Is published in Advances in Computer Science for Engineering and Education II, ICCSEEA'2019, Springer, Vol. 938. P.238–246

Software/Hardware Co-design of the Microprocessor for the Serial Port Communications

Oleksii Molchanov[1], Maria Orlova[1] and Anatoliy Sergiyenko[1]

[1] Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine
aser@comsys.kpi.ua

Abstract. The eight-bit stack processor architecture is proposed, which is designed for the FPGA implementation. The microprocessor with this architecture has small hardware costs, reduced software amount, and ability to add up to hundred new user instructions to its instruction set. The microprocessor architecture is adapted for programming the serial port communications and is able to perform the data stream parsing.

Keywords: stack processor, Forth, FPGA, VHDL.

1 Introduction

In recent years, field programmable gate arrays (FPGAs) became a reasonable alternative to von Neumann architecture-based approaches in computer systems design. Increasing demand of performance and energy-efficiency is neatly fulfilled by capabilities that FPGAs propose.

A need to organize the data exchange through the interfaces such as I2C, SPI, Ethernet and others often occurs when a system on an FPGA is developed. At the same time, it is more rational to use the microprocessor core, which has both small hardware costs, and simple programming and debugging procedures. In addition, such a microprocessor can replace the finite state machines, which are needed for control of a designed system. RISC processors can be considered as those, which match described characteristics.

There is a small selection of universal RISC processors offered by FPGA manufacturers such as Xilinx Picoblaze, Microblaze, Altera Nios, or clones of common microcontrollers, such as i8051 [1–3]. But in many cases, the data exchange is performed using a simple protocol and at a relatively small speed, such as in the case of the I2C interface. In this situation, the capabilities of the RISC microprocessors are used inefficiently.

For the implementation of many application-specific systems in FPGA, it is important to have a configurable microcontroller with both minimized hardware and software. This is dictated by the fact that the memory blocks, which are embedded in FPGA, have significantly limited volume. It is desirable to have such a microcontroller which instruction set can be manually adjusted by the programmer to the needs of the project, to simplify programming, allow program subroutines reuse and, as a result, to minimize the program length. Its instruction set has to be adapted for scheduling the data transfer through the interfaces. Implementation of the architecture of such a microprocessor is the goal of this work.

This paper is organized as follows: section 2 gives overview of related works; section 3 describes architecture of the developed microprocessor; section 4 contains evaluation results; conclusion of this work is presented in section 4.

2 Related works

The are several related works.

In [4] authors work on 8-bit input/output processor for performing USB operations. It has the stack architecture and its instruction set consists of seventeen 14-bit instructions. This processor was tested on Altera Cyclone II FPGA and proven to become a good substitution for the big USB IP Cores.

The parameterizable bitstream concept and its hardware implementation using the small processor core is introduced in [5]. This concept introduces the fast generation of parameterizable configurations in the commercial FPGAs and its implementation states significantly reduces the resources (up to 80%) in comparison to the MicroBlaze soft processor when it is used as a configuration generation engine.

The novel soft processor core that executes the native Forth language is presented in [6]. The core is designed to be a replacement of an embedded controller running Forth in a VM. The branch prediction architecture, which is part of the designed core, was introduced in order to the execution speed up.

In [7] a RISC 16-bit microcontroller Little16, is proposed. This novel microcontroller provides the small amount of silicon utilization with highly improved performance for the efficient data flow control. It was tested on Xilinx Zynq7000 FPGA platform and yielded a clock speed of 311 MHz at the cost of 366 LUT-6 blocks and 310 Flip-Flops.

The work [8] presents the 8-bit RISC processor with the reduced instruction set (29 instructions) and pipelining. It has 8-bit ALU, two 8-bit I/O ports, eight 8-bit general purpose registers and 4-bit flag register. The proposed processor is verified in the Xilinx Spartan-6 SP605 Evaluation Platform.

The high performance and low power MIPS microprocessor and its implementation in FPGA are proposed in [9]. The authors use different methods to achieve the high performance and low power consumption. They are unfolding transformation, C-slow retiming technique, and double edge registering. The design was tested in Quartus II 9.1 and Stratix II FPGAs and has demonstrated the high performance of the proposed microprocessor.

Other related works are [10], [11], [12]. All mentioned above projects show the high interrest to the small processor IP cores which are utilized for the simple control tasks. Many of them have the minimized hardware volume, but a few processors are well fitted for the serial communications.

