Laboratory number 3

Implementing a PicoBlaze microprocessor structure (state machine) using a reconfigurable hardware medium

Purpose of the laboratory:
A PicoBlaze™ processor version for Virtex™, Virtex-E, Spartan™-II and Spartan-IIIE FPGA circuits will be studied.

Processor Characteristics:
16-bit instruction set, 16 8-bit general-purpose registers and a 15-entry stack.
- General-purpose registers
- Arithmetic Logic Unit (ALU)
- Flags
- Reset
- Inputs and outputs
- Interrupts

Remark: The use of a microprocessor as a reconfigurable module in FPGA technology greatly simplifies the design process, by allowing for the development of certain applications without requiring advanced VHDL programming knowledge. The application will be developed mainly as a program which will run on the PicoBlaze module. Depending on the FPGA’s capacity, the development of multiprocessor systems on the same chip may or may not be possible, based on the considered application.

Laboratory Application:
The processor structure and the instruction set for PicoBlaze will be described along with the programming, compiling, assembling and integrating into the Spartan II FPGA circuit methods.

Introduction [21]:

One of the trends in FPGA production is to increase the architecture’s granularity by combining the traditional reprogrammable logic blocks in the FPGA with built-in microprocessors, including processor-specific peripherals. As such, a complete integrated system (“System on Chip”) is created. Examples of this type of hybrid technologies can be found in the Xilinx Virtex-II PRO and Virtex-4 circuits, these contain one or more PowerPC microprocessors along with the traditional FPGA. Another example is the Atmel FPSLIC system which makes use of the AVR processor.

An alternative to using microprocessors encapsulated on the same silicone crystal is to create virtual microprocessors by appropriately programming the FPGAs’ logical elements. This type of microprocessor is called a soft processor and it
represents a source program to be written into an FPGA. The source program is known as an „IP core“.

Starting from the above idea that some FPGAs can be programmed while part of the gates execute the code written inside, the reconfigurable systems idea, i.e. self-reconfiguring processing units that adapt to the task-at-hand, starts to take shape. An example of such a reconfigurable processor implemented onto an FPGA is the Mitrion Virtual Processor from Mitrionics. This processor does not have support for dynamic reconfiguration (during program execution) but it can self-adapt to a certain program. In addition, non-FPGA reconfigurable architectures are starting to appear. One such example are the software-configurable microprocessors such as Stretch S5000 which has a hybrid take on the matter by supplying a processing nucleus matrix and a nucleus similar to that of an FPGA on the same silicone crystal.

There are many soft processors available today, both with open and private sources. An important fact is that for any particular microprocessor architecture, the hardware version will always outperform the software implementation. In the following table is a list of some of the more widely-known software microprocessors.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Manufacturer</th>
<th>Source Code</th>
<th>Bus</th>
<th>Remarks</th>
<th>Initial Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSK3000</td>
<td>Altium</td>
<td>Private, for sale</td>
<td>WishBone</td>
<td>Modified Harvard architecture 32-bit RISC.</td>
<td>Altium TSK3000</td>
</tr>
<tr>
<td>TSK51x/52x</td>
<td>Altium</td>
<td>Private, for sale</td>
<td>WishBone/80C51</td>
<td>Compatible with 80C51, 8 bit, better than most 80C51 versions.</td>
<td>TSK51x TSK52x</td>
</tr>
<tr>
<td>TSK80</td>
<td>Altium</td>
<td>Private, for sale</td>
<td>Z80 bus</td>
<td>Optimized instruction set, compatible with Z80, 8 bit.</td>
<td>TSK80</td>
</tr>
<tr>
<td>MicroBlaze</td>
<td>Xilinx</td>
<td>Available through Xilinx license</td>
<td>OPB, FSL, LMB</td>
<td>Compatible Verilog nucleus with MicroBlaze EDK 3.2</td>
<td>Xilinx MicroBlaze</td>
</tr>
<tr>
<td>AEMB</td>
<td>Shawn Tan</td>
<td>Available</td>
<td>Wishbone</td>
<td></td>
<td>AEMB</td>
</tr>
<tr>
<td>PicoBlaze</td>
<td>Xilinx</td>
<td>Available through Xilinx license</td>
<td></td>
<td></td>
<td>Xilinx PicoBlaze</td>
</tr>
<tr>
<td>Nios, Nios II</td>
<td>Altera</td>
<td>Private</td>
<td></td>
<td></td>
<td>Altera Nios II</td>
</tr>
<tr>
<td>Mico32</td>
<td>Lattice</td>
<td>Open-source</td>
<td></td>
<td></td>
<td>LatticeMico32</td>
</tr>
</tbody>
</table>

