A PSD Based HC11 Microcontroller Development Board: Building on Past Experience

Andrej Nussdorfer, Tadej Tuma, Iztok Fajfar, Janez Puhan, Arpad Bürmen
University of Ljubljana
Faculty of Electrical Engineering
Tržaška 25, 1000 Ljubljana, Slovenia
Phone: +38614768724
E-Mail: andrej.nussdorfer@fe.uni-lj.si

1. Abstract

The success of the educational process of teaching microcontroller programming is greatly improved by the possibility of the students having individual access to the prototyping system and experimenting with it. While low cost microcontroller development boards are widely available, they do not offer the possibility of debugging the code directly on the target system. The usual approach to this problem would be a prototyping board and a processor emulator. During the past decade we have employed two different strategies to overcome this problem. In this paper we present the sum of our past experiences and discuss a new approach that we intend to use in the future.

2. Introduction

A modern microcontroller programming workstation consists of a PC, a microcontroller prototyping board, a processor emulator to debug the code, and respective software.

While this combination offers a reasonable price/flexibility ratio it is usually too expensive to equip a large laboratory with, especially when considering the lifetimes of the processor emulator systems, which are considerably shortened because of long operation times and student mishandling.

This and the complexity of the commercial processor emulators led to the development of a custom development system based on the popular 6803 microcontroller [1,2]. The first system was developed back in 1993 and it served its purpose well up to 1999, when it was replaced with a modernized system based on the MC68HC11 microcontroller [3]. Both systems are available to the students that want to have their own development board. While the latter system is still relatively new, we are already researching a possible substitute, based on the MC68HC11 and Waferscale PSDs . In this paper we outline the planned capabilities of the researched system. For the discussion to be more clear we briefly describe both existent systems.

3. The early work

The first system was developed back in 1993 (figure 3)[1,2]. A "virtual processor" approach is used to execute single instructions. The basic philosophy of this approach is a "simulation RAM". This is a small piece of RAM where the instruction to be stepped is copied along with all its operands. A jump back to the BIOS is placed after the instruction so that the BIOS can copy the contents of all the CPU registers in RAM.

This way, almost all the instructions can be executed and then the contents of the CPU dumped, with the exception of jumps and branches, which are simulated by the BIOS. While it does not feature a true RUN/STOP mode, this system can "animate" the program, sequentially executing as many instructions as we like, displaying the contents of the CPU registers after each executed instruction.

A very useful feature of this system is its ability to handle external interrupts, which are serviced in real time without hindering the BIOS in any way. The user communicates with the BIOS via a simple VT-100 terminal (figure 1), thus enabling almost any computer with an RS-232 port to be used in the debugging process.

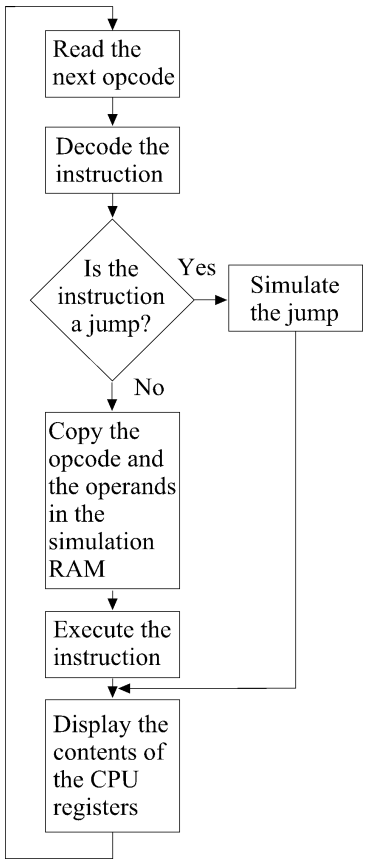

Figure 1: The early system – block diagram of the stepping algorithm

The instruction set of the debugger is very simple, only 4 instructions are used to debug the code:

T[num] sequentially executes num instructions of code, tracing into subroutines. All the CPU registers are dumped on the screen after every executed instruction

P[num] sequentially executes num instructions but steps over subroutines, which run at the full CPU speed. The contents of the CPU registers are also dumped on the screen after every instruction

R[num] behaves like P[num], but the CPU registers are dumped only after the last instruction is executed

D[addr] dumps on the screen the contents of 256 bytes of RAM after the addr address