3 Microprocessor architecture for serial port communications

2.1 Stack processors

The stack processor architecture is distinguished among all microprocessor architectures. Its instruction set differs in that the operands have implicit addressing because they are usually placed in a few fixed stack registers. Therefore, such instructions have a short instruction length because they have the implicit register addressing. Since these instructions support algorithms that actively use the stack addressing, the programs that are composed for this processor occupy very small memory volume [13].

Various authors have developed several projects of stack processors, which are implemented in the FPGA and which are available for reproduction [14-16]. All of them have 16-bit instructions and process 16-bit data. It is shown in [16], that the stack processor has approximately 2.5 times less program length than the program for the Xilinx MicroBlaze processor when the data exchange protocols through the serial interfaces are implemented. In addition, all stack processors allow the designer to extend the instruction set. To do so, the appropriate changes should be made to the description of the processor at the register transfer level.

Consequently, the architecture of the stack processor provides both firmware amount and hardware costs minimization. In addition, it is easy to develop compilers for such architecture, because, as a rule, its instruction set is a subset of the Forth language commands ([17]). It is known that this language is convenient for both grammatical parsing of lines and for the interpretation of high-level language operators. The stack processor assembly language has the same syntax as the Forth language [13]. Therefore, it is attractive to develop the stack processor architecture, which gives not only minimized hardware costs but also simplified implementation of user instructions, which are adapted to the serial port communications.

2.2 SM8 microprocessor

The structure of the developed SM8 microprocessor is shown in Fig.1. This processor has the well-known two-stack architecture. It consists of a program counter PC, Data RAM, Program ROM, instruction register IR, user instruction encoder UIE, return address stack RS, data stack DS, ALU and peripheral registers R0,..., Rn, $n < 32$. The registers T, N, P are the top registers of the DS-stack and are designed to store the operands and the ALU results. The Program ROM has the volume up to 7936 bytes, and the Data RAM has up to 256 bytes, and both of them have a common address space.

The SM8 microprocessor instructions are sampled in Table 1. All instructions, except CALL, LIT, and IF, have the 8-bit length. The branch and input-output instructions are executed in two cycles, and the rest of the instructions are single cycle instructions. Due to the frequent use of the CALL, LIT, and IF instructions, the average duration of one instruction execution is 1.5 clock cycles.

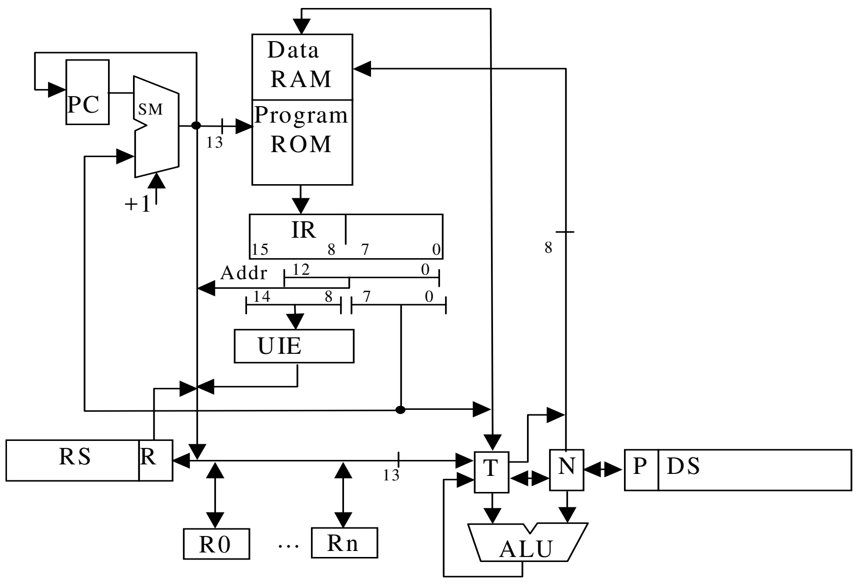

Fig.1. Structure of the SM8 microprocessor

2.3 User-defined instructions

The user-defined instructions are implemented as follows. First, the instruction code is associated with the specified address in the user subroutine library, where the user-defined subroutine is located. Second, when the control-flow approaches this instruction, it writes the instruction code to the IR register. Then the code is encoded by UIE to the address of the subroutine, associated with it. The return address (address of the next instruction) is saved in the RS stack. After that, the control-flow is passed to the first action (sub-instruction) of the subroutine (subroutine is 'called') and all actions which it contains are executed. The return of the control-flow from the subroutine to the next instruction is performed by the RET instruction. This subroutine can also read and process the operand fields that follow the byte of the opcode. But the return address in the R register must be properly corrected.