1 „Intellectual Property core“.
The Xilinx PicoBlaze microprocessor is specifically designed and optimized for the FPGAs: Virtex, Spartan and CoolRunner-II CPLD. This way, fewer resources are used, compared with an 8-bit microcontroller architecture. The processor is available for free, the source code is written in VDHL, which makes this processor easily-adaptable to the new-generation FPGA/CPLD circuit families, this leads to small development costs.

Another advantage of this microprocessor is the fact that it has a very easy to use compiler supplied as a simple DOS-executable program. After compiling, VDHL or Verilog files are generated for the program in order to define the program inside a memory block. Other development tools include graphic interfaces such as IDEs – Integrated Development Environments, a graphical instruction set simulator (ISS), VDHL source-code and simulation models.

Features: PicoBlaze can run between 44 and 100 million instructions per second (MIPS) depending on the circuit family and the speed grade – this makes it much faster than commercial microcontrollers.

PicoBlaze takes up 192 logical cells (5% of the Spartan-3 XC3S200 circuit), 76 logical Spartan-IIIE cells (9% of the XC2S50E circuit and 2.5% of the XC2S300E circuit). Since it takes up only a fraction of the FPGA circuit, several PicoBlaze processors can be integrated into such a circuit.

It has a 100% integration capacity: PicoBlaze can be completely integrated into FPGAs and CPLDs and does not require any external resource. Another advantage is the fact that its basic functionalities can be extended by connecting additional logic to the microcontroller’s input/output ports.
Comparison for PicoBlaze in different circuit families:

<table>
<thead>
<tr>
<th>Feature</th>
<th>PicoBlaze for Spartan-3, Virtex-II/Pro and Virtex-4</th>
<th>PicoBlaze for Virtex-E and Spartan-II/E</th>
<th>PicoBlaze for CoolRunner-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Space</td>
<td>1024</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>Instruction Size</td>
<td>18 bit</td>
<td>16 bit</td>
<td>16 bit</td>
</tr>
<tr>
<td>Internal Programs</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>8-Bit Registers</td>
<td>16</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Stack Depth</td>
<td>31</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>Assembler</td>
<td>KCPSM</td>
<td>KCPSM</td>
<td>ASM</td>
</tr>
<tr>
<td>Size</td>
<td>96 Spartan-3 slices</td>
<td>76 Spartan-IE slices</td>
<td>212 macrocells in XC2C256</td>
</tr>
<tr>
<td>Performance</td>
<td>44 MIPS (Spartan-3)</td>
<td>76 MIPS (Virtex-II)</td>
<td>19 MIPS (Spartan-IE)</td>
</tr>
<tr>
<td></td>
<td>100 MIPS (Virtex-II Pro)</td>
<td>100 MIPS (Virtex-I)</td>
<td>21 MIPS (Spartan-IE)</td>
</tr>
<tr>
<td></td>
<td>100 MIPS (Virtex-4 LX, SX)</td>
<td>100 MIPS (Virtex-4 FX)</td>
<td></td>
</tr>
</tbody>
</table>

PicoBlaze is also known as **KCPSM = Constant (k) Coded Programmable State Machine** – Functional nucleus.

The main PicoBlaze module – The VHDL module which distributes part of the FPGA as a program memory.

