ELEKTROTEHNIŠKI VESTNIK 79(1-2): 55-60, 2012
ENGLISH EDITION

Design of Custom Processors for the FPGA Devices

Andrej Trost, Andrej Žemva

University of Ljubljana, Faculty of Electrical Engineering, Tržaška 25, 1000 Ljubljana, Slovenia
E-mail: andrej.trost@fe.uni-lj.si

Abstract. Programmable devices in the Field Programmable Gate Array (FPGA) technology enable design and prototyping implementation of complex digital systems executing their tasks on central processing units (CPU) and application-specific logic components. The paper presents development of a custom processor system in the FPGA device. The system is based on a CPU with an accumulator easily extendable with additional instructions. The processor core and program memory are described in a VHDL language and no additional compiler is required. The CPU is optimized for the FPGA devices and synthesis results of 4- to 32-bit cores are presented. An upgrade of the CPU by using peripheral units, Wishbone compatible bus and case with a graphical controller is presented.

Keywords: digital systems, programmable devices, processor, graphical controller

1 INTRODUCTION

Field programmable gate array (FPGA) devices are used for development and prototyping implementation of complex digital systems. These systems are typically based on one or more central processing units (CPU) and custom circuit networks for efficient execution of the tasks [1].

The FPGA vendors provide software tools for digital circuit synthesis from high-level description languages VHDL or Verilog [2]. The basic tools are freely available and can be used for educational purposes [3]. In order to include microprocessors inside the FPGA device, a CPU core generator and a compiler tool are required. Some families of the FPGA devices include embedded standard hardware processor cores (for example ARM or PowerPC). The other FPGA devices can emulate a CPU with the general programmable logic resources. The FPGA vendor Xilinx provide a small 8-bit open-source processor called Picoblaze [4] and a powerful 32-bit CPU core Microblaze, which requires commercial tools.

A CPU-based digital system contains the generally used digital building blocks, such as registers, memory, decoders and computation units with sequential control logic or finite-state machines. Developing a custom microprocessor is a good task for the students learning design of complex digital systems [5]. The customized processor cores are a common research topic [6-8].

Optimal CPU core implementation depends on the technology used, so a typical CPU embedded in the FPGA device has a different instruction set architecture compared to the standard silicon chip processors. In order to efficiently use the custom processor, a new compiler needs to be developed.

The paper presents development of an educational generic CPU core and its optimization for the FPGA technology. The core circuit and the assembler instructions are described in the same VHDL hardware description language and no additional compilers are required. A case study with a custom CPU driving a graphical controller is presented.

2 DEVELOPMENT OF THE CPU CORE

The CPU carries out the program instructions. An instruction set architecture (ISA) defines a set of basic instructions that a processor understands. The instructions and data are stored in the main memory. In a typical embedded system, a CPU with reduced number of instructions RISC (Reduced Instruction Set Computer) is used. The program instructions and data are stored in the main memory and registers in a register file are used for quick temporary data storage.

When building the CPU with FPGA devices, we can simplify the architecture by utilizing the embedded memory blocks instead of the main memory and register file.

We define a simple CPU microarchitecture with one register called accumulator, which is the target register for all data processing instructions [9]. Arithmetic and logic instructions with two operands hold one operand in the accumulator and receive the other operand from the memory block.

Received April 24, 2012
Accepted May 9, 2012

Program instructions are binary codes composed of an instruction code and memory address. In the first educational processor implementation we limit the instruction code to 4 bits which can describe in total 16 operations. The instruction codes are defined as a VHDL constant in a custom VHDL package:

```
subtype  koda is unsigned(3 downto 0);
constant  lda:   koda := "0001";    --  a = [M]
constant  sta:   koda := "0010";    -- [M] = a
constant  add:   koda := "0100";    -- a = a + [M]
constant  sub:   koda := "0101";    -- a = a - [M]
constant  anda:  koda := "0110";    -- a = a and [M]
constant  ora:   koda := "0111";    -- a = a or [M]
constant  jmp:   koda := "1000";    -- jump
constant  jze:   koda := "1001";    -- jump if a=0
```

The memory address is an 8-bit number, which is enough for simple programs used for educational purposes. The complete 12-bit instruction is fetched from the memory in one clock cycle. In the FPGA technology we can define memory blocks of a variable data width, so the same 12-bit data is used for the instruction operands. The proposed 12-bit CPU microarchitecture is presented in Fig. 1.

