A Case Study in Hardware-Software Co-Design of EPROM Emulators

Zdravko Karakehayov
Technical University of Sofia
zgk@computer.org

Emil Saramov
Technical University of Sofia
egs@vmei.acad.bg

ABSTRACT

We present a case study in hardware-software co-design of an EPROM emulator. The emulator is oriented towards 8051-compatible targets. Two emulator architectures are considered. In case of a microcontroller architecture for the emulator, a target microcontroller running at 1 MHz will need an emulator microcontroller with oscillator frequency 20 MHz. A successful emulator design requires hardware-software co-design approach and a microcontroller plus ASIC architecture is the only solution which provides sufficient application range. We break down the ASIC architecture into data and control. Along with the data path - controller hardware we introduce a communication channel which links both microcontrollers. The ASIC has been implemented by two FPGAs ispLSI 1016 produced by Lattice. We used a hardware description language called ABEL as a design entry language for both FPGA circuits. The Synario CAD tool was employed for logic optimization, mapping and routing.

Keywords *- Embedded systems, debugging tools, microcontrollers.*

1. Introduction

Embedded systems design has been a hot topic for the past several years. There are two important trade-offs which underlie the design of embedded systems. First, the designers have to strike the balance between the target system parameters and the design process in terms of resources. Unfortunately, our efforts to optimize the design object, the embedded system, result in longer and more expensive development cycles. The second trade-off comes with the borderline which the design process puts between the embedded system's hardware and software. The hardware-software trade-off has a significant impact on the system performance. A good starting point for the emulator case study can be Figure 1 which presents the overall system architecture viewed as a distributed system. The embedded system under consideration, the emulator, is a single processor system. So is the system that will be tested, the target system.

The EPROM emulator can be used when the target system has one or more EPROMs. We remove an EPROM from the target and plug in the emulator cable. Thus, the emulator RAM substitutes the target EPROM. The personal computer downloads the code to the RAM.

The emulator single chip microcomputer controls the debugging process according to the instructions from the PC.

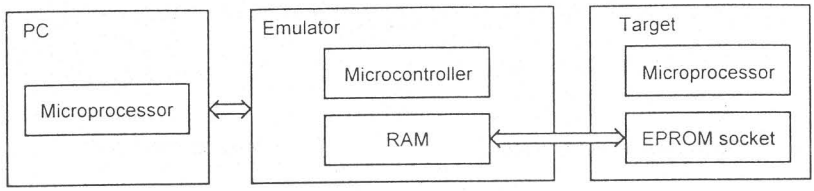

Figure 1 The EPROM emulator in conjunction with a PC and a target system

Figure 2 shows the EPROM emulator functionality broken into six subtasks. The subtask "Set breakpoints" marks a certain number of addresses. When the target system reaches a breakpoint address, the execution of the user program is suspended and a program called Monitor is activated. The Monitor controls the communication between the PC and the target system. Consequently, the emulator emerges with two extra RAMs as shown in Figure 3. Essentially, RAMs are addressed in parallel.

The three-RAM architecture is motivated by the following considerations. The simplest manner to organize a breakpoint is to replace the pattern of a certain instruction with a jump to the Monitor. Inevitably, this solution imposes a limitation. The breakpoint instruction must be at least two bytes long. As far as the 8051 family is concerned, the method is not feasible for 44% of the instructions. We overcome the problem by introducing the breakpoint RAM. In this case, we still need to carry out the jump to the Monitor. The second extra RAM accommodates the Monitor (RAM MON). Of course, we could cut down the emulator architecture to two RAMs and place the Monitor in the user RAM. However, this will limit the accessible size of the user program.

Apparently, the subtask "Stop the user program on a breakpoint" requires the fastest reaction which will be a challenge for the selected architecture. The design flow goes on with the following assumption: the target is based on an 8051 microcontroller.

When the target microcontroller runs a user program, the emulator checks the breakpoint bit BP from the BP RAM. As far as the bit BP is not set, the emulator moves the code from the USER RAM to the EPROM socket. When a set bit BP occurs, the emulator emits the three bytes of the instruction **LCALL** to the EPROM socket. The destination address of this instruction is an entry point in the Monitor. Thus, the execution of the user program is suspended and the target microcontroller runs the Monitor program. Beginning now, the emulator redirects the code from the MON RAM to the EPROM socket.

Naturally, the emulator will make a decision of whether to insert a breakpoint or to continue the execution of the user program within a certain period of time which we call reaction time (t_R). The reaction time determines a target oscillator frequency, which must not be exceeded.

