2006-386: A NEW APPROACH IN MICROPROCESSOR/MICROCONTROLLER COURSES/LABORATORIES MATERIAL DESIGN AND DEVELOPMENT

Steve Hsiung, Old Dominion University

STEVE C. HSIUNG Steve Hsiung is an associate professor of electrical engineering technology at Old Dominion University. Prior to his current position, Dr. Hsiung had worked for Maxim Integrated Products, Inc., Seagate Technology, Inc., and Lam Research Corp., all in Silicon Valley, CA. Dr. Hsiung also taught at Utah State University and California University of Pennsylvania. He earned his BS degree from National Kauhsiung Normal University in 1980, MS degrees from University of North Dakota in 1986 and Kansas State University in 1988, and a PhD degree from Iowa State University in 1992.

Jeff Willis, Utah State University

Jeff Willis Jeff Willis is a Software Engineer developing Mission Planning Software at Hill Air Force Base in Utah. He earned a BS degree in Computer Electronic Technology and a Masters degree in Computer Science from Utah State University. As part of his Master's Thesis he co-authored two papers on self-configuring, deterministically latent intercommunication architectures for satellite payloads.

A New Approach in Microprocessor/Microcontroller Courses/Laboratories Material Design and Development

Abstract

Courses in microprocessors and microcontrollers are standard parts of the Engineering Technology core curricula. The traditional course material developments include both lectures and associated laboratory exercises. No matter how creative is the curriculum; it is usually budgetary constraints that confine the creativity when developing new curricula. This limits the freedom of the major approach in new course development.

This article demonstrates new course lecture and laboratories material development that starts from ground up with both a hardware platform and simulation software design for microprocessor/microcontroller related courses. It is not only very cost effective, but also does not limit the instructor's creativity when developing new curricula. The only obstacle is the instructor's imagination on courses and laboratories activities. This system can be implemented at no cost to the department for sponsoring the courses. As a matter of fact, the initial trials of this system have generated revenue, thereby supporting future improvements and development needs.

This new approach in course improvement starts with the design of a hardware platform in a custom made evaluation board. It involves the system circuit and power supply design, printed circuit board layout, prototype testing, and circuit board fabrication. The second step is to design the simulation software for laboratory uses. The total design and development of both software and hardware was a two year evolutionary process.

I. Introduction

The 68HC11 EVB (evaluation board) was made by Motorola, Inc. in the 1980's.[9] Due to the effort of Motorola University Support program, this EVB was very popular in most of the universities and community colleges microprocessor/microcontroller related courses and projects designs. When Motorola spin off their microprocessor division to Freescale Inc.,[5] the 68HC11 EVB became very hard to obtain. The alternative EVB made by Axiom is more expensive.[1] Another draw back is that the alternative board has limited functions as compared to the original Motorola 68HC11 EVB.[1,9]

In order to extend the use of the 68HC11 EVB and keep all 68HC11 CPU laboratory exercises and project designs intact, the design/development of a modified Motorola 68HC11 development system became a reasonable choice. The objectives of this new approach are: (1) sustain the use of the 68HC11 CPU, (2) keep the EVB hardware cost to a minimum, (3) make a smooth transition from 8 bit CPU to 16 bit CPU applications, (4) give students ownership of flexible hardware that can be used in several courses, and (5) relieve the financial burden on the institution. After two trials in designing and testing of the hardware circuits and implementation in the laboratory with students for two years, this hardware was named the "CETHC11EVB2" and has been successfully used in several related courses.

To minimize the students' errors in utilizing the 68HC11 instructions and addressing modes, a teaching assistant software simulator was also developed to be used with this hardware. This software is not a comprehensive simulator and is not intended to compete with

commercially available packages, but it assists student learning, reduces mistakes, and enhances trouble shooting skills.[15] This simulator software is named as "CETHC11SIM" and has been used in the beginning level microprocessor/microcontroller course.

II. The CETHC11EVB2 Hardware Designs

The design of the hardware is based on the original Motorola HC11EVB structure that uses the 68HC24 port replacement unit to bring all the available I/O ports to the user's control while still running 68HC11 (marked as U1) in its expended mode.[9]

1. Keeping All the Physical Available Pins of the 68HC11

The 68HC11 has four different modes controllable through the MODA and MODB pins: Single Chip Mode: Uses the 68HC11 in its independent state only and does not allow for any external memory to be attached.[2,8,11] Expanded Mode: Uses the 68HC11 along with external memory (EEPROM, EPROM, RAM).[2,8,11] This mode sacrifices general I/O ports. Bootstrap Mode: Uses the 68HC11 for internal memory programming. Special Test Mode: Used for 68HC11 factory test operations.[7,8], Under normal conditions, if the 68HC11 is running in expanded mode, all the standard 16 I/O pins must be used for address and data buses for accessing external memory. Therefore, there is no general I/O available for the user to use in intended applications. This is unacceptable in academic laboratory exercises. These 16 I/O pins can be made accessible by forcing the CPU into single chip mode, but this restricts the CPU's available memory to the on-chip memory, which, in turn, restricts the user's programming code size. For laboratory instructional purposes, this is also undesirable. The design of the CETHC11EVB2 uses the 68HC24 (marked as U7) to make all 16 I/O pins available to the user and keep 68HC11 running in expended mode, thus allowing access of 8K or 16K RAM, and 8K EPROM at the same time. This 68HC24 called a port replacement unit was designed by Motorola for this particular purpose but was later discontinued.[5,14] We are able to get this important part from Tekmos, Inc. (base in Austin, TX) that continues manufacturing it at a slightly higher cost.[14]

2. The Communications and Memory Map

T It was not necessary to design the software for communication between the CETHC11EVB2 and a PC, since Motorola already has a good user interface software called BUFFALO that is stored in a 2764 (an 8K EPROM marked as U6), and decoded at address $E000. The communication uses the SCI on the 68HC11 and the RS232 on a PC serial port.[9] A simple MAX 232 chip (marked as U9) is used for the communication signal conversion between TTL and ±12V levels, for downloading and debugging. The CETHC11EVB2 uses 9600 BAUD rate, 8 data bits, 1 stop bit, and no parity to communicate via a PC's serial port.

