EP32 - a 32-bit Foth Micorprocessor

Edvin Hjrtland
Electrical and Computer Engineering Department
South Dakota School of Mines and Technology
Rapid City, U.S.A
ehjortland@gmail.com

Li Chen
Electrical and Computer Engineering Department
University of Saskatchewan
Saskatoon, Canada
li.chen@usask.ca

***Abstract*—This paper introduces a design of a 32-bit Forth microprocessor – EP32. It is a stack based processor, well suited for the Forth programming language. The instruction set includes 35 RISC-like instructions. The EP32 processor has two stacks: the data stack and the return stack which are 256 levels deep. The 32-bit Forth processor is designed in VHDL, and synthesized with a 0.18 µm standard CMOS library. The processor will be synthesized with a radiation-hard-by-design digital library in the future for space applications.**

Keywords-forth, microprocessor, stack-based computer

I. INTRODUCTION

Forth is used widely for programming embedded systems because of its simplicity and efficiency [1]. It explicitly manipulates data on a stack, and so defines a simple virtual machine architecture which makes programs independent of the CPU - only the interpreter needs to be ported [2]. Because of this, extra CPU features are wasted when running Forth programs, and since cost reduction is important to embedded systems, it is logical to want a simpler, cheaper CPU which runs only Forth programs.

Forth processors use the minimum instruction set (MISC Instruction Set), which is similar to RISC. But it has even simpler instruction set, which usually has less than 32 instructions [3]. The well-known MuP21 [4] is a 20 bit CPU which has 25 5-bit instructions and implemented in 1.2 micron CMOS process, uses only 7000 CMOS transistors and has a peak execution rate of 100 MIPS. A new configurable 32-bit forth processor - EP32 is introduced to target for the applications in communication, high-end electronic games, and embedded control, where low power consumption and high execution speed are essential [5, 6].

EP32 uses an RISC-like instruction set with 35 machine instructions. Its architecture has 35 6-bit instructions. Five 6-bit instructions are packed into one 32-bit word, and are executed consecutively after a word is fetched from memory. It can be viewed as a 5-instruction cache that provides the optimal balance between the slow RAM and the fast CPU. EP32 has two stacks: the Parameter Stack and the Return Stack which are 256 levels deep. There are 31 primitive words created from 35 machine instructions.

This work is supported by South Dakota State and NASA Seed grant.

The paper is organized as follows. Section II gives a brief introduction to the instruction sets of the processor. Section III describes the architecture of the forth engine. The implementation of the design is presented in section IV. Evaluation results includes the chip layout are also presented in this section. Finally, section V concludes the paper.

II. INSTRUCTION SET

The CISC processors generally have 100 or more instructions. The RISC processors have about 50 instructions. It was obvious that 16 instructions are not sufficient to support all the necessary functions required in a microprocessor. 50 instructions are too many for a forth engine using MISC. With 6-bit instructions, it can provide a maximum of 64 instructions. EP32 has 35 6-bit instructions, which are listed in the Table 1. The instructions can be classed into four groups: transfer instructions for jump between addresses and subroutine operations, memory instructions for fetching the instructions from the memory, ALU instructions for arithmetic/logic operations on the operand, register instructions for data transfer between registers, and interrupt instructions for interrupt operations.

ADD instruction is implemented but not subtraction, XOR instead of OR instruction and OVER but not SWAP. Obviously, subtraction can be synthesized by compliment and addition. OR can be synthesized by compliment, AND, and XOR. OVER and SWAP are very similar, in that they allow accessing the top of the data stack. It is difficult to determine which is more fundamental in a stack machine.

Table 1: List of EP32 instructions.

Type of Instructions	**Name of Instructions**
Transfer Instructions	BRA, RET, BZ, BC, CALL, NXT, TIMES
Memory Instructions	LDC, LDRP, LDSP, LDI, LDX
ALU Instructions	RR8, STX, COM, SHL, SHR, XORR, ANDD, DIV, ADDD
Register Instructions	NOP, DROP, TX, PUSHR, OVER, DUP, XT, POPR, STC
Interrupt Instructions	EI, DI

III. ARCHITECTURE OF THE PROCESSOR

A. Genral Information

Fig. 1 shows a block diagram of the EP32 CPU core. The address bus contains the 32-bit address vector provided by the CPU for memory and other I/O devices. The 32-bit data bus is used for transferring data between the CPU, memory and other I/O devices. For the CPU to support external interrupts, a 5-bit interrupt vector is included. This way the CPU supports up to 31 different interrupt functions. The 6-bit icode vector contains the instruction code that is currently being executed. Since the EP32 has a 32-bit CPU core, the 4-bit byte_enable vector is included. The byte_enable vector keeps tracking of which the four bytes in the 32-bit word that are being processed. For example, if the first byte in the word is being processed, the byte_enable vector will be set to "0001". However, if the third byte is being processed, the byte_enable vector will contain "0100". If the whole word is being processed, all of the bits in byte_enable are simply set to '0'. The read and write signals are control signals to the memory. If the read signal is set high, the CPU will read from the memory. However, if the write signal is set 'high', the CPU will execute a write operation; hence the read and write signals can not be set to 'high' during the same clock cycle. The intack signal acknowledges an external interrupt, while the ack_o signal acknowledges the ready signal.