This system proved extremely efficient, considering that no additional hardware is needed to debug the code, only approximately 3kB of memory space is used. It was also very well accepted by the students, who showed a much improved interest in the field of microcontroller programming, which is probably the best indicator of its usefulness.

```
.T1
E490:BD(JSR   ) E4 A2 T=0015 >> A=C3 B=37 X=E000 S=3FFD P=E4A2 F=H0N000
.P3
E4A2:86(LDAA #) 14    T=0017 >> A=14 B=37 X=E000 S=3FFD O=E4A4 F=H00000
E4A4:CE(LDX  #) 20 6F T=001A >> A=14 B=37 X=206F S=3FFD O=E4A7 F=H00000
E4A4:CE(JSR   ) E3 B8 T=0020 >> Dobro jutro!! A=14 B=0D X=206F S=3FFD O=E4AA F=H00000
.D2060
2060 : 66 72 6F 21 21 0D 20 6A 75 74 20 66 F7 FF FF 44 = fro!!. jut f...D
2070 : 6F 62 72 6F 20 6A 75 74 72 6F 21 21 00 F7 BF FF = obro jutro!!....
2080 : FF EF FF FB 49 42 0D FB FF 7F FF DF F9 3B 20 FF = ....IB.......; .
2090 : FF FF FF FF AB 69 65 FF FF FF FF FF 00 1A D8 FF = .....ie.........
```

Figure 2: An example of the communication between the user and the development system BIOS

4. An improved system

The first system has some clear advantages, but it shows some minor flaws too, especially the awkward user interface. Thus we decided that a new system should be developed, using a simpler user interface running in a Windows 95 environment (figure 2) [3]. Apart from the implementation of a new user interface, another method of tracing the code is used, too. A more common approach using SWI instructions as breakpoints is implemented.

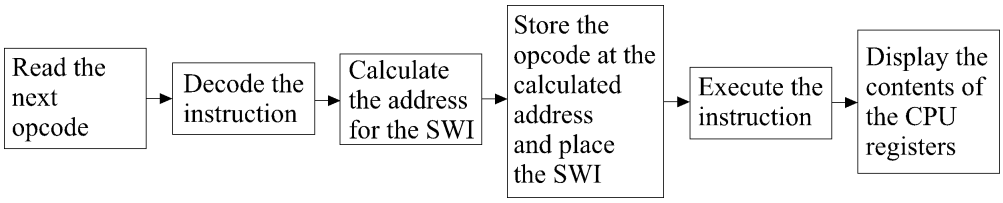

Figure 3: The current system – block diagram of the stepping algorithm

This approach has the advantage of not needing to copy anything. While the earlier system has to copy an instruction to the simulation RAM, with this method only a SWI is placed behind the instruction to be stepped and the program counter adjusted to point to that instruction. The SWI then handles everything else – it dumps all of the CPU registers on the stack, jumps to the service routine which acts as BIOS, and after setting another SWI breakpoint and some tweaking of the stack it returns to the next instruction to be stepped. Another advantage of the system is that before a breakpoint the program runs at the full CPU speed. This system also features a true RUN/STOP mode. The RUN is accomplished by clearing all breakpoints (SWIs) and letting the CPU to execute the first instruction of the user code. The STOP is implemented with an XIRQ interrupt.

Like the earlier, this system was also welcomed by the students, especially because of the easy to use interface. At the same time some feedback came from the industry too, where some former students used the microcontroller boards as real-time control systems. It was there that the development system showed its most severe shortcoming, the inability to trace a program with interrupts enabled. Since this system was developed
primarily for educational purposes this shortcoming wasn't considered critical.

Another shortcoming of the system is the method of tracing it utilizes. A disadvantage of this approach is that the code to be traced must reside in RAM. Once the code is burned in read-only memory, the system is not able to trace it anymore.