► PicoBlaze Architecture [22]

►► General-purpose registers: there are 16 such registers, specified as s0-sF (can be renamed using the assembler). All operations using the registers are completely flexible, without any special-purpose registers. There is no accumulator, but any register can be used as such. **The Arithmetic Logic Unit** (ALU) handles all simple operations which can be expected from an 8-bit processing unit.

Any operation uses the operand supplied by any register and the result is returned to the same register. For operations which require a second operand, another register or a constant 8-bit value must be used.
For operations requiring more than 8 bits, addition or subtraction, there is the option of including a CARRY. In addition, bitwise operations are available: LOAD, AND, OR, XOR.

The ALU operations influence the values of the ZERO and Carry flags. This information determines the program’s instruction flow by means of conditional and non-conditional instructions. The JUMP instruction specifies an absolute address in the program memory; the CALL and RETURN instructions provide sub-program facilities. The CALL is made to an absolute address while the return address is saved in the program counter stack.

The RESET command forces the processor to return to its initial state. The program is executed starting with address 00 and the interrupts are inactive. The status and stack flags CALL/RETURN are also reset but the registers will remain unaffected.

►► Inputs/Outputs

The PicoBlaze module has 256 input and 256 output ports. An 8-bit address supplied to the PORT_ID bus together with the READ_STROBE or WRITE_STROBE signals are used to specify the accessed port. The port’s address can be included in the program as an absolute value or it can be indirectly specified by writing it into one of the 16 general-purpose registers. The indirect addressing mode is ideal for accessing a memory block. For an input operation (indicated by the READ_STROBE signal), the value on the input port is transferred to any one of the 16 registers. For an output operation (WRITE_STROBE), the content of any of the 16 registers can be transferred to the output port.

►► Interrupts

The processor has only one interrupt signal. This interrupt signal is implicitly inactive and can be activated through the program. By activating an interrupt PicoBlaze macro is forced to initiate the “CALL FF” sub-routine (which calls the last location in the program memory).

The ZERO and CARRY flags are saved automatically and any following interrupts are deactivated. A special command (RETURNI) insures that at the end of the interrupt sub-routine the state of all flags is restored and future interrupts will be activated.

The PicoBlaze module can be viewed as a constant-based state machine (constant values for the ALU operands, constant port values and constant address values for the program flow control).

All instructions, in any conditions are executed in 2 clock cycles. The program is 256 instructions long, in accordance with the 256x16 format of the RAM block for the Virtex-E or Spartan-IIE.

PicoBlaze is supplied as a VDHL code (kcpsm.vhd), which will not be modified in any way.
Declaring the KCPSM component in VHDL

```vhdl
component kcpsm
  Port (
    address : out std_logic_vector(7 downto 0);
    instruction : in std_logic_vector(15 downto 0);
    port_id : out std_logic_vector(7 downto 0);
    write_strobe : out std_logic;
    out_port : out std_logic_vector(7 downto 0);
    read_strobe : out std_logic;
    in_port : in std_logic_vector(7 downto 0);
    interrupt : in std_logic;
    reset : in std_logic;
    clk : in std_logic);
end component;
```

Initializing the KCPSM component in VHDL

```vhdl
processor: kcpsm
  port map (  
    address => address_signal,  
    instruction => instruction_signal,  
    port_id => port_id_signal,  
    write_strobe => write_strobe_signal,  
    out_port => out_port_signal,  
    read_strobe => read_strobe_signal,  
    in_port => in_port_signal,  
    interrupt => interrupt_signal,  
    reset => reset_signal,  
    clk => clk_signal);
```

The assembler generates a VHDL-type file after compiling the source file in which the RAM block and the initial content are defined. This VHDL file can be used to implement and simulate the processor.

► PicoBlaze-Instruction set

To describe the instructions:
1. “X” and “Y” refer to the “s” registers from 0 to F.
2. “kk” – constant values between 00 - FF.
3. “aa” – addresses between 00 - FF.
4. “pp” – port address between 00 - FF.