The system is built around the structure specified by BUFFALO which keeps all the interrupt vectors the same as 68HC11 families.[7,8] This makes the system board flexible enough to accept almost all available 68HC11 chips. There are jumper selections (J4, J5, & J6) to choose 8K RAM (6264 static RAM) or 16K RAM (62256 static RAM) chip marked as U8. This design gives the CETHC11EVB2 the flexibility to accept different RAM chips, depending on the availability and cost.

3. The Power Supply and Connections

There is also an on-board power unit that uses +5V regulator, 7805 (marked as U10), to regulate any wall mount power plug that ranges between 6V and 12V DC down to +5V for the operation of the system board and possible low power to the users' external experimental circuits. This allows the CETHC11EVB2 to be operated in any convenient place. There are two 0.1" dual row receptacles (marked as J1 & J2) that are used for interfacing between the CETHC11EVB2 and a user's breadboard circuit or any prototype board. These connectors can adapt any jumper wires to standard breadboard or any header pin size connectors to any user interface application.

4. The Features

This system is the second version of the HC11 evaluation board. It is an improved design for enhancing lab experimentation and project use. Based on the suggestions from the students, the CETHC11EVB2 has several modifications over its predecessor CETHC11EVB and providing several advantages:

- CETHC11EVB2 uses an HC24 port replacement unit to gain access to Port B and Port C I/O pins control.
- There is no need to change the HC11's mode. The CETHC11EVB2 always remains in Expanded Mode.
- CETHC11EVB2 has full access to external 8K or 16K RAM (Jumpers: J4, J5, & J6 selectable) while maintaining control over all I/O pins. This makes the system flexible enough to accept different RAM chips based on the availability.
- CETHC11EVB2 can be used as a full target system development tool. The only required changes in the developed software are the starting address and RAM access area when moving to a single chip EEPROM or ROM or RAM memory.
- There are no problems in dealing with insufficient memory and mode changes on the CETHC11EVB2 as compared to other EVBs that only operate the 68HC11 in single chip mode.
- Most of the available 68HC11 family members can be used on the CETHC11EVB2.

There are different kinds of software can be used on this CETHC11EVB2, such as (1) DOS based AS11.EXE for assembler and KERMIT.EXE for downloading,[9] (2) a window based AXIDE from Axiom,[1] and (3) public domain window based MiniIDE.[12] These are all compatible with the CETHC11EVB2. The U8 socket accepts an 8K*8 byte or 32K*8 byte RAM, the speed of which is irrelevant.

Figure 1 and 2 are the CETHC11EVB2 system circuits that use (a) external memories at addresses $E000-$FFFF for the BUFFALO monitor program and $4000-$7FFF for the user RAM, and, (b) internal memory at $0000-$00FF or $0100-$0FFF (depending on the type of 68HC11 family members) for interrupt vectors and user memory.[7,8,9] Picture 1 presents a photo of a fully assembled CETHC11EVB2 system board.