5. Upgrading the hardware

The reason for such a prompt replacement of the development board lies in the complexity of the board itself and

the appearance of PSD9xx and PSD8xx devices by Waferscale (now a division of ST). The board was designed with cost-effectiveness in mind so standard LSI logic was used instead of programmable devices like PALs. While the board is still relatively simple compared to professional microcontroller boards, it is still quite complex for the average student to build.

in FLASH, as is the case when using the PSD chips. This forced us to redesign the software too. Since we already had two working development systems, a comparison between them seemed the most reasonable thing to do first. Here is the comparison:

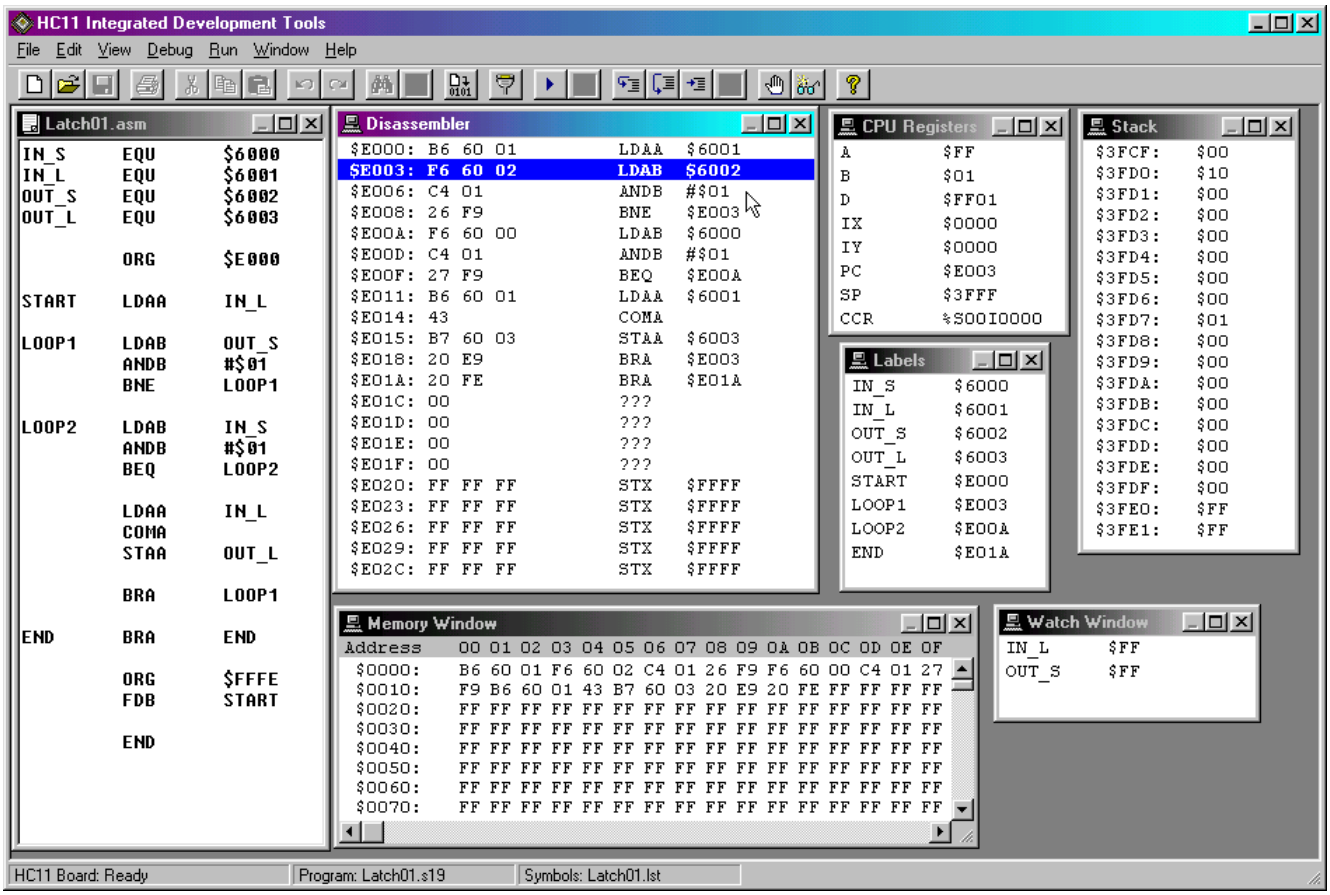

Figure 4: A screenshot of the current system user interface

A large number of components poses a serious problem for the inexperienced students that want to build their own boards. More components also mean more parts that can break, making it more difficult to repair the boards. That, combined with the introduction of the advanced PSD chips by ST, made us consider the possibility of designing a new development board. With the implementation of the PSD chips the development board could shrink to half the size of the current board with virtually no components besides the M68HC11, the PSD chip, the voltage regulator, and the RS232 level shifter. And less components also mean better reliability.