This instruction is coded by a single byte comparing to the two-byte CALL instruction. Therefore, the user instructions can save the software memory volume comparing to the equivalent instruction implementation using the CALL instruction.

The user-defined instructions can be stored in both Program ROM and Data RAM. Thus, a microprocessor can store a certain dynamic data processing script, which is formed by the user instructions and respective data bytes for them. It can perform a line parsing as well. For example, this line can be a string of decimal calculator operations and digits.

Table 1. Instruction set of the SM8 microprocessor

Name	Instruction	Description
CALL	001 Addr	PC+1 –> R, PC = Addr, subroutine call
INR	010 n	Rn –> T, data receiving
OUTR	011 n	Rn = T, data sending
NOP	0000 0000	No operation
LIT	0000 0001 B	B –> T, constant input
IF	0000 0010 D	PC = PC + D by T = 0, else PC = PC + 1
DUP	0000 0110	N = T
SWAP	0000 1001	X = T, T = N, N = X
@	0000 1010	T = RAM[T], memory reading
!	0000 1011	RAM[T] = N, memory writing, T–>
R>	0000 1100	R –> T
>R	0000 1110	T –> R
RET	0000 1101	R –> PC, return from the subroutine
DROP	0000 1111	T–>, stack purge
NOT	0001 0000	T = not T
OR	0001 0001	T = T or N
AND	0001 0010	T = T and N
XOR	0001 0011	T = T xor N
ADD	0001 1000	T = T + N
INC	0001 1001	T = T + 1
SUB	0001 1010	T = T – N
DEC	0001 1011	T = T – 1
	1xxx xxxx	User instruction

2.4 Dynamic reconfiguration

A common problem for many FPGA-based architectures is the reconfiguration process. Usually, the need in reconfiguration leads to recompilation of a hardware circuit, which is a very CPU-intensive and time-consuming task (it can last from minutes to hours). The solution to this problem was presented in several works [18], [19] as an implementation of task-specific architectures that can be reconfigured 'on-the-fly'. For example, in [19] the authors propose the segment-based architecture for the XML filtering. The sequence of configured segments implements the XPath pattern of the interesting part of the whole XML. This pattern can be changed 'on-the-fly', and the hardware reconfiguration takes from nano- to microseconds.

Another approach is implemented in SM8. The processing script, which is saved in RAM, can be rewritten or loaded from other memory, for example, from ROM. Such simple rewriting allows the system changing the behavior of a chip in terms of microseconds. As far as such action is performed in a synchronization point, neither data loss nor wrong behavior happens. In such a way, the segments in [19] are reconfigured without stop of the input data processing with preserving all currently processed XML node tree parts.

2.5 Assembler of the SM8 microprocessor

An assembler was developed for programming the SM8 microprocessor. The assembler is written in Java and is called from the command line. Below, an example of a program in the SM8 assembly language is shown, which performs a single-byte transfer to the I2C port.

```
DEFINE nap 9        \ memory address width
DEFINE WAITRDY 82h  \ user instruction - wait for port is ready
DEFINE DELAYN  83h  \ user instruction - delay for N cycles
EQU START  2
EQU A_SEND 4
EQU D_SEND 5
EQU STOP   12
EQU PAUSE  15
ORG 256             \ program segment begin
\Write byte to I2C
: WR2I2C (r1 - I2C address, r2 - inner address, r3 - byte,
                             r8 - I2C data, r9 - I2C control)
  lit START outr r9
  inr r1 outr r8 lit A_SEND outr r9 WAITRDY
  inr r2 outr r8 lit D_SEND outr r9 WAITRDY
  inr r3 outr r8 lit D_SEND outr r9 WAITRDY
  lit STOP   outr r9
  lit PAUSE  outr r9
  lit 100    DELAYN  xor if END
;
: DELAYN           (N --  - N cycles)
  dec dup ifn  DELAYN
;
: WAITRDY          (do while rdy=1)
  inr r10          \0-th bit = rdy
  lit 1 and if WAITRDY
;
: END
```

The assembly language of the SM8 core uses the syntax of the Forth language. Therefore, the comments here are enclosed in parentheses or followed after a backslash. The label follows a colon. Operators and literals are separated by spaces. A semicolon indicates the subroutine return instruction.