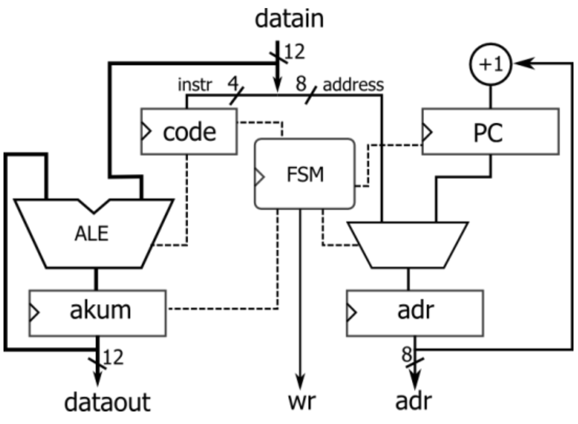

Figure 1. Microarchitecture of the 12-bit CPU core

The processor data path contains two registers: a 4-bit instruction code register and a 12-bit accumulator. The memory is connected with two data busses: datain is the source of the program instructions, while dataout is used for storing the results back to the memory.

The initial CPU core is described in a VHDL language by a very compact code. A synchronous VHDL process is used for describing the registers and data processing instructions in the processor data path. A signal st is a state register from a finite state machine with two alternating states: *fetch* and *execute*. In the state *fetch*, the instruction code is read from the memory. In the state *execute*, the previously fetched instruction is executed. The bus datain contains operand data in the state *execute*.

```
if st=fetch then          -- save instruction code
   code <= instr;
elsif st=execute then -- execute accumulator instr.
   case code is
     when  lda    =>  akum <= datain;
     when  add    =>  akum <= akum + datain;
     when  sub    =>  akum <= akum - datain;
     when  anda  =>  akum <= akum and datain;
     when  ora    =>  akum <= akum or datain;
     when others => null;
   end case;
end if;
```

The CPU controller contains two address registers: a memory address (adr) and a program counter (PC). The program counter holds the address of the next instruction. The controller VHDL description is:

```
--  Control signals, PC and address register
wr_o <= '0';
if st = fetch  then
   st <= execute;
   pc <= adr + 1;    -- address of the next instruction
   adr <= address;  -- addres for the operand
   if instr=sta then
     wr_o <= '1';
   elsif instr=jmp or (instr=jze and akum=0) then
     st <= fetch;
   end if;
else
   st <= fetch;
   adr <= pc;
end if;
```

Most of the instructions are executed in two cycles. An exception is the jump instruction, which is executed in only one cycle. Fig. 2 shows a detailed timing of a small 3-instruction code running on the CPU. The instruction "lda M1" is fetched first from the address 0, and the operand from the address M1 is fetched in the next cycle. The operand is stored in the accumulator. The next instruction "add M2" is executed in a similar manner, and the value from the memory location M2 is added to the accumulator. The result of the jump instruction "jmp 00" is the change in the address for the next instruction.

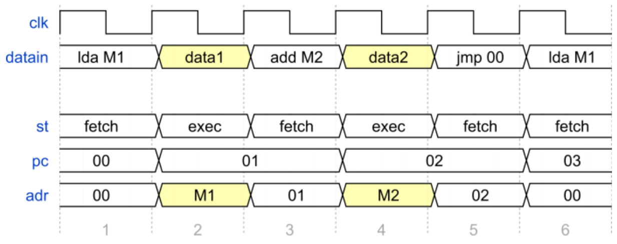

Figure 2. Timing waveform for instructions: lda, add and jmp

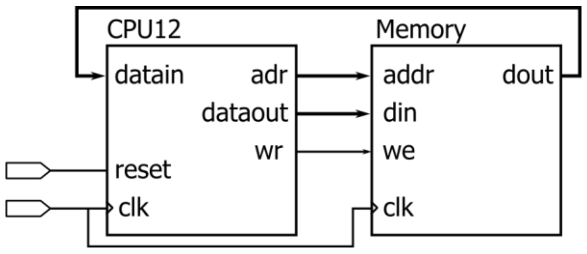

Figure 3. 12-bit CPU connected with the main memory block

Fig. 3 shows the connection between the CPU and the program memory. Memory blocks in the FPGA device have separate read and write data busses. We used a memory block with synchronous writing and asynchronous reading. In the Xilinx FPGA devices, such blocks are implemented with a look-up table memory called distributed memory. Only one instruction, "sta M", performs memory write operation by activating control signal wr_o.