6. Playing mother nature

As already mentioned, the current development system dynamically replaces instructions with SWIs to achieve single stepping. While this system works fine as long as the code to be traced is stored in RAM, it fails completely if the code is stored

The early system:

Advantages:

- all the overhead is executed on the board itself, minimizing the needed communication between the board and the host PC
- it is possible to single-step code resident in read-only memory, like FLASH or EPROM
- this system allows interrupts to occur without hindering the tracing algorithm
- the interrupt subroutines run at the full CPU speed
- so do normal subroutines when stepped over

Disadvantages:

- the BIOS is quite large (approximately 3kB)
- the 68HC11 internal RAM is used by the BIOS
- the user interface is somewhat awkward, because of the use of a standard VT-100 terminal mode
- breakpoints are not supported very well

- it cannot execute a number of instructions at full speed and then halt

The current system

Advantages:
- the BIOS on the board is very compact, since all the overhead is executed on the host PC
- the user interface is easy to use
- it has a very efficient system of breakpoints
- before encountering a breakpoint the code runs at the full CPU speed
- it features a robust communication protocol between the development board and the host PC

Disadvantages:
- the breakpoints are implemented with SWI instructions, effectively disabling the system to trace a program with interrupts enabled
- the code to be traced must reside in RAM

After analyzing the characteristics of both existing development systems we decided that a cross-breeding of both systems would yield an efficient and user friendly development system. We decided to use the graphic user interface and communication protocol from the current system and the stepping strategy from the early system. Thus the characteristics of the proposed system are:

- the development board is very compact
- the use of a PSD device greatly improves the functionality of the board while minimizing the required number of components
- there is up to 288kB of FLASH RAM for the code and 8kB of SRAM
- the BIOS on the development board is very compact
- all the required overhead is kept on the host PC
- the user interface is easy to use even for inexperienced students
- a robust communication protocol between the board and the host PC
- all the communication between the host PC and the board is carried via the standard RS-232 port
- the serial communication protocol enables remote debugging
- code residing in read-only memory is traceable
- interrupts are allowed
- interrupt routines are executed at the full CPU speed
- normal subroutines are also executed at full speed when stepped over

Clearly, some of the biggest advantages of the current system are lost due to the changed stepping algorithm, but the added functionality, especially the availability of interrupts and the ability to trace code in read-only memory, largely overweight the loss. As the main reason for the researching a new system lied in the inability of the current system to trace code residing in FLASH, the possibility to trace code with interrupts enabled encouraged us even more. Since all the communication between the board and the host PC is carried over a serial channel it would be possible to extend the distance between them by sending and receiving data over the IP. Although it would present some major security[4] and reliability problems it would be interesting nonetheless.

The system thus obtained will not only be applicable to development boards using FLASH program memory, but will also be useful for professional code developers working on real time problems often including interrupts.

7. Conclusion

The research into a new development system was prompted by the release of the PSD devices. Since we are always on the lookout for new technologies, taking such a step was inevitable. We believe that the proposed system is more than a suitable replacement for the existing one. While first meant only as an educational tool, with the described characteristics it could also be an efficient and cost effective development system for professional users. Already the first two development systems found their way out of the microcontroller classes into real-world applications. With the experience gained over the past years we are quite confident that the new system will be greeted with at least the same enthusiasm.

8. References

[1] Tadej Tuma, Franc Bratkovič, Iztok Fajfar and Janez Puhan, "A Microcontroller Laboratory for Electrical Engineering", Int. J. Engng Ed. Vol. 14, No. 4, p. 289-293, 1998

[2] Tadej Tuma, Iztok Fajfar, Janez Puhan, "The Principle of Vaccination in Teaching", Frontiers in Education Conference '99, San Juan, Puerto Rico, 1999

[3] Tadej Tuma, Iztok Fajfar, Janez Puhan, Arpad Bűrmen, "A Non-Virtual Distance Education Course in Software Engineering", International Conference on Information Technology Based Higher Education and Training, Istanbul, Turkey, p. 287-292, July 2000

[4] Sašo Tomažič, "Secure communications over the Internet", The European journal of teleworking, Vol. 4, no. 1 (Spring 1996), p. 6-13