The EP32 has a set of six registers. An overview of the registers is shown in Table 2. All of the registers are 33 bits wide. The most significant bit of T register, T (32) is the carry produced by the 32-bit adder. The carry bit preserved as data in T is transferred to other registers and to the stacks. The preservation of carry bit greatly simplifies the logic processing of data, and allows interrupts to be serviced when the next program word is fetched from the memory, without having to save the carry bit and restore it on return.

Table 2: List of EP32 registers with a short function description.

Register	Description
X	Address register, supplying address for memory (read and write)
I	Instruction latch, holding instructions to be executed
P	Program counter, pointing to the next program word in memory
R	Top of return stack
S	Top of data stack
T	Accumulator for ALU

B. Architecture of EP32

EP32 has two 256-deep stacks: S stack for data and R stack for return addresses. A stack is a typical LIFO structure. LIFO stands for last in - first out, and refers to the way items stored in a data structure are processed. The last data to be added to the structure will be the first data to be removed. The data width of both stacks is 33-bit wide to preserve the carry bit produced by the ALU. The return stack is mainly used to preserve return addresses on subroutine calls, and store local variables. The data stack is used to pass parameters among the nested subroutine calls. With these two stacks in the CPU, EP32 is optimized to support the Forth programming language.

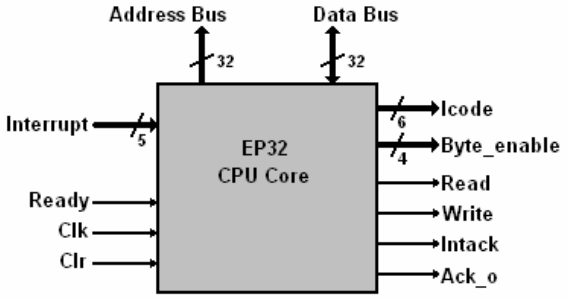


Fig. 1: Block diagram of EP32 CPU Core.

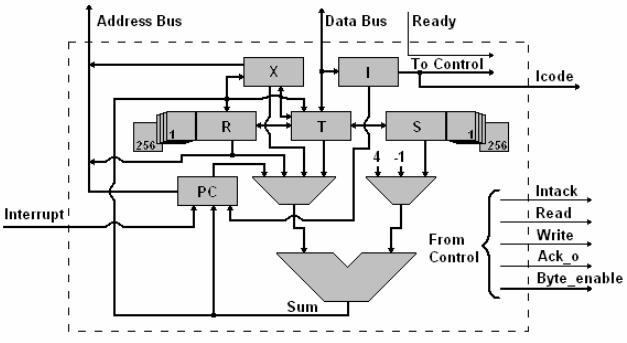


Fig. 2: Simplified dataflow chart of EP32 CPU core. The control unit and several multiplexers and control signals are not shown in the diagram.

EP32 provides a mini-RISC like instruction set of 35 instructions. Five 6-bit instructions are packed into one 32-bit word, and are executed consecutively after a word is fetched from memory. It functions as a 5-instruction cache which provides the optimal balance between the slow RAM and the fast CPU. 32-bit words are fetched from RAM at a rate of 20 MHz, and the 6-bit instructions are executed at a rate of 100 MIPS. A 6-bit instruction field can accommodate 64 different instructions. Since only 35 instructions are defined in the VHDL description, there will be many available instruction slots for future use if necessary. Fig. 2 shows a simplified data path diagram of the EP32 CPU core. The control unit and several multiplexers that select the signal going into various registers are not shown in the diagram.