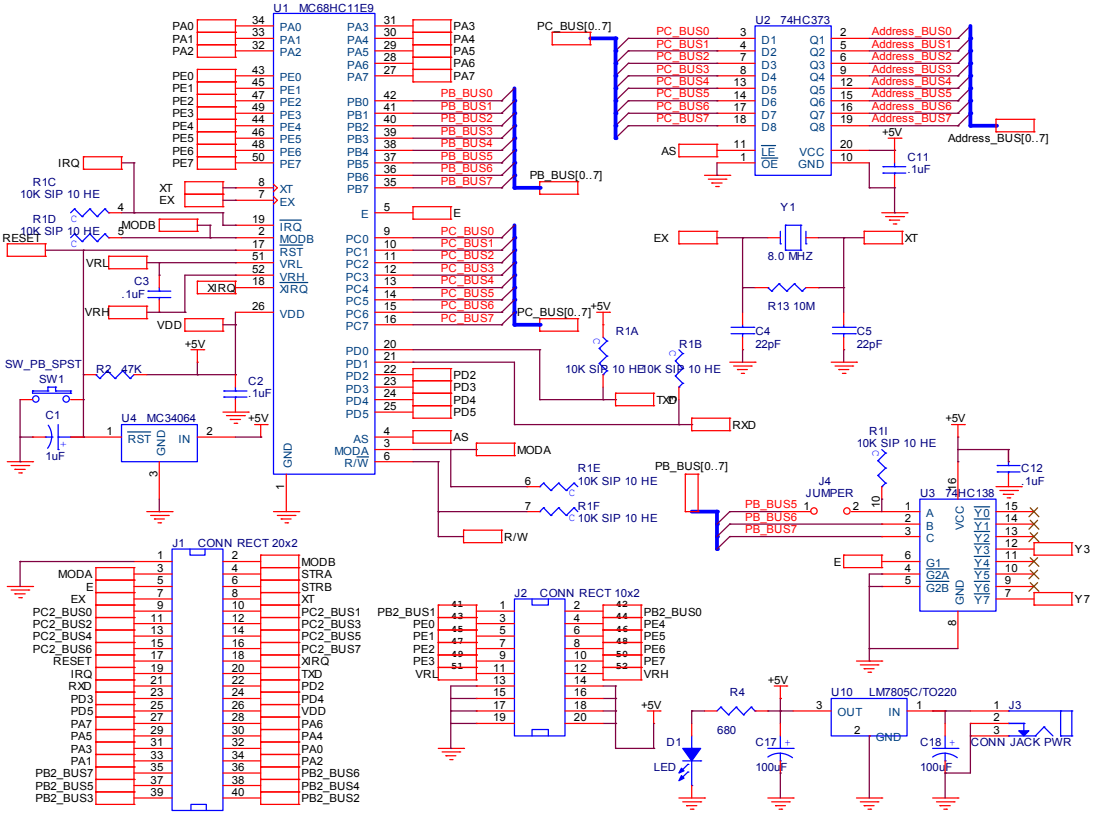


Figure 1. CETHC11EVB2 System Circuit 1

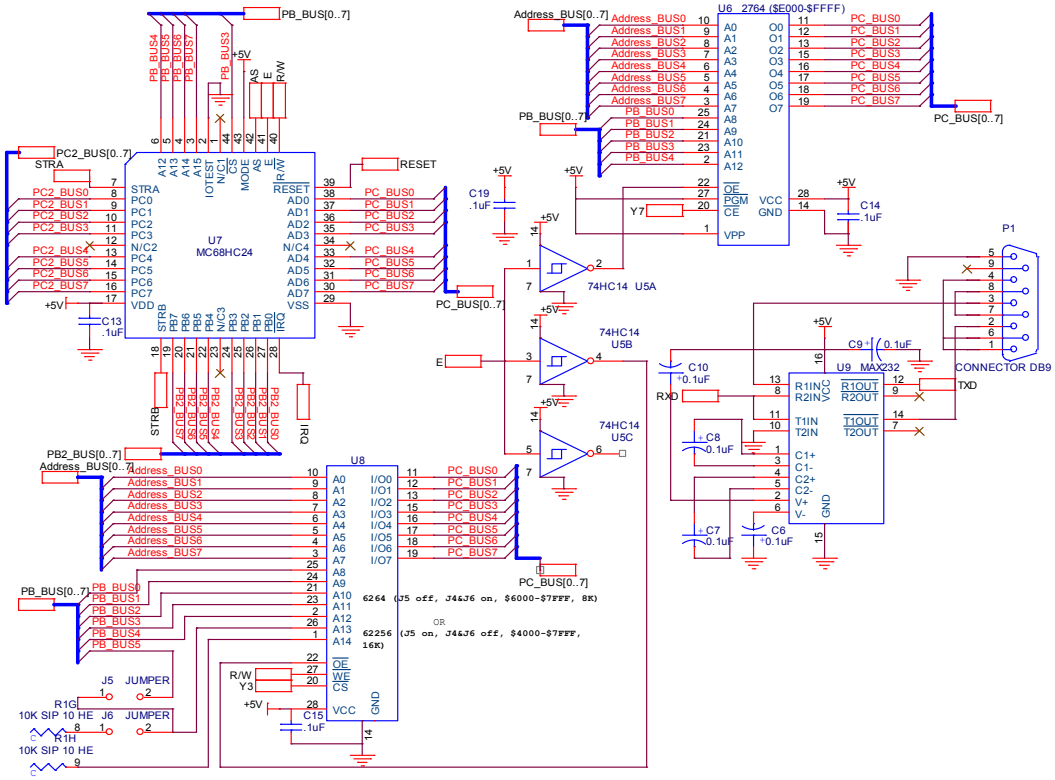
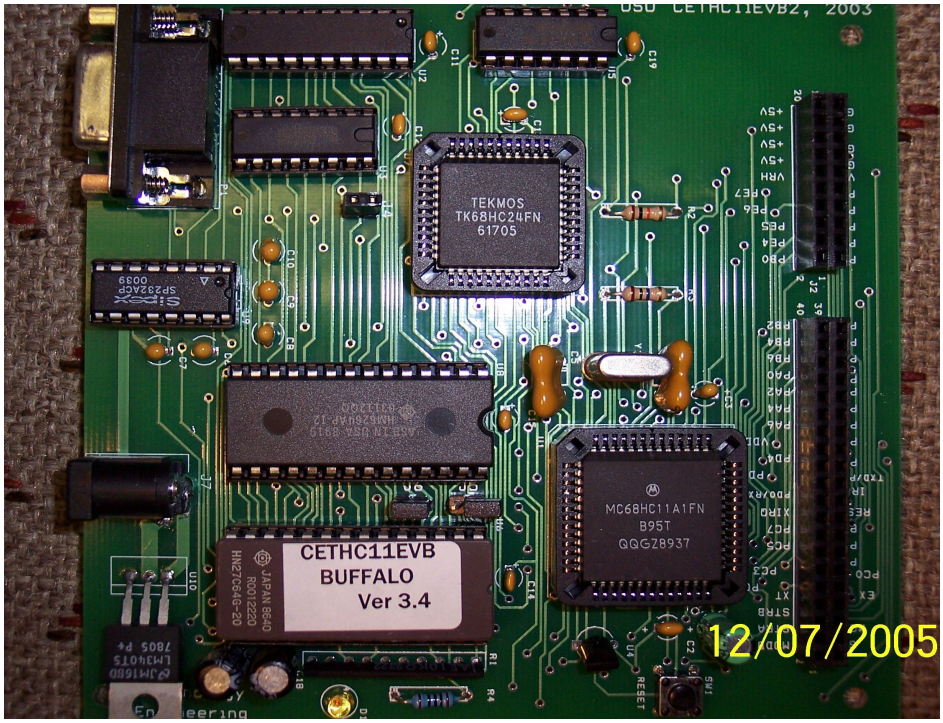


Figure 2. CETHC11EVB2 System Circuit 2



Picture 1. CETHC11EVB2 System Board

III. The 68HC11 Simulator Designs

The CET11SIM simulator is a simple and easy to use tool designed for Motorola's MC68HC11 microcontroller. CET11SIM allows users to gain a better understanding of the architecture and operations of the HC11 microcontrollers. It takes an assembled listing file and allows a user to step through each individual operation and display all hardware registers and memory changes accordingly; this gives the user a perspective on how the file they created interacts with the HC11 hardware.