The memory model is in the VHDL described with an array, which should be initialized with the program code:

```
signal m : memory := (
  lda & x"03",
  add & x"03",
  jmp & x"00",
  x"005"
);
```

The instruction codes previously defined in the VHDL package can be used in memory initialization. The content of the memory array looks like a disassembled symbolic code and is quite readable for small programs. The CPU programmer does not need a cross compiler and both the processor hardware and program can be designed in the VHDL. By extending this principle we can design processors with a different ISA and quickly test their operation with a VHDL simulator.

3 UPGRADE AND OPTIMIZATION

The presented CPU microarchitecture is a good initial candidate for upgrading with new instructions and units. Additional arithmetic, shift or logic operations with one or two operands can be directly added to the data path. The designer can add input and output ports for the data connection of the CPU core in the digital system.

3.1 Input and output ports

Input and output (I/O) ports are used for the data transfer between the CPU core and external logic. They can be implemented by mapping certain memory locations to the dedicated registers or with additional I/O instructions and connections in the CPU core. The first solution does not require any change in the processor architecture. On the other hand, we can double the address space by adding only two I/O instructions and minimal logic.

3.1.1 Wishbone Bus

Wishbone is a standardized synchronous parallel system-on-a-chip bus implementing the well-known master Master/Slave (M/S) protocol [10]. The data connections are a one-directional parallel, since this is the preferred architecture in integrated circuits. The standard proposes a variety of interconnection architectures: point to point, data flow, shared bus or switched network.

The basic M/S protocol with handshaking does not require many logic resources in digital systems. A typical write cycle is presented in Fig. 4. The write request is issued by the master asserting signals WE_O and STB_O at the rising clock edge. The slave should respond with an asynchronous acknowledge confirming or delaying the requested operation.

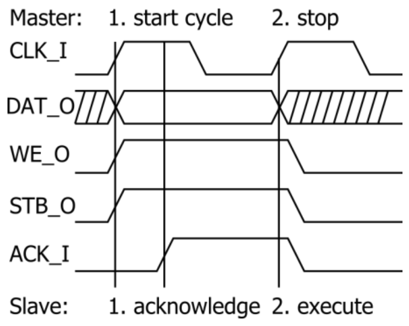
Master: 1. start cycle 2. stop
CLK_I
DAT_O
WE_O
STB_O
ACK_I
Slave: 1. acknowledge 2. execute

Figure 4. Write cycle on the Wishbone bus

Fig. 5 presents a new processor microarchitecture with an input and output port. The data is transferred from the input port to the accumulator with a new instruction "inp M". The output port is implemented as a register in which the accumulator data is transferred by the new instruction "outp M". The address bus is set to the argument value M of both instructions and the Wishbone handshaking signals are activated.

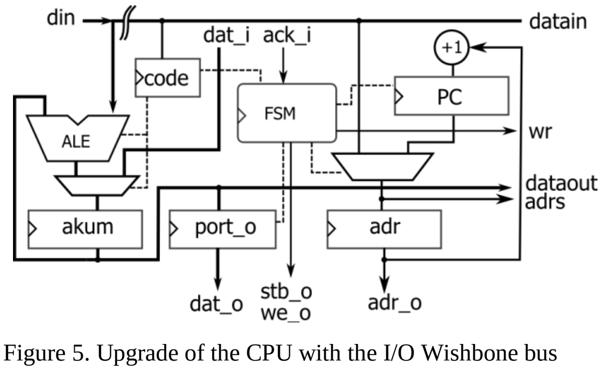

Figure 5. Upgrade of the CPU with the I/O Wishbone bus

3.1.2 Synchronous memory

The contemporary programmable devices contain embedded fully-synchronous memory blocks, by vendor Xilinx called Block RAM. Their memory has one clock delay for write and read cycles, so the data is available one cycle after setting the address. The address register is part of the memory block in order to speed-up the synchronous memory accesses.

Adopting the CPU core to use the synchronous main memory is a simple task. The designer should remove the address register from the memory data path and route the address from the input of this register as presented in Fig. 5 (signal *adrs*). The register itself is still required to hold the next value of the program counter in the control logic loopback.