The instruction decoding logic applies the proper control signals for registers and multiplexers. Table 3 shows some of the control signals which are directly affected by the control logic. Other select signals for various multiplexers are also affected, but these signals are not defined by name in the VHDL description. The synchronous program execution clocks the aslot signal, which selects the next 6-bit instruction in the I-register to be executed. Depending on the instruction, the proper value of the control signals in Table 2 will be generated. At rising clock edge the selected data are latched into the registers and stacks. All signals must be stabilized before the next rising clock edge. The slot0 signal acknowledges the processor when all the instructions in the I-register have been executed. Fig. 3 shows the relation between the master clock and the slot0 signal. When the aslot counter is 0, the slot0 signal is set high, and a new instruction word needs to be

fetched from memory. Instructions are being executed consequently until the aslot counter reaches 5. When the aslot counter reaches 5, it is reset to 0, and a new instruction word is fetched from memory. However, if the first instruction in the I-register is some kind of branch or jump instruction, the CPU will need to fetch a new instruction word from memory consequently. The relation between the master clock and the slot0 signal when a branch or jump instruction is executed is shown in Fig. 4.

IV. Implementation Methology and Implementation

The original VHDL description of the EP32 microprocessor targeted to FPGA applications was provided by Signal Processing and Microelectronics Branch, Goddard Space Flight Center, NASA. We modified and verified the design so that it can be implemented as an ASIC design. Testbenches were also developed for each level of the design. TSMC 0.18µm CMOS standard library is used for synthesizing and implementation. In this section, firstly we discuss the design of the stack, including simulation results. Then the implementation of the EP32 block is described, simulation results will also be included. Finally the layout of the chip is presented. All the tools that were used in this design flow are included in the ADK 3.1 (ASIC design kit) package from Mentor Graphics.

A. Implementation of Stacks

According to design recommendations of the LeonardoSpectrum user's manual, the VHDL description for the stack was revised to accommodate it. These recommendations suggest separating the code into smaller sub-blocks. This includes placing similar logic together, i.e., state machines, datapath logic, decoder logic etc. State machines should also be placed into separate blocks of hierarchy to speed up the optimization process, and provide more control over encoding. By following these rules, LeonardoSpectrum can more easily generate an efficient netlist. Both the R and S stacks are partitioned into stack cells, whose block diagram is shown in Fig. 5. The floorplan and connections of stack cells are shown In Fig. 6(a). The Layout of the stack are shown in Fig. 6(b). The physical layout was created with IC Station of Mentor Graphics. IC Station supports autoplacement and routing, which simplified the physical layout process.

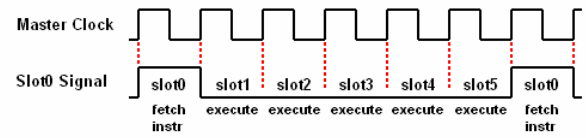


Fig. 3: Relation between the master clock and the slot0 signal of the EP32 when operating normally.

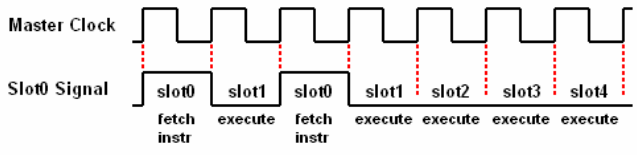


Fig. 4: Relation between master clock and slot0 signal when branch or jump instruction is executed.

Table 3: List of control signals of EP32.

Signal	Description
Rst	Master reset
t_load	If set, t_in is loaded into T register
r_load	If set, r_in is loaded into R register
x_load	If set, x_in is loaded into X register
i_load	If set, i_in is loaded into I register
p_load	If set, p_in is loaded into P register
Spopp	If set, pop the data stack
Spush	If set, push T onto the data stack
Rpopp	If set, pop the return stack
Rpush	If set, push R onto return stack
Aslot	Instruction slot counter. Counts from 0 to 5, depending on which instruction in I to be executed
adder_sel	2-bit vector selecting first input of the ALU
argument_sel	2-bit vector selecting second input of the ALU
slot0	If set, fetch new instruction word from memory

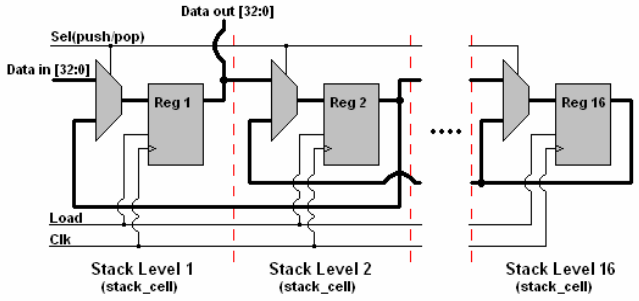
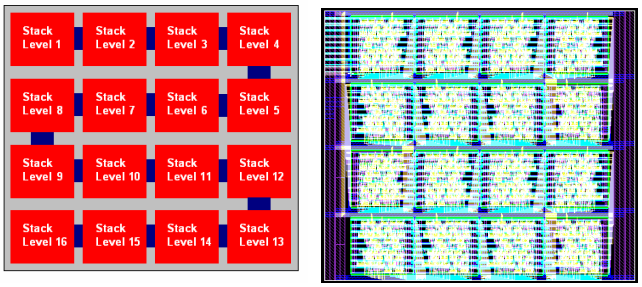


Fig. 5: Architecture of the stack design that are divided into simple stack cells. Both the return stack and the data stack are using this architecture.