Debugging a piece of software can leave a person feeling discouraged and sometimes confused. With assembly level programming this is more common than not, especially for students who are first time learners and novice programmers. The majority of the time the bug is a simple syntax error, such as inadvertently using an incorrect addressing mode. Such errors are difficult to diagnose even for the professional developer. Debugging these errors is a time consuming process. With every modification to the program the user must assemble and export the program to the microcontroller, and then execute and examine the results for clues, which hopefully will lead to a solution.[2]

The CET11SIM application is designed specifically for Motorola's MC68HC11 eight bit microcontroller series. The CET11SIM simulates the HC11 by performing all of the HC11 operations and then displaying the results of each operation. This allows each HC11 instruction to be verified and corrected immediately before it is downloaded to the HC11's internal/ external memory for verification. The repeated time-consuming process of exporting the HC11 program to the microcontroller, executing and then verifying the results can be nearly eliminated.

1. Overview of Operations

CET11SIM operation consists of opening and loading a list file to the CET11SIM. The list file, a text file, is a product of the assembling process. The programmer creates this text file which contains the operations to be performed by the microcontroller. The text file is an input file for the assembler. The assembler generates a hexadecimal file (that will be loaded to the memory of the microcontroller for execution) and the list file. The list file displays the relationship between the user created text file and the hexadecimal file executed by the microcontroller. The list file also displays any assembly errors to the user.[2,7,11] The assembler adds to each line a line number, the hexadecimal value of each opcode, and its arguments, as shown in Figure 1. CET11SIM displays this file to the user with the next line to be executed highlighted in green. A break point is registered by clicking on the line where the break point is to be established, at which point the first four characters of that line are highlighted in red.

A user can press either the "Step" button to execute each HC11 operation individually or the "Run/Continue" button which will execute successive operations until a break point is encountered. At this point the user may step through each individual HC11 operation. After each operation is executed the results can be viewed in hexadecimal (default), decimal or binary. There is a dedicated text box for each HC11component. These include accumulators A, B, and D; index registers X and Y; the Stack Pointer (SP), the Program Counter (PC) and the Condition Code Register (CCR).[2,4,9]

The contents of the HC11's memory are displayed in a table that indicates an address and the address contents. Each time a value in memory is modified it is highlighted in yellow, which makes finding and viewing pertinent memory locations easier.

The IRQ interrupt is the only interrupt supported by the CET11SIM. The user triggers an IRQ interrupt by clicking the "INT" button. This causes the program counter to load the dedicated interrupt vector address and perform the operation indicated by the values at that address. If the memory contents at the interrupt vector address contain appropriate instructions, the interrupt service routine will be executed. [2,11] This gives the user the ability to examine a process that is usually hidden.

2. Operating Instructions

The proper operations of the CET11SIM simulator can be classified in the following three steps.

(1) Step1: Java's Run-Time Environment Installation

The only system requirement is that Java's virtual machine is properly installed on the machine where the application is to be executed. The Java virtual machine or "Run-Time Environment" can be downloaded at http://java.sun.com/getjava/. CET11SIM was developed using the Java run-time environment version 1.3.1_02 and tested up through version 1.4.2_06. The CET11SIM executable file is a Java jar file.[3,13] Once the Java runtime environment is installed, the jar file can be placed in any directory to be executed. To execute the jar file double click on the file. The application will then start.

(2) Step2: Creation of a List File

A list file contains a line number, a hexadecimal address and the hexadecimal values of

the opcodes. All are used as input for the CET11SIM to model the current state of the HC11 microcontroller. The list file is a text file that can be created when an HC11 program is compiled or assembled. The ".s19" file is a hexadecimal file that is created with the specific purpose of being executed on the HC11 hardware. Most assemblers generate a list file by default. The user can generate a list file from the command line using "as11 filename -l > filename.lst".[9] This redirects the list file output of the as11 command to a destination file. The generated list file can then be used with CET11SIM.

There are many freely available HC11 assemblers that will automate this process. These Microsoft Windows based assemblers are available from web sites, such as AxIDE at http://www.axman.com/files/Other/AxIDE/AxIDE363.exe or MiniIDE at http://www.mgtek.com/miniide/Download. Either of these assemblers will assemble HC11 source code and generate the ".s19" and ".lst" files.[1,12]

(3) Step3: Execute the CET11SIM Applications

To execute the CET11SIM, simply double click on the executable jar file. To open a list file click the "Open list file" at the top of the application, this will bring up a dialog box. Select the list file you would like to debug and open that file. The application will appear as shown in Figure 3.