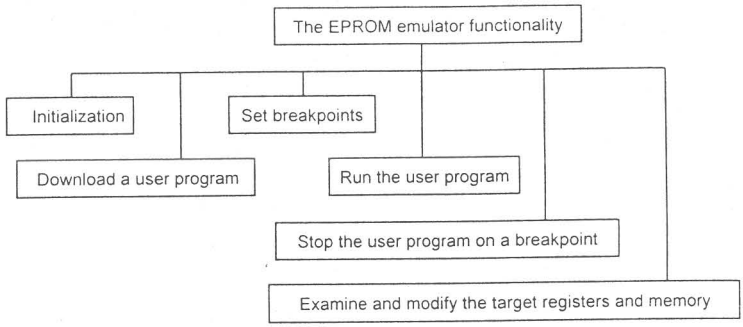

Figure 2 The EPROM emulator functionality shown in subtasks

The emulator microcontroller must test the signals $\overline{CE}$,$\overline{OE}$ and BP. Also, the change of the address line A0 is taken into account. The emulator must know if the signal A0 has just been altered. In this way, the design will be consistent with the 8051 feature to read bytes from the Program Memory ahead.

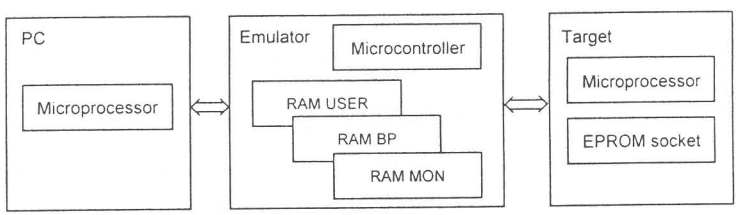

Figure 3 The EPROM emulator with two extra RAMs

Unfortunately, there is another case when it seems impossible to find a solution at a reasonable price. Assume that we would like to insert a breakpoint immediately after a **JB** (Jump if bit) instruction. This instruction performs a redundant read from the first address of the following instruction. The code is ignored, but the breakpoint scheme is activated regardless of the result produced by the conditional jump. As a consequence for our emulator: the user is not allowed to insert breakpoints immediately after the program control instructions. The user could work around this problem by adding **NOP** instructions where necessary.

The emulator microcontroller is selected to be a member of the 8051 family.

We focus on two architectures for the EPROM emulator: a microcontroller architecture and a microcontroller plus ASIC architecture.

2. A microcontroller architecture

We assume that **JB** (**JNB**) instructions will be used to test the signals $\overline{CE}$,$\overline{OE}$ and BP. Thus, the reaction time becomes 8 cycles (96 oscillator periods). When we

put a few calculations into a table, the correspondence between the target and emulator clocks immediately become obvious (Figure 4). For instance, a target microcontroller running at 1 MHz will require an emulator microcontroller with oscillator frequency 20 MHz. The calculations are based on BP RAM access time 50 ns and a correction of 20 ns which covers the delay introduced from buffers, the cable and the decoding circuit in the target.

Target oscillator frequency f_{OSC} (MHz)	0.2	0.4	0.6	0.8	1.0	1.2
$T_{OSC} = 1 / f_{OSC}$ (ns)	5000	2500	1666.7	1250	1000	833.3
The reaction time in case of the 80CL31 target microcontroller $t_R = 5T_{OSC} - 115 - t_{ACC}^{RAM\_BP} - 20$ (ns) $t_{ACC}^{RAM\_BP} = 50$ ns	24815	12315	8148	6065	4815	3981
The emulator microcontroller oscillator frequency $f_{OSC\_E} = 96 / t_R$ (MHz)	3.869	7.796	11.783	15.829	19.938	24.115

Figure 4 The correspondence between the target and emulator microcontroller clocks

Furthermore, Figure 5 illustrates the design process when the subtask "Stop the user program on a breakpoint" is moved from software to hardware. The transition point is oscillator frequency of 1 MHz. The hardware implementation of this time critical subtask would demand an ASIC.

Two different target microcontrollers are involved in this example - 80CL31 and 80C31. Figure 5 indicates the difference in the timing parameter TAVIV (Address to valid instruction in), which may vary from one device to another. The period of time TAVIV is used in the equation for the reaction time t_R (Figure 4).

While the emulator architecture based on an 87C51 microcontroller with a 24 MHz crystal is suitable for target systems running up to approximately 1 MHz, a microcontroller plus ASIC architecture expands the application range up to 36 MHz, if the ASIC propagation time is 23 ns and the BP RAM access time is 40 ns.

The overall conclusion is that the EPROM emulator design requires a hardware-software co-design approach and a microcontroller plus ASIC architecture is the only solution which provides sufficient application range.

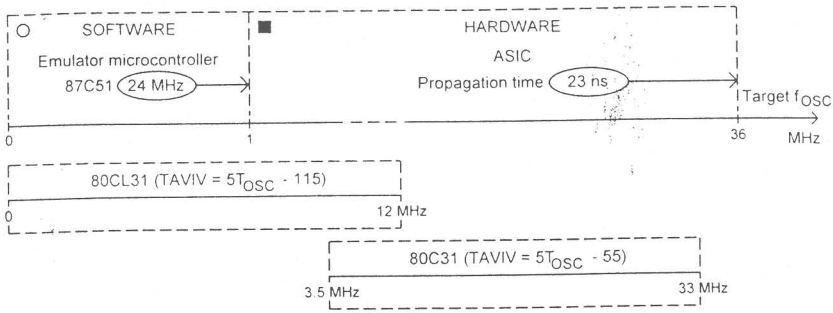

Figure 5 The implementation of the subtask "Stop the user program on a breakpoint"

3. A microcontroller plus ASIC architecture

Figure 6 outlines a microcontroller plus an ASIC architecture. The picture concentrates on the address and data paths. In order to distinguish between the emulator and the target signals we add a prefix "Target" or "T" to all EPROM I/Os. The target microcontroller reset input is named TRST as well.

