Specification

Project an automaton which resembles a micro-controller on 8 bits. This micro-controller should have specific operations with 8 bit numbers:

- ALU Unit:
  - Logical operations:
    - Load on 8 bits
    - And on 8 bits
    - Or on 8 bits
    - Xor on 8 bits
  - Arithmetic operations:
    - Half-adder on 8 bits
    - Full-adder on 8 bits
    - Half-subtractor on 8 bits
    - Full-subtractor on 8 bits
  - Shift Register operations:
    - Shift Left (SL0,SL1,SLL,SLA) on 8 bits
    - Rotate Left(ROL) on 8 bits
    - Shift Right (SR0,SR1,SRL,SRA) on 8 bits
    - Rotate Right(ROR) on 8 bits
- Program Counter Unit:
  - Jump (with Zero/Carry units)
  - Call (with Zero/Carry units)
  - Return (with Zero/Carry units)
- Interrupt Control Unit:
  - Enable/Disable Interrupt the Program Counter
  - ReturnI Enable/Disable from the Program Counter
- Input/Output Unit:
  - Input – a number into a register for the ALU Unit
  - Output – a number from the registers of the ALU Unit represented on the LED 7-Segment Output Element from the FPGA Board. The numbers are represented in hexadecimal (from 00 to FF).

This micro-controller should also support interrupt operations.

Block Diagram

We have created a block diagram of this project in order to show all the components of the micro-controller and its structure. We chose this method in order to understand the function of this automata by having each component structurely detailed.

The device is formed by two major components: the assembler and the micro-controller. These components have a special bond between them in order to be user friendly. All the details of each component is presented in the pages below. The block diagrams for each component of the micro-controller are attached at the end of the document.

Components

The micro-controller contains different blocks having special functions, all of them communicating inside the device via some wires and addresses. The components of the micro-controller are:

1. RAM Memory
2. ALU – formed by:
    - Shift Register Unit
    - ALU_2
3. Program Counter
4. Program Stack RAM with Counter
5. Zero and Carry Flag Store
6. Interrupt Flag Store
7. ROM Instructions Memory
8. Debouncer
9. BCD LED 7-Segment Display

Implementation

1. RAM Memory

The RAM Memory is the element which the ALU unit uses its numbers in order to do all the operations specified by the ROM Instructions Memory. This memory is made by 16 registers (coded from 0 to F – in hexadecimal), each register storing an 8-bit number (from 00 to FF). This RAM Memory has:

```
entity register_ent is
    port(input: in std_logic_vector(7 downto 0);
    reg_select1: in std_logic_vector(3 downto 0);
    reg_select2: in std_logic_vector(3 downto 0);
    clk: in std_logic;
    reset:in std_logic;
    output1: out std_logic_vector(7 downto 0);
    output2: out std_logic_vector(7 downto 0));
end register_ent;

architecture arh1 of register_ent is
signal internal_cont: reg_cell;
begin
    process(clk,reset)
    begin
        if reset='1' then
            internal_cont<=(others=>"00000000");
        else
            if clk'event and clk='1' then
                internal_cont(conv_integer(reg_select1))<=input;
            end if;
        end if;
    end process;
    output1<=internal_cont(conv_integer(reg_select1));
    output2<=internal_cont(conv_integer(reg_select2));
end architecture arh1;
```

- Input signal *input* – the input signal on 8 bits – the value is written on the register with the ID the first select signal.
- Select signals *reg_select1* and *reg_select2* - these signals are on 4 bits and indicate which registers' numbers are on the output signals *output1* and *output2*.
- Clock signal – *clk;*
- Reset signal – *reset;*
- Output signals *output1* and *output2* – the values of these signals are the values of the registers with the IDs *output1*, respectively *output2*.

2. ALU

ALU (Arithmetic and Logical Unit) is a component which uses the elements from the RAM Memory in specific operations chosen by the user. The port map of the ALU is represented by:

```
entity ent_ALU is
    port(A,B,Const: in std_logic_vector(7 downto 0);
    Instr: in std_logic_vector(15 downto 0);
    CIN: in std_logic;
    Y: out std_logic_vector(7 downto 0);
    Zero,Carry: out std_logic);
end;
```

- Input signals *A,B* and *Const* – *A* is *output1*, *B* is *output2*, both from the RAM Memory and *Const* is a constant number.
- Instruction signal *Instr* – this signal is on 16 bits and comes from the ROM Instructions Memory and says which operation does the ALU.
- *CIN* – Carry In signal for the arithmetic operations.
- Output signal *Y* – this signal goes into the RAM Memory.
- *Zero* and *Carry* – these signals are flags which go into the flag store.