(a) (b)

Fig. 6: (a) Floor plan of the stack cells. The blue fields between the stack cells indicate connection between the cells. (b) Layout of the stack.

B. Implementation of the EP32 Block

The synthesis process for the EP32 block is very similar to the synthesis process for the stacks. The VHDL description of the stack and the EP32 were loaded into the in-memory database of LeonardoSpectrum. The stacks are treated as black boxes. The EP32 file was analyzed, elaborated and pre-optimized. The in-memory database was optimized using the same technology as before, at a clock frequency of 100 MHz. The timing report indicates that the EP32 block can handle clock frequencies up to 113 MHz. This is a drastic drop in the frequency compared to that of the stack cells and whole stack. This is due to much longer combinational logic paths between registers in the control and datapath logic of EP32; compared to

the stack design, where only one multiplexer separates the registers. Fig. 7 shows the final layout of the EP32.

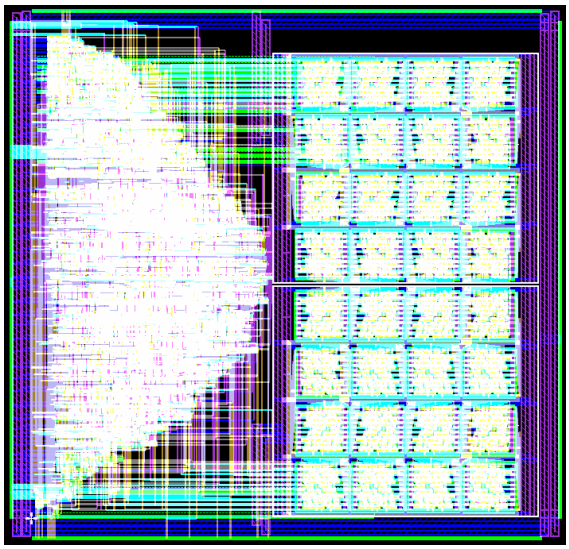


Fig.7: Physical layout of the EP32 block. The layout is build up by 81135 transistors, and the size is 1550x1500 µm.

C. *Verifications*

Functional verification is performed for each sub-block of the design. A testbench which includes all the instructions has been written in VHDL to verify the functionality of the EP32 CPU core. The testbench uses all the instructions at least one time; however, most of the instructions are used several times which improves the reliability of the testbench. The testbench has been used to do functional simulation as well as timing simulation for EP32.

The after-synthesis simulation of the EP32 block was executed by loading all the generated Verilog netlists into a project using Modelsim. The SDF-files were linked up to their respective components to include SDF timing information. The clock frequency is set to be 100 MHz. The design has been verified to be functional correctly.

V. CONCLUSIONS

This paper introduces a 32-bit forth microprocessor design, well suited for the Forth programming language. This Forth engine architecture provides high speed and high volume processing capability required for synthetic-aperture radar (SAR) signal processing and image processing. The design coded with VHDL has been successfully synthesized and implemented to an ASIC test chip with a commercial TSMC 0.18 µm CMOS technology. It is planed to design a radiation-hardened version of the EP32 CPU with a radiation-hardened library in the future.

ACKNOWLEDGMENT

The authors would like to thank Umesh Patel with Goddard Space Flight Center, NASA for providing the original design of the EP32 and the technical support for this project. We also would like to thank NASA and the South Dakota state for sponsoring this project.

REFERENCES

[1] E. D. Rather D. R. Colburn C. H. Moore, "The evolution of Forth," *The second ACM SIGPLAN conference on History of programming languages*, 1993, pp. 177 - 199.

[2] P. Koopman, Stack Computers: the New Wave, Ellis Horwood, 1989.

[3] P.H.W. Leong, P.K. Tsang & T.K. Lee, "A FPGA Based Forth Microprocessor," *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 254-255.

[4] C. Ting and C. H. Moore, "MPU21 – A High Performance MISC Processor," *Forth Dimension*, January 1995.

[5] R.K. Bardin, "Architectures for High-speed Processing", *Proceedings of the 1987 Rochester Forth Conference*, 1987, vol. 5, no. 1, pp. 83- 86.

[6] J.R. Hayes, M. Faeman, R. Williums, T. Zareman, "a 32-BIT Forth Microprocessor" *Proceedings of 1987 Rochester Forth Conference*, vol. 5, no. 1, 1987, pp. 39-48.