►► Program Control Instructions Group

- JUMP aa
- JUMP Z,aa
- JUMP NZ,aa
- JUMP C,aa
- JUMP NC,aa
- CALL aa  //Call and Return support a 15-level stack
- CALL Z,aa
- CALL NZ,aa
- CALL C,aa
• CALL NC,aa
• RETURN
• RETURN Z
• RETURN NZ
• RETURN C
• RETURN NC

►► Logic Instructions Group
• LOAD sX,kk
• AND sX,kk
• OR sX,kk
• XOR sX,kk
• LOAD sX,sY
• AND sX,sY
• OR sX,sY
• XOR sX,sY

►► Arithmetical Instructions Group
• ADD sX,kk
• ADDCY sX,kk
• SUB sX,kk
• SUBCY sX,kk
• ADD sX,sY
• ADDCY sX,sY
• SUB sX,sY
• SUBCY sX,sY

►► Shift and Rotate Instructions Group
• SR0 sX
• SR1 sX
• SRX sX
• SRA sX
• RR sX
• SL0 sX
• SL1 sX
• SLX sX
• SLA sX
• RL sX

►► Input/Output Group
• INPUT sX,pp
• INPUT sX,(sY)
• OUTPUT sX,pp
• OUTPUT sX,(sY)

►► Interrupt Group
• RETURNI ENABLE
• RETURNI DISABLE
• ENABLE INTERRUPT
• DISABLE INTERRUPT
Further details about the processor’s instructions are available in Appendix 3.

► PicoBlaze Assembler

The assembler is supplied as a DOS executable file and 2 template files. The KCPSM.EXE, ROM_form.vhd, and ROM_form.coe files will be copied in the user’s working folder. The programs will be written in a text editor (Notepad or WordPad). The source file will be saved with the .psm extension (there is an 8-character limit to the file name).

A DOS `kcpsm <filename>[.psm]` window will open.

The compiling operation will stop when it finds errors and will display the row in which the problem occurred. In order to save the messages displayed by the assembler, the following command line will be used. The assembler process will stop the moment it detects an error. `kcpsm <filename>[.psm] >> screen_dump.txt`

The PicoBlaze Assembler (KCPSM.EXE) uses the 3 input files ROM_form.vhd, ROM_form.coe and the source program code which is a *.psm extension file and generates the VHDL program code.

►► Assembler Directives

- **The CONSTANT Directive** A constant is global, even if it is defined at the end of a program and it can be used anywhere inside that program. By means of constants, the code becomes easier to follow and can be modified faster.
The NAMEREG Directive

This directive allows for the naming of any of the 16 general-purpose registers, this leads to a better understanding of the program code.

| CONSTANT max_count, 18 ;count to 24 hours |
| NAMEREG s4, counter_reg ;define register for counter |
| CONSTANT count_port, 12 |
| start: LOAD counter_reg, 00 ;initialize counter |
| loop: OUTPUT counter_reg, count_port |
| ADD counter_reg, 01 ;increment |
| LOAD s0, counter_reg |
| SUB s0, max_count ;test for max value |
| JUMP NZ, loop ;next count |
| JUMP start ;reset counter |

The ADDRESS Directive

This directive supplies a means of forcing the assembly of the next instruction starting from a new address value; this is useful for separating subroutines in specific locations and for handling interrupts. The addresses are specified as hexadecimal values between 00 and FF.

| JUMP NZ, inner_loop |
| RETURN ;Interrupt Service Routine |
| ISR: LOAD wait_light, 01 ;register press of switch |
| OUTPUT wait_light, wait_light_port ;turn on light |
| RETURN DISABLE ;continue light sequence but no more interrupts |
| ADDRESS PF ;interrupt vector |
| JUMP ISR |

Program Syntax

- There can be no empty lines, in order to keep a line free of code, the “;” character is introduced; this character is used to mark a commentary in the program code.
- The registers are defined as s0…sF
- Constants between 00-FF must be written in hexadecimal using 2 digits.
• Labels are user-defined and case-sensitive
• The instructions must respect the instruction set format. For 2-operand instructions the comma is used for separation.

