprocessor mode is done by sending the ASCII command “U” to the unit as will be discussed in Section III. The module accepts the signals UC_D7 - UC_D0 and some other control signal from the main microcontroller module. Since the emulator can dump and verify the written programs, the data bus is connected to 74245. The flow direction is controlled via the memory output enable signal, M_OE. To save the microcontroller pins, multiplexing is used to send the lower and higher address bus signals using and latching them using the two signals LATCL and LATCHH.

5) Memory control module

This module is responsible for connecting the memory module to the microcontroller side or to the microprocessor side. This is accomplished using tristate buffers which are controlled by the signals ENUC and ENUP. The main microcontroller generates the output enable signal and the write signal for the RAM, UC_OE and UC_RW, respectively. It also generates a chip select signal, UC_CS, where UC_CS = 1 for RAM selection and UC_CS = 0 for EPROM selection. The EPROM chip select may be low for normal program execution or pulsed low during EPROM burning. This switching is established using tristate buffers that either connect M_ECS to US_CS for normal reading and to a 50ms pulse during burning process.

6) Keypad module

If the host computer is not available, the emulator can still be used as a standalone unit thanks to this module which partially simulate the host computer. The module has a 25 keys keypad which has the 16 hex numbers ( 0 to 9 and A to F) as well as the important single character commands used for communication with the main microcontroller. The processing part of the unit is
the secondary microcontroller, AT90S1200, which scans the keypad and send the ASCII code for the pressed character to the main microcontroller. Unlike the host, the keypad module does not receive or process any response from AT90S8515.

7) Display module

![Display Module Diagram]

This module has 6 common anode seven segment displays. They are used to display the 8 bit data and the 16 bit address currently handled by the memory. PORTD of the main microcontroller is used to generate the anode selection signals and PORTB is used for generating the segments from a through g. The display is multiplexed under software timer interrupt control.

The hardware prototype of the project circuit is shown below.

III. Software Description

The software is written in a structured systematic fashion with explicit functions for each simple task. It is not optimized for speed or memory requirements. The software is written in C++ using Borland C++ Builder [8]. The main processing tasks are handled by the monitor code written into the AVR AT90S8515. The microprocessor program may be dumped into the memory of the emulator via one of the following methods:

a) Using the built in ASCII based command set which allows it to operate with any host through a serial port and a terminal program.

b) Using a high level language program that allows the users to download program into the emulation memory.
c) If the host is not available, the external keypad can be used to dump the object code byte by byte into the emulation memory. Methods 1 and 3 will be called console or terminal modes and method 2 will be called program mode.

The monitor software is divided into four categories: initialization routines, command routines, monitor support routine, and general purpose routines. The initialization routines are executed at the beginning of the monitor program. The monitor routines are those invoked as a response to the ASCII commands. The monitor support routines are called by the command routines. The general purpose routines are useful utilities.

1) Initialization routines

The initialization routines are executed once at the beginning of the monitor program. They include stack initialization, Timer0 setup, enabling interrupts and setting interrupt vectors, UART initialization with baud rate of 9600, setting up the ports as outputs, and initializing the emulation memory with hexadecimal values “FF” (to emulate a blank EPROM), and finally setting the operation mode as console mode. The main program follows the initialization and it is simply an infinite loop that waits for a single ASCII character (command) from the serial link, checks it and, and executes the corresponding command routine.

2) Command routines

The emulator responds to some single ASCII letters commands. The lower case letters are used for program mode and the upper case letters are used for the console mode. Almost all the commands associated with the console mode are supported with the external keypad. So we will limit our discussion to the host console mode. For example, the ‘A’ commands used for locating a certain memory address. ‘D’ command is used for changing the contents of a memory location. The program mode option facilitates using high level languages to communicate with the emulator.

By sending the appropriate commands, the object code for the target microprocessor is dumped into the emulation memory. After dumping, the microprocessor side is enabled using the ‘U’ command and the AT90S8515 goes into sleep mode. At this point, the microprocessor buses are connected to the emulation RAM and after pressing the reset button of the target microprocessor system, the program is executed. If there are needed modifications, the microprocessor side is disabled and the microcontroller wakes from sleep mode by pressing an external push button that provides a hardware interrupt for the AT90S8515. This steps are repeated until the user is satisfied with the program, now by choosing the EPROM (using ‘M’ command) and then issuing the program EPROM command, ‘W’, the final version of the object code in the emulation memory is dumped into the EPROM. The EPROM is checked first if it is blank or not and if it is not, the programming stops.

As an example, consider that it is required to examine the memory contents from address 0000 to 007F by using the ‘E’ command. The following screen will be dumped into the terminal program after issuing the command in terminal mode.