Some special operators (called the pragmas) are used in the script for the special purpose. Table 2 contains the description of all pragmas.

As it is seen from the example above, none of the subroutines contain the RET instruction. It is explained by the fact that the semicolon sign represents the RET instruction in the Forth language. The user also can specify its own RETs in his subroutine if needed.

The assembler generates two VHDL files, which contain the data and programs in the memory and the user instruction encoder content. As a result, this assembly language by its user properties occupies an intermediate position between the usual assembly language and the high-level language. Thanks to this, writing and debugging of programs is significantly accelerated. Besides, the VHDL model of the SM8 microprocessor core is equipped with a disassembler. Such a feature significantly simplifies the program debugging in the VHDL simulator.

Table 2. Pragmas set for SM8 assembler

Name	Arguments	Description
DEFINE	<name> <value>	associates <name> with <value> for assembler. <name> can be a common setting attribute (like nap – memory address width) or name of a user-defined instruction. <value> is a value for a setting or code of instuction
EQU	<name> <value>	defines constant with name <name>. Each occurrence of <name> in script is replace with <value>
ORG	<shift>	defines a shift of next and all the following instructions in memory to address <shift>

4 Experimental results

The SM8 microprocessor core is described in VHDL and has synthesized for different FPGA circuits. Table 3 presents the results of the SM8 microcontroller synthesis in the Xilinx Spartan-6 FPGA while setting the optimization parameters for hardware costs. Also, the parameters of the microprocessors, which were synthesized in the same conditions, are presented in this table for a comparison. The analysis of the table shows that the SM8 microprocessor has the lowest hardware costs in the look-up tables (LUTs), and in configured logical blocks (CLBs), and the highest speed in millions of instructions per second (MIPS) among stack processors. This is explained by the fact that the reduction of the data bit width up to eight digits reduces both hardware costs and delay in ALU.

Table 3. Parameters of the microprocessor core configured in Xilinx FPGA

Microprocessor	Instruction bit width	Hardware costs		Max. clock frequency, MHz	Speed, MIPS
		LUTs	CLBs		
FS8051 [20]	8, 16, 24	1293	470	89	30
KCPSM6 [2]	18	87	26	140	70
MSL16 [14]	16	235	61	100	67
b16-small [15]	16	280	73	100	50
J1 [16]	16	342	93	106	70
SM8	8, 16	181	50	140	94

Other synthesis results comparison is presented in Table 3. These results show core parameters for some Intel FPGAs.

The waveforms from the ActiveHDL simulator showing the first clock cycles of the processor operation after its reset are presented in Fig. 2. The cop signal shows the opcode as a result of the disassembler function.

Table 4. Microprocessor core parameters in Intel FPGA

Microprocessor	FPGA	Hardware volume	Max. clock frequency, MHz	Speed, MIPS
Nios II/f [21]	MAX10	2268 LE	150	135
Nios II/f [21]	Cyclone 5	867 ALM	170	150
SM8	MAX10	1164 LE	150	100
SM8	Cyclone 5	210 ALM	205	140

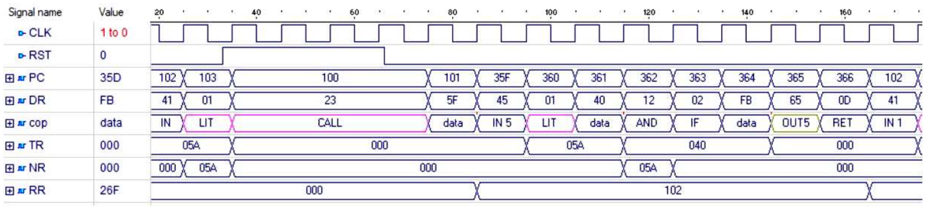

Fig.2. First cycles of simulation in ActiveHDL

5 Conclusion

The proposed SM8 microprocessor core has small hardware costs at high performance and reduced hardware volume. It is designed to implement simple control algorithms, for example, to control the data exchanges through the I2C interface. The core is described in VHDL and can be implemented in an FPGA of any series. The programmer has the ability to add his own instructions to the instruction set without changing the core description. The developed assembler provides to write and compile the programs written in the Forth language style. This simplifies the design of devices that implement the protocols for the serial port communications through interfaces such as RS232, I2C, SPI, Ethernet.