Example:

LOAD s5, 7E
ADDCY s8, sE

Programming steps:

• High-level VHDL design which will include PicoBlaze
  • Writing the processor program “myprog.psm”,
  • Running KCPSM.EXE which will also use some template files
  • A “myprog.vhd” file will result which will contain generic simulation data and the code for the synthesis operation.
  • This file will be included into the project and will initialize ROM program memory into the project.

Main files for the software PicoBlaze processor (in the xapp213.zip archive) can be downloaded from the Xilinx website.

kcpsm.vhd – VHDL definitions for PicoBlaze.
kcpsm.ngc – an alternate file used to define PicoBlaze which will be employed as a black-box in a new VHDL design.
kcpsm_embedded – VHDL file in which the software PicoBlaze processor is connected to the associated ROM program memory. It can also be used as an example.

KCPMS.EXE – The assembly program
ROM_form.vhd – VHDL file read by the assembler and used to define the manner in which the ROM memory will be implemented. This file must be placed in the same folder as KCPMS.EXE.
ROM_form.coe – defines the manner in which the ROM memory will be implemented. This file must be placed in the same folder as KCPMS.EXE.
int_test.psm – Example PSM file used in documentation.
kcpsm_int_test.vhd – VHDL design example used in documentation.
test_bench.vhd – VHDL example test file to be used along with kcpsm_int_test.vhd in order to reproduce the waveforms shown in the documentation.

Other files

PSMSPLIT.EXE – utility program used to double the program memory size and switch between memory banks.
pmsplit.vhd – VHDL template file read by the PSMSPLIT utility program. This file must be placed in the same folder as the utility program.
kcpsm_split_rom.vhd – design example for which PSMSPLIT is used.
lo_prog.psm – the first program used to design kcpsm_split_rom.
hi_prog.psm – the second program used to design kcpsm_split_rom.
seven_segment_display.vhd – also used to design kcpsm_split_rom, but can also be useful in other projects, as it is a decoder for a 7-segment LED.
UART_Manual.pdf – documentation for a simple UART, macro used for transmission and reception, fit for connecting with the PicoBlaze Soft Processor.

uart_tx.vhd - UART transmitter, 8-bit, no parity, 1 stop bit, 16-byte FIFO buffer.

uart_rx.vhd - UART receiver, 8-bit, no parity, 1 stop bit, 16-byte FIFO buffer.

kcuart_tx.vhd - UART transmitter, 8-bit, no parity, 1 stop bit. Can also be used as stand-alone but is usually employed as part of uart_tx.vhd.

kcuart_rx.vhd - UART receiver, 8-bit, no parity, 1 stop bit. Can also be used as stand-alone but is usually employed as part of uart_rx.vhd.

bbfifo_16x8.vhd – Synchronous FIFO 16-byte buffer. Used in uart_tx.vhd and uart_rx.vhd, but can also be used as stand-alone.

Example: Write a program sequence which compares 2 values.

1st alternative:
LOAD s0, sF
SUB s0, 27
JUMP Z, my_routine

2nd alternative:
Count up: ADD sF, 01 ; Increment the counter
SUB sF, 27 ; test if the counter equals 27
JUMP Z, my_routine ; the counter value is 27
ADD sF, 27 ; the counter value is not 27 and its initial value is restored
JUMP count up

3rd alternative:
INPUT sF, switch_port
AND sF,sF ; test for zero
JUMP NZ, switch_routine

Remark: From http://www.mediatronix.com/pBlazeIDE.htm, pBlaze IDE can be downloaded – it is an integrated development environment dedicated to the KCPSM/PicoBlaze soft processor, invented by Ken Chapman from Xilinx. The program allows editing, compiling and troubleshooting the KCPSM/PicoBlaze source codes.

► Design stages description and details

The problem at hand is to display digits on the 7-segment LED display after the push-button is pressed. The PicoBlaze module and the XSA-50 development board will be used.