3.2 Extending the arithmetic and logic unit (ALU) instruction set

New instructions can be added in the CPU data path by extending the selection statement. In order to extend the arithmetic computation capabilities a carry flag flip-flop can be used for storing the addition carry bit or subtraction borrow. Two additional arithmetic instructions should be implemented: addition with a carry bit:

$$akum = akum + [M] + Carry$$

and subtraction with borrow:

$$akum = akum - [M] - Carry$$

These instructions directly described in the VHDL are synthesized to additional adder structures and increase the circuit size. The CPU design optimization should be considered with respect to the FPGA resources.

3.3 ALU optimization

The ALU is the computational core of the processor and presents a large portion of the used resources in a simple CPU architecture. The ALU is a combinational circuit with multiple bus inputs and a regular architecture. The basic addition operation is performed by a series of full adders. The operations can be extended by adding some logic to the inputs or to the output of the basic full adder.

The ALU optimization is based on dividing the ALU to the basic cells called arithmetic cells. Similar structures are register cells, which can be seen as an ALU with output register (in our case accumulator) [11]. Division simplifies the design and logic optimization. For an arithmetic unit with two parameters, we can define the following operation:

$$op = \begin{cases} a + ci, & p = 00 \\ a + b + ci, & p = 01 \\ a + \bar{b} + ci, & p = 10 \\ a - 1 + ci, & p = 11 \end{cases}$$

The operation depends on parameter value *p* and input carry *ci*. When *p*=00, the output is equal to the first input or the input is incremented by 1. When *p*=01, the output is addition with carry, and when *p*=10, we get subtraction with borrow. The combination *p*=11 is used to decrement the input value by 1, when the input carry is 0.

Logic operations can be added by a multiplexer on the output and on two additional parameters m0 and m1. On the multiplexer input, the bit-level operations AND, OR and EX-OR are calculated, as shown in Fig. 6. The multiplexer selects either the result of the arithmetic or of one of the logic operations.

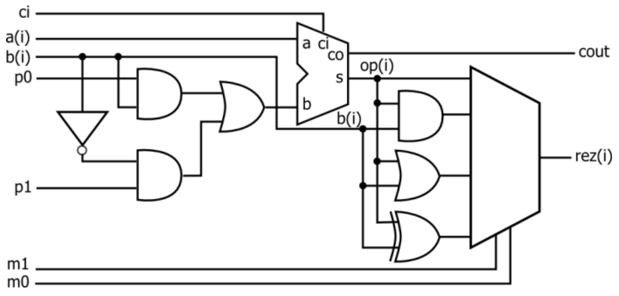

Figure 6. Basic cell of the optimized ALU

The proposed architecture is optimized for mapping the logic into 4-input look-up tables (LUT), which are part of the FPGA structure. One LUT per ALU bit is required for selecting arithmetic operations and one for logic operations. Additional signals are used for carry calculation:

- *Cprop* propagate carry *ci*,
- *Cinv* invert and propagate carry, and
- *Cset* set *ci* to 1 (used for increment and subtract).

Table 1. Decoding table for the ALU instructions

sel(3:0)	p(1:0)	m(1:0)	Cprop	Cinv	Cset	instruct.
0000	00	00	0	0	0	**lda**
0001	01	00	0	0	0	**add**
0010	10	00	0	0	1	**sub**
0011	11	00	0	0	0	**dec**
0100	00	00	0	0	1	**inc**
0101	01	00	1	0	0	**adc**
0110	10	00	0	1	0	**sbc**
1x01	00	01	0	0	0	**anda**
1x10	00	10	0	0	0	**ora**
1x11	00	11	0	0	0	**xora**

Fig. 7 presents a design excerpt with ALU and decoding logic requiring only one LUT for implementation. The complete ALU data path is presented in Fig. 8.

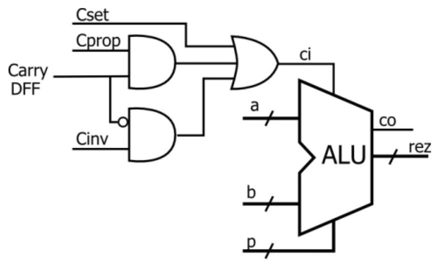

Figure 7. Determination of the input carry value

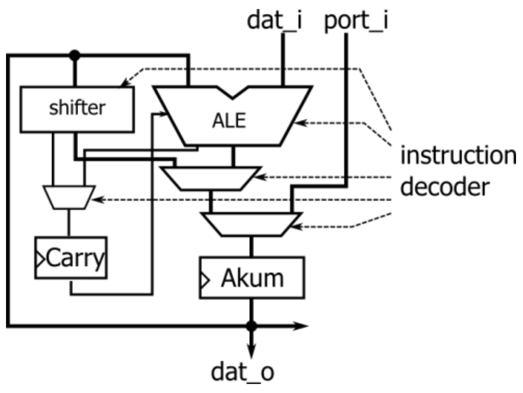

Figure 8. Data path with ALU, shifter, accumulator and carry

3.4 Subroutines and interrupts

In order to support jumps to a subroutine or an interrupt, an address stack is required. It can be implemented as a part of the main memory or as a separate hardware unit.