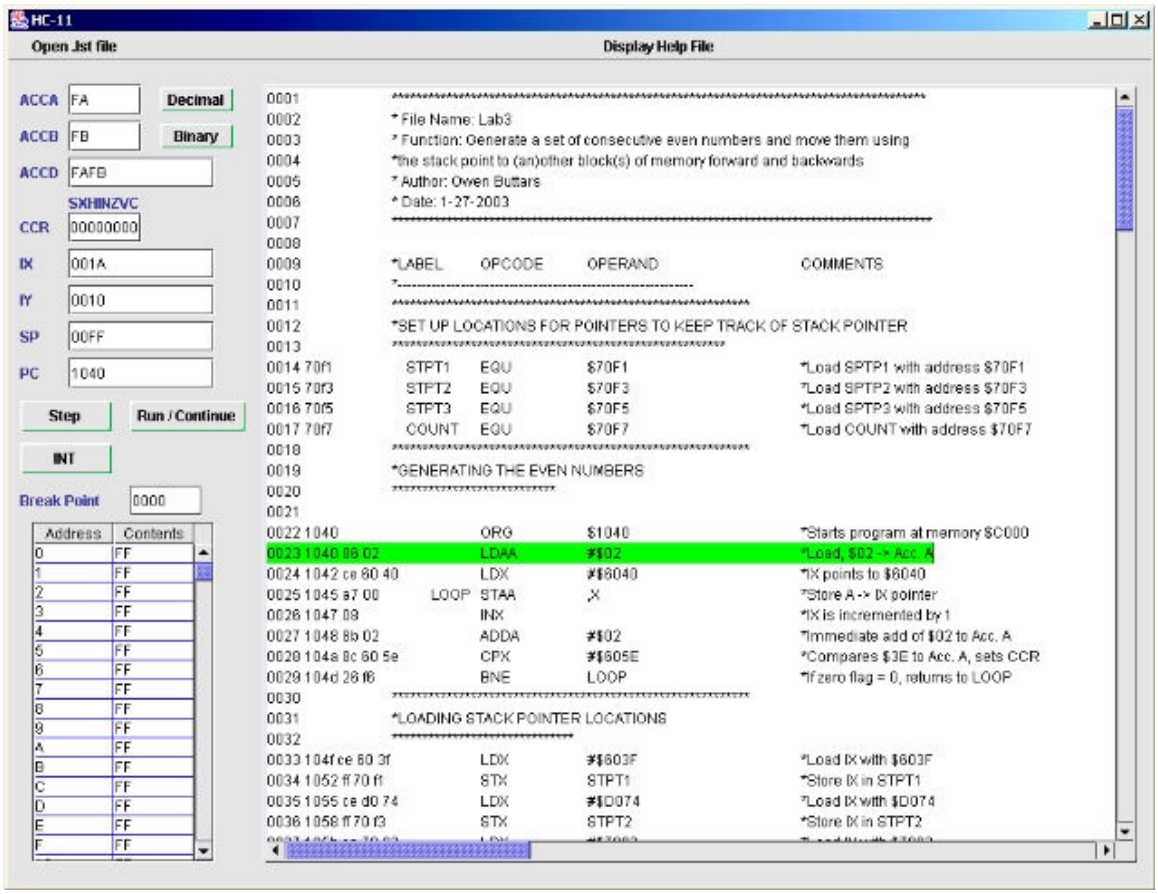


Figure 3. Display Screen of CET11SIM

After setting a break point and stepping through a few instructions, the CET11SIM application will look similar to Figure 4.

The line highlighted in green will be the next line to execute. CET11SIM uses the "ORG" opcode to establish an entry point for the HC11 application, the first opcode following the "ORG" instruction will be the first executed.[2,7,16]

The "Step" button executes the highlighted line, updates the HC11 and the display.

The "Run/Continue" button executes successive lines until a break point or the end of the program is encountered.

Only one break point is allowed, which can be changed at any point. When a new break point is recorded the old break point will be discarded. To place a break point, just point and click on the line of the file where the break point is desired. The break point display area shows the line number where a break point has been recorded. The line number will also be highlighted in red to indicate the break point.

The values in the registers are displayed by default in hexadecimal, except for the Condition Control Register (CCR), which is displayed in binary. The decimal button changes the HC11's displayed values to a decimal representation. The binary button changes the HC11's displayed values to a binary representation. The values will be returned to a hexadecimal representation after the next line is executed.

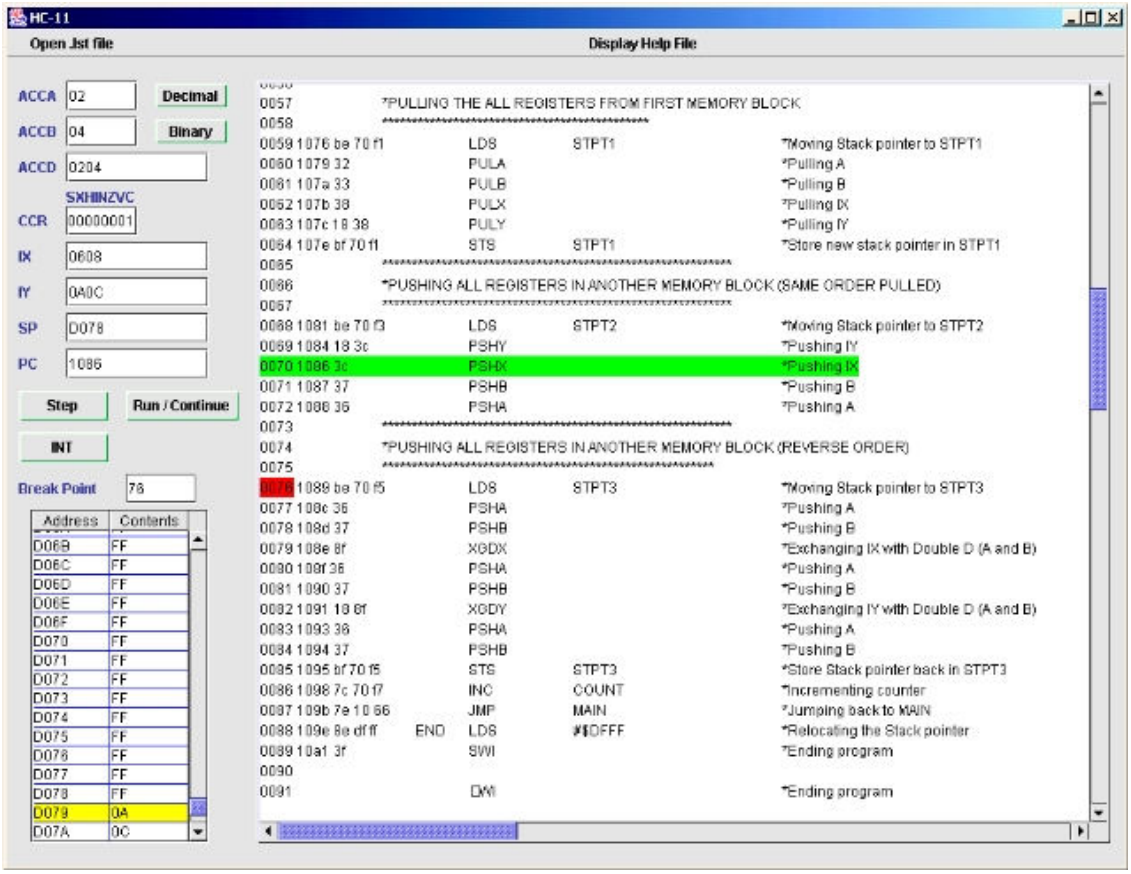


Figure 4. Single Step through CET11SIM

The area below the "Step" and "Run" buttons is a display of the current contents of the HC11's memory. When these values are changed, the changed address and value are highlighted with yellow to make it easier to verify the program's operation.