References

1. Processor Design. System-on-Chip Computing for ASICs and FPGAs. Nurmi, J. (ed.) Springer (2007).
2. Chapman, K.: PicoBlaze for Spartan-6, Virtex-6, and 7-Series (KCPSM6). Xilinx, Inc. (2012).
3. Meyer-Baese, U.: Digital Signal Processing with Field Programmable Gate Arrays. 4th edn. Springer (2014).
4. Al-Dujaili, A., Hiung, L. H., Tan, S.: ASH1: A stack-based input/ output processor for USB operations. Proc. of 2015 IEEE International Circuits and Systems Symposium (ICSyS), pp. 76-79. IEEE (2015).
5. Abouelella, F., Bruneel, K., Stroobandt, D.: Efficiently Generating FPGA Configurations through a Stack Machine. Proc. of 2010 International Conference on Field Programmable Logic and Applications, pp. 35-39. IEEE (2010).

6. Hanna, D. M., Jones, B., Lorenz, L., Porthun, S.: An embedded Forth core with floating point and branch prediction. Proc. of 2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS'2013), pp. 1055-1058. IEEE (2013).
7. Hin, W. K., Chiu-Sing, C., Oliver: Little16 - small scale 16-bit controller architecture for FPGA systems flow control. Proc. of TENCON 2015 - 2015 IEEE Region 10 Conference, pp. 1926-1929. IEEE (2015).
8. Jeemon, J.: Low power pipelined 8-bit RISC processor design and implementation on FPGA. Proc. of 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), pp. 476-481. IEEE (2015).
9. Daghooghi, T.: Design and Development MIPS Processor Based on a High Performance and Low Power Architecture on FPGA. International Journal of Modern Education and Computer Science (IJMECS), Vol. 5, No. 5, pp. 49-59. MECS (2013). doi: 10.5815/ijmecs.2013.05.06.
10. Afzal, S., Hafeez, F., Akhter, M. O.: Single Chip Embedded System Solution: Efficient Resource Utilization by Interfacing LCD through Softcore Processor in Xilinx FPGA. International Journal of Information Engineering and Electronic Business (IJIEEB), Vol. 6, No. 7, pp. 23-27. MECS (2015). doi: 10.5815/ijieeb.2015.06.04.
11. Oyetoke, O. O.: A Practical Application of ARM Cortex-M3 Processor Core in Embedded System Engineering. International Journal of Intelligent Systems and Applications (IJISA), Vol. 9, No. 7, pp. 70-88. MECS (2017). doi: 10.5815/ijisa.2017.07.08.
12. Rani, A., Grover, N.: Novel Design of 32-bit Asynchronous (RISC) Microprocessor & its Implementation on FPGA. International Journal of Information Engineering and Electronic Business, Vol. 10, No. 1, pp. 39-47. MECS (2018). doi: 10.5815/ijieeb.2018.01.06.
13. Koopman P.: Stack computers: the new wave. Ellis Horwood, Mountain View Press, CA. (1989).
14. Leong P. H. W., Tsang P.K., Lee T.K.: A FPGA Based Forth Microprocessor. Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, USA, Napa Valley, California (1998).
15. Paysan B.: b16-small — Less is More. Proc. EuroForth 2004, Jul. 9 (2006).
16. Bowman J., Garage W.: J1: a small Forth CPU Core for FPGAs. Proc. EuroForth'2010, January, pp. 1–4 (2010).
17. Kelly, M., Spies, N.: Forth: A Text and Reference. Englewood Cliffs, NJ. Prentice Hall, (1986).
18. Najafi, M., Sadoghi, M., Jacobsen, H.-A.: Configurable Hardware-based Streaming Architecture using Online Programmable-Block, ICDE 2015, Seoul, South Korea, April 13-17, pp. 819-830 (2015).
19. Teubner, J., Woods, L., Nie, C.: XLynx — An FPGA-based XML Filter for Hybrid XQuery Processing. ACM Transactions on Database Systems (TODS), 38 (4), Article 23, November, ACM New York, NY (2013).
20. Maslennikov, O., Shevtshenko J., Sergyienko, A.: Configurable microcontroller array. Proc. Parallel Computing in Electrical Engineering, PARELEC'02, 25 Sept. 2002. Warsaw, Poland. 47-49. IEEE (2003).
21. Nios II Performance Benchmarks. DS-N28162004. Intel. pp. 1-7. (2018).