Fig. 9 presents implementation of the stack hardware unit, which is actually a small LIFO register added in the control logic. The LIFO size is typically 4 to 32 memory addresses for small CPU cores.

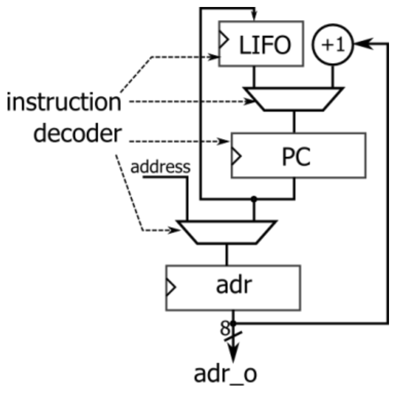

Figure 9. Control part of the CPU with the hardware stack

3.5 Results of generic CPU synthesis

An 8-bit CPU implemented in a Xilinx Spartan-3 FPGA device uses 30 flip-flops and 62 LUT, while a 12-bit CPU uses 34 flip-flops and 79 LUT.

A high-level ALU with 10 operations uses 176 LUT. An optimized ALU with same number of operations occupies only 31 LUT which is 18% compared to the high-level description.

An optimized 12-bit ALU with arithmetic, logic and shift operations uses 46 LUT. A CPU core with 21 instructions occupies 92 LUT, 57 flip-flops and has a maximum frequency 130MHz.

Fig. 10 summarizes synthesis results (in LUT) for a generic CPU core with different data widths, from 4 to 32 bits. The program memory size is either 128 words (CPU) or 8k words (CPU8k).

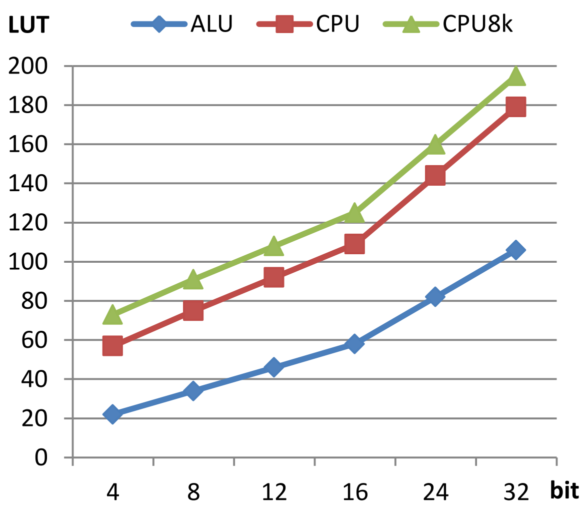

Figure 10. Synthesis results for the generic ALU and CPU

4 Graphical Controller Case Study

Programmable devices are used for processing high-speed signals in real-time, for example video signals [12]. With our case study we will demonstrate the usage of the custom CPU in a graphical controller for computer VGA screen at resolution 640 x 480 pixels [13]. This is a basic monitor graphical mode and is used extensively in mobile devices.

Figure 11 presents components of the graphical controller. A synchronization component (Sinhro) is used for a characteristic VGA timing generator. The color coding component (Color) defines the color of the output pixels, which are internally stored in a pixel memory (Pram). The component MovePixels is in charge of setting image pixels in the pixel memory and character display. This component is connected to the custom 12-bit CPU through the Wishbone bus.