The INT (Interrupt) button triggers the Program Counter (PC) to load $00EE, which is the IRQ Interrupt vector. The opcode at this address is executed. If the user has properly linked their Interrupt Service Routine with the interrupt vector table the first line of the Interrupt Service Routine will be the next line to execute. The following is an example of the contents of the interrupt vector table.[7,9,12]

Address	Contents	Description
$00EE	$7E	Jump Instruction
$00EF	ISR address High byte	location of Interrupt Service routine (high byte)
$00F0	ISR address Low byte	location of Interrupt Service routine (low byte)

Figure 5 is an example of what might be seen immediately following pressing the IRQ button. Notice the contents of all registers have been pushed onto the stack.

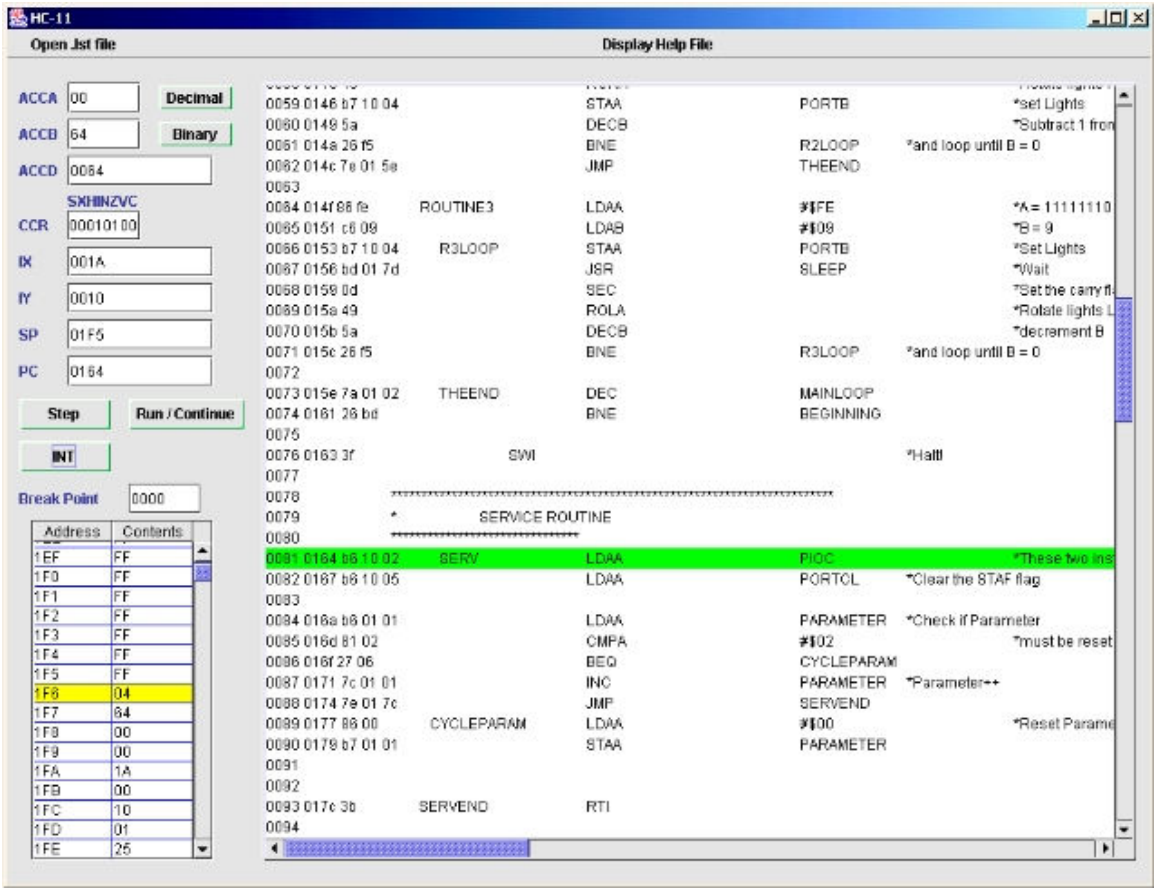


Figure 5. Interrupt Operation in CET11SIM

At any point a new list file may be opened to debug. This will overwrite the previous data objects used to model the HC11 and create a new model from the program loaded.

3. The Development

The CET11SIM was developed using a broad and simplistic approach. It is designed specifically to target university students. Colleges around the country utilize many different types of computer systems, so a platform is needed that is independently executable. CET11SIM is not designed to meet the needs of a commercial user so execution speed was not a primary concern; these two factors led to the use of the Java programming language.

Java is similar to C++ in that they are both object oriented programming languages and they follow the same syntax guidelines. They both allow the development of unique data structures, making them very versatile. However, there are a few things that differentiate the two. Java is platform independent meaning once the code is compiled it can be run on any computer with the Java virtual machine. Java is much more reliable than C++ with its built in exception handling. Exceptions allow a program to continue running even after an error has occurred. For example if a program is expecting a specific file format for input and an incorrect format is encountered, the program will create an exception. This means the program will stop executing the code to input the file and return to the state previous of the file being input. Java does not allow multiple inheritances which removes a great deal of complexity. Java does not use pointer syntax. Everything is either an object or a primitive such as float, integer or character.[3,13] This makes Java code easy to read and write. Java is free to any non-commercial user and its documentation is easy to understand.