As can be seen in Figure 6, an ASIC links the target with the emulator microcontroller and the RAMs. The ASIC passes the code from the user RAM to the target and the user program is executed. In parallel, the breakpoint RAM emits the BP bit. When a set BP bit occurs, the ASIC generates the three bytes long instruction **LCALL**. From then on the ASIC conveys the code from the Monitor RAM to the target.

4. An ASIC implementation

The ASIC architecture has been split into data and control. Along with the data path - controller hardware we introduce a communication channel which links both microcontrollers. The ASIC has been implemented by two FPGAs ispLSI 1016 produced by Lattice. We used a hardware description language called ABEL as a design entry language for both FPGA circuits. The Synario CAD tool was employed for logic optimization, mapping and routing.

5. Conclusions

Precise system specifications are often not provided at the beginning of the design process. The conceptualization used in the beginning of the design cycle gradually matured in rigorous specification. We moved a task from software to hardware and from then on continued the project by concurrent design of the hardware and software parts. The ASIC implementation based on in-circuit programmable FPGAs demonstrates an efficient design of a small-scale embedded system. The FPGA not only provides the required timing parameters, but also allows the hardware to be reconfigured for different target processors.

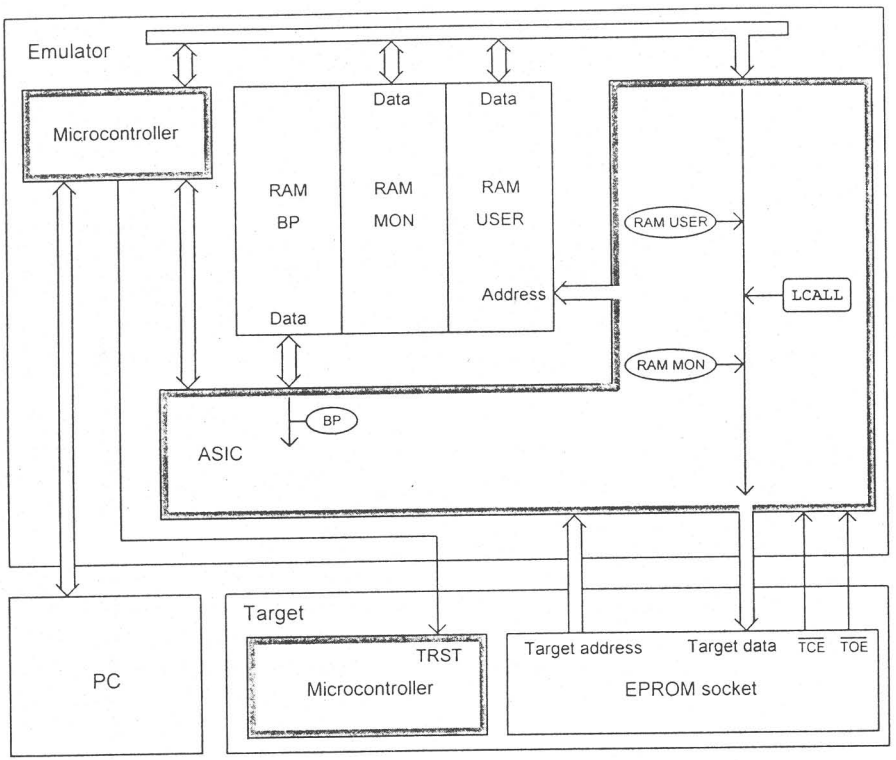

Figure 6 The EPROM emulator architecture in outline

6. Acknowledgements

We are grateful to Prof. Knud Smed Christensen of the Technical University of Denmark who proposed the three-RAM architecture of the EPROM emulator for this project.

7. References

[1] Data I/O Corporation, *SYNARIO, ABEL-HDL Reference*, 1996.

[2] Lattice Semiconductor, *Data Book*, 1994.

[3] David Pellerin and Michael Holley, *Practical Design Using Programmable Logic*, Prentice Hall, 1991.

[4] Philips Semiconductors, *80C51-Based 8-Bit Microcontrollers, Data Handbook IC20*, 1997.

[5] Wayne Wolf, *Modern VLSI Design*, Prentice Hall, 1994.

[6] Wayne Wolf, "Hardware-software co-design of embedded systems", *Proc. IEEE*, vol. 82, No. 7, July 1994, pp. 967-989.