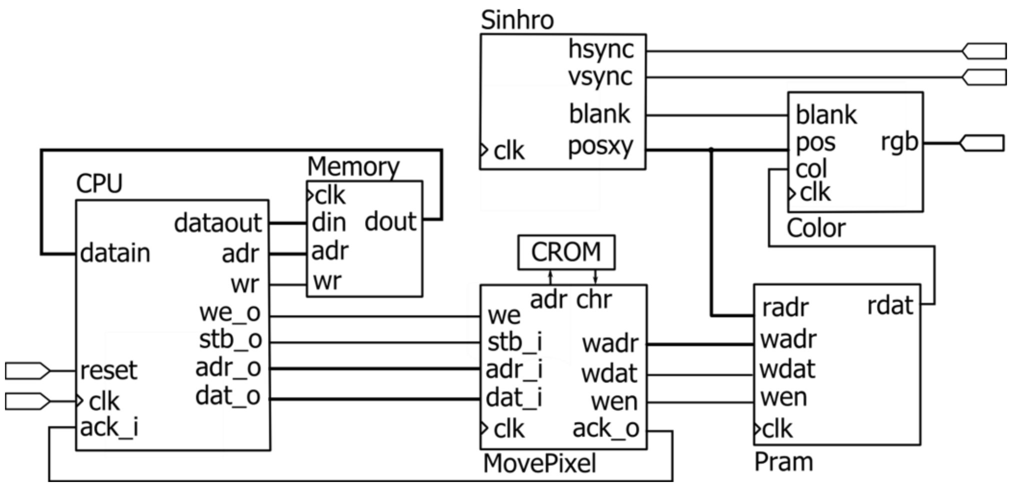

Figure 11. An example of a digital system composed of CPU and graphical controller

The CPU program executes algorithms for drawing basic shapes on the screen. The Bresenham's line and circle drawing algorithms [14] can be implemented as small machine language routines. The CPU native data size (12-bits) is selected according to the algorithm requirements and the image resolution.

5 Conclusions

In the paper we presented development of a generic microprocessor core optimized for the FPGA technology. The CPU circuit with a simplified basic architecture and a lot of extension possibilities is specifically tailored for educational purposes.

References

[1] W. Wolf, FPGA-Based System Design, Prentice Hall, New Jersey, 2004

[2] D. L. Perry, VHDL, 3rd ed. McGraw-Hill, 1998.

[3] D. Hanna and R. E. Haskell, "Learning Digital Systems Design in VHDL by Example in a Junior Course," Proceedings of the ASEE North Central Section Conference, Charleston, West Virginia, March 2007.

[4] PicoBlaze 8-bit Embedded Microcontroller User Guide, Xilinx inc, 2011, www.xilinx.com

[5] V. Angelov, Volker L., 2009 "The Educational Processor Sweet-16", International Conference on Field Programmable Logic and Applications, 2009, Praga, pp. 555-559

[6] G. Hempel, C. Hochberger, "A resource optimized Processor Core for FPGA based SoCs," *Digital System Design Architectures, Methods and Tools,* pp.51-58, August 2007

[7] N. Maheshwari, P. K. Jain, D.S. Ajnar: A 16-Bit Fully Functional Single Cycle Processor, International Journal of Engineering Science and Technology, 2011, vol. 3, no. 8, pp. 6219-6226

[8] M. Schoeberl: Leros: A tiny microcontroller for FPGAs, In Proceedings of the 21st International Conference on Field Programmable Logic and Applications (FPL 2011), September 2011

[9] T Böscke, MCPU - A Minimal 8Bit CPU in a 32 Macro cell CPLD, www.opencores.org

[10] WISHBONE, Revision B.4 Specification, 2010, http://opencores.org/opencores,wishbone

[11] F. Vahid, Digital Design. John Wiley & Sons, Inc., 2007.

[12] S. Lapanja, A. Trost, Testno okolje za razvoj vgrajenih naprav obdelave videosignala, Elektrotehniški vestnik, vol 77, no. 2-3, pp. 137-142, 2010

[13] A. Trost, Načrtovanje digitalnih vezij v jeziku VHDL, založba FE/FRI, 2011

[14] A. Zingl, The Beauty of Bresenham's Algorithm, *http://free.pages.at/easyfilter/bresenham.html, 2012*

Andrej Trost received his Ph.D. degree in 2000 from the Faculty of Electrical Engineering, University of Ljubljana. Currently he works at the same faculty as an assistant professor teaching high-level design techniques on several graduate and post-graduate study levels. His research interests include the FPGA technology and digital-system design for academic and industrial applications.

Andrej Žemva received his B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana in 1989, 1993 and 1996, respectively. He is Professor at the Faculty of Electrical Engineering. His current research interests include digital signal processing, HW/SW co-design, ECG signal analysis, logic synthesis and optimization, test pattern generation and fault modeling.