4. Software Structure

The main component of the CET11SIM is the window or frame. Its job is to take information input by the user and route that information to functions that will handle the request. Java's JFrame class is used as a base class that was extended to fit the project's specifications. The JFrame object controls how each component inside the frame is displayed, how messages between components are handled and the actions that result when an event such as a mouse click occurs. Sub-components such as the text area, buttons, text fields, and the menu all reside within the JFrame object. [3,13]

The key component in the JFrame class is the TextArea class. This class displays the list file that the HC11 simulator will be executing. The TextArea class interacts with the user by allowing break points to be set and displayed and by displaying the next line of the list file to be executed. The TextArea object is also responsible for inputting the file to be displayed. [3,13] As the file is input, information is extracted and stored to develop a model of the current state of the HC11.

The Program class is an abstraction of the HC11 program being debugged. It stores information about the HC11 program that has been loaded for simulation. The program object is the central link between all other objects. It must maintain a data structure that links a line number in the TextArea object with an HC11 memory address. This link is utilized when a Jump or Branch instruction is encountered. The opcode instruction contains an address of the memory location to jump to, at which point the simulator must jump to the corresponding line in the list file and display to the user that it is the next statement to execute. This is accomplished with a Hash Table object that uses the address as the key and a Statement object as the value. The program object also maintains an array of the HC11's memory contents.

The Statement class is used to organize each of the list file's statements. The statement object breaks a line down to its relevant components such as the HC11's corresponding memory address, the opcode, the data and the line number.

The HC class holds the abstraction for the HC11 object. This includes single byte accumulators A, B, and D, and the Condition Code Register (CCR), and double byte registers X, Y, the Program Counter (PC) and the Stack Pointer (SP).[2,6,8,16] The SP is a virtual stack that represents the contents of the actual stack. The HC class also includes all the operations/methods needed to simulate an actual HC11. Included in the HC11 class is a "memory table" which is used to display the current memory contents of the HC11. The table object uses the program's memory array to fill in the table. The HC11 classes as well as the majority of the classes contain a utility class which performs basic operations such as converting a hexadecimal string to a decimal or binary value.

The Opcode class contains all the implementations of the HC11's operations. A "fetch" call is made from the HC11 class, which determines the appropriate opcode function to be executed, calls/executes the function, and updates the pertinent components of the HC11 class.

The description of the software implementation of CET11SIM is also presented in a flow chart format in Figure 6.

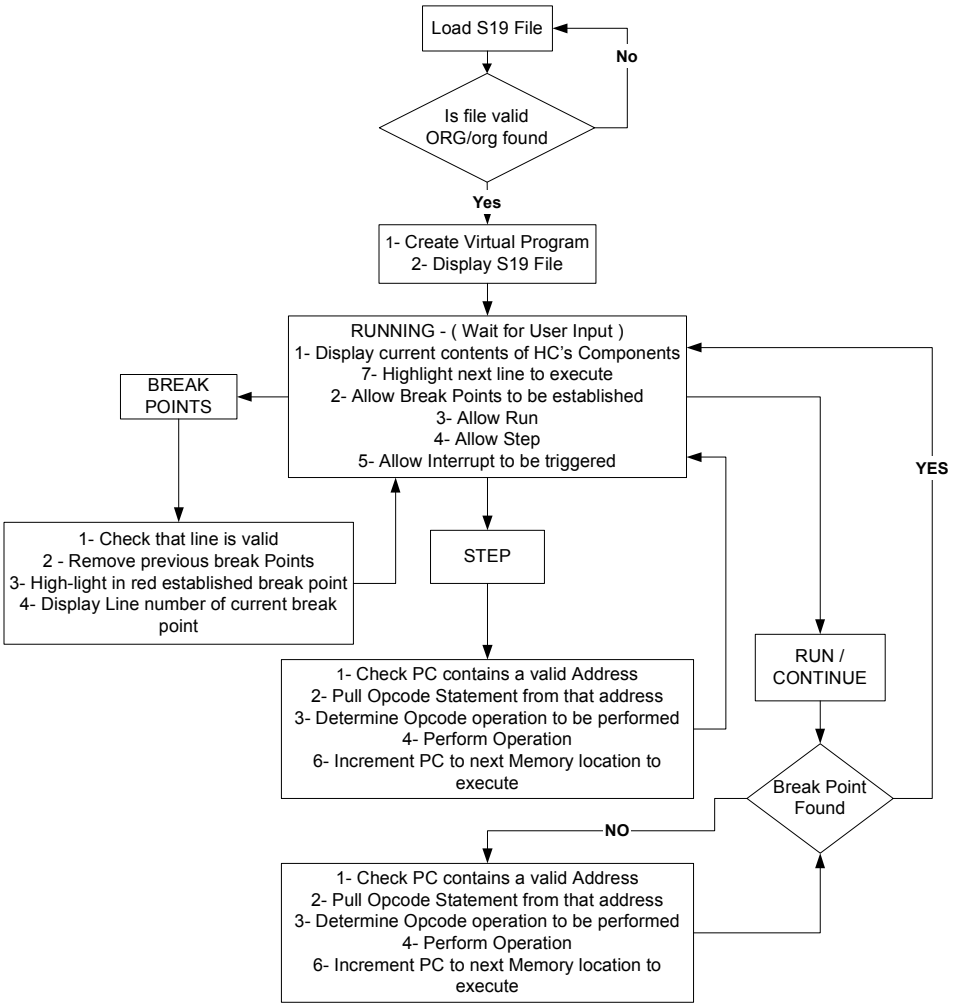


Figure 6. Software Structure of CET11SIM

IV. The Microprocessor/Microcontroller Related Course Implementations

The implementation of the CETHC11EVB2 and CETHC11SIM can be adapter to different levels of the microprocessor/microcontroller courses. By requiring students to own the EVB system board, they are required to build it from components to a complete assembled functional board. It is not only an interesting and challenging task at the beginning of the entry level class, but it also enhances student understanding of the system in future classes. Since each student processes his/her own board, there is no borrowing/lending of the system board, and the maintenance expenses are nonexistent. Additionally, the simulator software can be simply downloaded from the department's server computer. Table 1 summarizes various exercises of hardware and software implementation in different level classes.[6,10]