If the program mode was set and the ‘e’ command is issued, the following will be sent back to the terminal

The program mode option facilitates using high level languages to communicate with the emulator. To illustrate the usage of the SERIAL.cpp program, consider the following C code that increments the address of the current memory location by 1.

```c
send_character('P'); // set the emulator to work in program mode
send_character('f'); // increment the current memory address by 1
```

where the definition for the function send_char is given below.

```c
void send_character(int ch) {
```
TWENTEITH NATIONAL RADIO SCIENCE CONFERENCE
March 18-20, 2003, CAIRO, Egypt.

//0x3F8 is the address of COM1 port.
char status;
do
{
    status = inportb(0x3F8 + 5) & 0x40;
} while (status!=0x40);
outportb(0x3F8,(char) ch);

By sending the appropriate commands, the object code for the target microprocessor is dumped into the emulation memory. After dumping, the AT90S8515 goes into sleep mode and the microprocessor side is enabled using the ‘U’ command. At this point, the microprocessor busses are connected to the emulation RAM and after pressing the reset button of the target microprocessor system, the program is executed. If there are needed modifications, the microprocessor side is disabled and the microcontroller wakes from sleep mode by pressing an external push button that provides a hardware interrupt. This steps are repeated until the user is satisfied with the program, now by choosing the EPROM (using ‘M’ command) and then issuing the ‘W’ command, the final version of the object code in the emulation memory is dumped into the EPROM. The EPROM is checked first if it is blank or not and if it is not, the programming stops.

3) Monitor support routines

The monitor support routines are called by the command routines. They include routines for dealing with address and data and serial communication. As an example of these support routines, the routines invoked when you issue the ‘A’ command. A function called GetAddress is invoked which invokes another function, GetNibble, four times. GetNibble invokes by its turn the function getc which receives a character from the serial port, and echoes it back to the terminal. The four characters are converted to a 16 bit address and the contents of that address is read and echoed back to the terminal.

One important routine is the display routine which is invoked automatically every 5 ms using Timer 0 overflow. This routine gets the current memory address and displays it on a 4 seven segment display and gets the current memory content and displays it on a 2 seven segment display. This display is essential when the host is not available and the unit is to be used in a standalone mode. Monitor commands ‘S’ and ‘C’ are used to clear the software interrupt or enable it according to the mode (usually the software interrupt responsible for the display is enabled in console mode and disabled in program mode). The pattern for the hexadecimal numbers are stored on the EEPROM of the AT90S8515.

4) General purpose utility routines

These routines are invoked so often when the program is running. Examples of such routines are ascbin (ASCII to binary), and binasc (binary to ASCII) which are very useful for handling the data between the host and the emulator. Other routine check the validity of the arguments passed from the terminal, for example, a valid hexadecimal address or data should be checked if its digits are in the ranges 0 to 9 or A to F.

On the other hand, in the keyboard code, another program is written for the microcontroller in the keypad module, AT90S1200. Since AT90S1200 does not support hardware UART, a bit banging software is written to implement a UART with 9600 baud rate. Also the software scans a 25 keys keypad and send the corresponding command through the implemented UART simulating the host in terminal mode. The keypad keys include the hexadecimal numbers from 0 to 9 and A to F. Also they include 9 commands B, F, R, M, U, S, C, Z, and O.

One of the important features of the proposed project is that it can be used as a standalone emulator (without a host), Thanks to the reliable serial communications that can be established between AT90S8515 and AT90S1200.

IV. Conclusion and Future Modifications

This project describes the design and construction of a versatile EPROM emulator and programmer based around Atmel AVR risc microcontrollers. This design enables electronics enthusiasts and professional to develop applications using any 8 bit microprocessors. Instead of keep blowing EPROMs each time you change a firmware - based code, download the object code through a host computer equipped with a serial link, or simply enter your code manually into the emulator’s static ram which will operate like an EPROM to the system into which it is plugged. The unit is extremely flexible and easy to use and can em-
ulate standard 27 family of devices, from 2716 (2K) to 27512 (64K). Once the software has been tested, it can be dumped into an actual EPROM supported by the emulator. The programmed chip can be inserted into the target microprocessor system and a final working product is obtained.

The performance of the current project can be enhanced using the following suggestions:

• Extending the project to deal with 16 bit microprocessors. This needs the addition of another emulation memory to handle the high byte of the data bus.

• Using an LCD instead of the seven segment display. In this case, more user friendly messages can be displayed, also AT90S1200 can be programmed to accept the response of AT90S8515. This may allow the keyboard and display modules to act more similar to the host computer.

• Adding an additional RAM to simulate the stack of the microprocessor system, this will help in checking the contents of the stack any time.

REFERENCES