- Write the program in assembler language
- Compile the program using the tools available in the PicoBlaze processor archive.
- Integrate it into a Xilinx project.
- Test the program using ModelSim
- Test the program on the development board.
The program is as follows:

; at every button press (connected on IN0)
; an internal 8-bit register is incremented
; and the result is transmitted to the LED group.
;
; in this simple case, no port addressing is required
; an arbitrary address is used (00)

;The program in PicoBlaze assembler language

LOAD s1, 00  ; starting code (will be displayed on the LED)
ciclu:    OUTPUT s1, 00 ; writes the code from s1 to the LED group
INPUT s0, 00 ; reads the button state.
AND s0, 01;  masking – if the button is pressed => becomes 0
JUMP NZ, ciclu ; if the button is not pressed, it returns to the loop
ADD    s1, 01 ; increment
JUMP ciclu

The program is saved under the name Led_test.psm. In the same folder, the KCPSM.EXE, ROM_form. Vhd, ROM_form.coe files are copied and a .bat file named, for example, assemble.bat is defined, this file has the following content:

ekcp sm led_test.psm
pause

Execute assemble.bat and compile the source-code file, the result will be a VHDL file.
Define a Xilinx project using the Spartan-II circuit coordinates which have been used up until now.
Add the Led_test.vhd source file to the defined project. The link file should appear as follows:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Example1_PicoBlaze is
  Port (  clk_50M Hz : in std_logic;
          switches  : in std_logic_vector(7 downto 0);
          leds: out std_logic_vector(7 downto 0));
end Example1_PicoBlaze;

architecture Behavioral of Example1_PicoBlaze is

component KCPSM
```vhdl
port (
    address : out std_logic_vector(7 downto 0);
    instruction : in std_logic_vector(15 downto 0);
    port_id: out std_logic_vector(7 downto 0);
    write_strobe: out std_logic;
    out_port: out std_logic_vector(7 downto 0);
    read_strobe: out std_logic;
    in_port: in std_logic_vector(7 downto 0);
    interrupt : in std_logic;
    reset : in std_logic;
    clk: in std_logic
);
end component;

component led_test
    port (
    address : in std_logic_vector(7 downto 0);
    instruction : out std_logic_vector(15 downto 0);
    clk: in std_logic
);
end component;

signal addx_bus: std_logic_vector(7 downto 0);
signal inst_bus: std_logic_vector(15 downto 0);

begin
    -- describing the interconnection scheme
    processor: kcpsm
        port map(
            address => addx_bus,
            instruction => inst_bus,
            port_id=> open,
            write_strobe=> open,
            out_port=> leds,
            read_strobe=> open,
            in_port=> switches,
            interrupt => '0',
            reset => '0',
            clk=> clk_50MHz
        );

    program: led_test
        port map(
            address => addx_bus,
            instruction => inst_bus,
            clk=> clk_50MHz
        );

    end Behavioral;
```
--- execute the following actions
--- a) load the kpctsm.vhd file
--- b) compile the program associated to a led_test.vhd module
--- c) load the resulted the source file led_test.vhd
--- continue as with any other project (assign pins etc.)

-- the program describing the application can be found in led_test.psm
-- this program can be compiled using assem.bat -> the led_test.vhd file results,
--corresponding to the PicoBlaze ROM memory.

The diagram viewed by means of Xilinx-ECS is shown in the image below:

Use the constraints file to associate the pins to the LEDs (for the XSA50 board, the pins are the same as described in Laboratory 1), to the push-button (P93) and to the clock signal (P88).

By introducing appropriate test waveforms and using ModelSim you can check the resulting system's functioning (figure below). In this case, the button has been pressed during the second clock cycle, even so, the effect can only be observed after several clock cycles. This can be explained by the fact that 2 clock cycles are necessary for each instruction to be executed by PicoBlaze.
Conclusions:
In this laboratory, introductory notions regarding soft processors have been presented, by illustrating one such processor – PicoBlaze, offered by Xilinx; in addition a simple application has been implemented from problem definition to FPGA implementation.