uP/uC Class Level	Related Topics/ Course Contents	CETHC11EVB2 / CETHC11SIM
Entry	Assembly Programming, Addressing Mode, Hardware Architecture, Subroutines, Stacks, Basic Math Routines, Simple I/O Controls, Interrupts	CETHC11EVB2 & CETHC11SIM
Medium	Advanced I/O Controls, Different Number Systems/Codes Conversations, Timer Functions, ADC Controls, Stepper Motors Controls, DC Motors Controls, Parallel Interface, Serial Interface (SPI, SCI, Bit Banning), Display Units (LCD), Keypad, Card Reader, Communications	CETHC11EVB2 & CETHC11SIM
High	Parallel and Serial Communication Protocols, Multiple Processor Communications, HC11 to PC Interfacing, 16/32 bit Precision Multiple and Divide Routines, External Serial Memories Interfacing/Storage, Wireless Communications, System Integrations/Designs	CETHC11EVB2
Project	Combine of All the Topics to a Useful Application Project	CETHC11EVB2

Table 1. The Related Courses with CETHC11EVB2 and CETHC11SIM

Certainly, there are no practical limitations on the use of this hardware and software. If there are limited numbers of classes' available, topics can be combined into two courses, or some subjects can be eliminated.[6]

V. Conclusion

Although the development of a new microprocessor/microcontroller educational system was a challenging task to the instructor, it is a worthwhile effort and pays off in the future for years to come. It has several attributes that make it attractive to students, administrators, and instructors.

From the students' point of view, the requirement to purchase all the components and assemble, solder, and test of the final system board is a valuable learning experience. The soldering experience is a fringe benefit that students do not normally receive as part of an engineering technology program. It is impractical to offer one particular course to teach student soldering and de-soldering skills but it is a common practice in real world in either

design or application. This assembling/trouble shooting experience provides them a valuable lesson and they are determined to be successful. This is a perfect example of competence based learning experience and students are absolutely motivated. Instructional materials are widely available to the student through Motorola's web site and instructors' hand outs. Additionally, requiring students to pay for the parts and the system board ($80.00) and using reference books instead of a required text book was preferable to the students.

From the administrators' point of view, there is only the initial cost in the development stage of the system board and simulator. After the design and development is finalized, there are no additional costs such as maintenance or replacement of any of the system boards. Since each student owns his/her own system board, it is guaranteed that the boards will be treated with care. Even if there is anything malfunctions in the system board, having previous trouble shooting experiences enables them to fix the problem and the cost is just the component expense. At the departmental level, there are no additional costs except for the basic lab instrumentation.

From the instructors' point of view, it is relieved from the task of maintaining and repairing system boards, and their associated costs. This also gives instructor flexibility in implementing lecture and lab material that is based on the manufacturer's data sheets, not a textbook. The simulation software is an excellent teaching aid that eliminates many mistakes and associated trouble shooting time before the student actually runs his/her software on the CETHC11EVB2.

After two years of implementing this hardware and software system, feedback and comments from the students who use it in their microprocessor/microcontroller courses is totally positive. It is a great access to the students who use it in their senior project designs. With this system board, they can basically do their design work anytime and anyplace. Additionally, it has been observed that students do not sell the system board after they have completed their classes or degree. It seems like a proud trophy that students want to keep for years to come.

Having students go through the process of building the system board and using the simulator, has triggered students interest in how the board operates, and the function of the codes in the simulator. In those microprocessor/microcontroller classes, there are interesting topics students asked about the system that they built and software they used, which provide the instructor with additional ideas for project, experiments, and teaching methods. This integration provides interesting concepts that offer students a better understanding of the links between hardware and software and the potential microcontroller applications at future workplaces.

VI. Bibliography

1. Axiom Manufacturing CME-11E9-EVBU MC68HC11 Low Cost Development System, Available: http://www.axman.com/, January, 2006.
2. Cady, Frederick, M. and Sibigtroth, James M. *Software and Hardware Engineering,Motorola M68HC11.* Oxford, NY: Oxford University Press, 1997.
3. Cay, Horstmann, S. *Core Java 2, Volume 2 – Advanced Features.* Englewood Cliffs, NJ: Prentice-Hall, 2001.
4. EVBU, Available: http://www.programmersheaven.com/zone5/cat26/31763.htm, September, 2005
5. Freescale, Inc., Available: http://www.freescale.com/webapp/sps/site/ homepage.jsp?nodeId=06, January,

2006.
6. Gendrachi, Thomas. “Teaching Effective Troubleshooting in the Microprocessor Lab”. *Proceedings of the 2004 ASEE Annual Conference and Exposition*, Salt Lake City, Utah, June 20-23, 2004.
7. M68HC11 Programming Reference Guide. *MC68HC11EGR/AD*. Motorola, Inc., 1991
8. M68HC11 Reference Manual. *MC68HC11RM/AD Rev 3*. Motorola, Inc. 1991.
9. M68HC11 Evaluation Borad User’s Manual. *M68HC11EVB/D1*. Motorola, Inc, 1986.
10. Mastronardi, Andrew, Montanez , Eduardo. “Microcontrollers in Education: Embedded Control Everywhere and Everyday”. *Proceedings of the 2005 ASEE Annual Conference and Exposition*, Portland, Oregon, June 12-15, 2005.
11. Miller, Gene H. *Microcomputer Engineering*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
12. MiniIDE, Available: http://www.mgtek.com/miniide/, January, 2006.
13. Sun Java website, Available: http://www.java.sun.com
14. Tekmos, Available: http://www.tekmos.com/standard_products/TK68HC24.htm, January, 2006.
15. THRSim11, Available: http://www.hc11.demon.nl/thrsim11/info.htm, September, 2005
16. Wray, W, Greenfield R., and Bannatyne R. *Using Microprocessors and Microcomputers, the Motorola Family*. Fourth Edition, Englewood Cliffs, NJ: Prentice-Hall, 1999.

VII. Biography