ALU has two major components:

- ALU_2 – this element contains the logical and arithmetic operations coded with the selections:
  - *0000* – load B in the register A;
  - *0001* – A and B;
  - *0010* – A or B;
  - *0011* – A xor B;
  - *0100* – A + B;
  - *0101* – A + B + CIN;

- *0110* – A – B;
- *0111* – A –B – CIN;

The function of ALU_2 is shown below:

```
begin
    op: process(A,B,CIN,SEL)
    begin
        case SEL is
            when "000" => intY <= B;
            COUT<='0';
            when "001" => intY <= A and B;
            COUT<='0';
            when "010" => intY <= A or B;
            COUT<='0';
            when "011" => intY <= A xor B;
            COUT<='0';
            when "100" => sum <= ('0'&A) + ('0'&B);
            COUT<=sum(8);
            intY<=sum(7 downto 0);
            when "101" => sum <= ('0'&A) + ('0'&B) + CIN;
            COUT<=sum(8);
            intY<=sum(7 downto 0);
            when "110" => sum <= ('0'&A) - ('0'&B);
            COUT<=sum(8);
            intY<=sum(7 downto 0);
            when "111" => sum <= ('0'&A) - ('0'&B) - CIN;
            COUT<=sum(8);
            intY<=sum(7 downto 0);
            when others => sum <= "000000000";
        end case;
        Y<=intY;
        if intY = "00000000" then ZERO <= '1';
        else ZERO <= '0';
        end if;
    end process op;
end architecture operatii;
```

- Shift_Reg – This is an universal shift module which has different functions:
  - SL0/SR0 – the elements are shifted to left/right, taking out an element and inserting a ’0’;
  - SL1/SR1 – the elements are shifted to left/right, taking out an element and inserting a ’1’;
  - SLL/SRL – shift left/right logical;
  - SLA/SLL – shift left/right arithmetical;
  - ROL/ROR – the elements are rotated to left/right.

The way in which the shift module works is written below:

```
signal iR1,iR2,intCarry0,intCarry1:std_logic;
signal iR3,iR4,intR:std_logic_vector(7 downto 0);

begin
    U0:MUX8_1 port map(CarryI,'0',A(0),'0',A(7),'0','0','1',sel,iR1);    --shift right
    U1:MUX8_1 port map(CarryI,'0',A(7),'0',A(0),'0','0','1',sel,iR2);    --shift left
    iR3(6 downto 0)<=A(7 downto 1);    --shift left
    iR3(7)<=iR1;
    iR4(7 downto 1)<=A(6 downto 0);    --shift left
    iR4(0)<=iR2;
    U2:MUX2_1
    generic map(pathwidth=>8)
    port map(iR4,iR3,B3,intR);
    U3:MUX2_1
    generic map(pathwidth=>1)
    port map(I0(0)=>intCarry0,I1(0)=>intCarry1,S0=>B3,O(0)=>CarryO);
    intCarry0<=A(7);
    intCarry1<=A(0);
    R<=intR;
    process(intR)
    begin
        if intR="00000000" then Zero<='1'; else Zero<='0'; end if;
    end process;
end architecture arh2;
```

3. Program Counter

The Program Counter is an 8-bit counter which returns an address that is used to access an instruction from the ROM Memory. The counter has the following signals:

```
entity num_princ is
    port(clk: in std_logic;
    reset: in std_logic;
    le: in std_logic;
    interruptSig: in std_logic;
    addressI: in std_logic_vector(7 downto 0);
    addressO: out std_logic_vector(7 downto 0));
end num_princ;

architecture arh1 of num_princ is
signal intQ: std_logic_vector(7 downto 0):="00000000";
begin
    process(clk,reset)
    begin
        if reset='1' then
            intQ<="00000000";
        else
            if clk'event and clk='1' then
                if le='1' then
                    intQ<=addressI;
                elsif interruptSig='1' then
                    intQ<="11111111";
                else
                    intQ<=intQ+1;
                end if;
            end if;
        end if;
    end process;
    addressO<=intQ;
end;
```

- Input address *addressI* – the address which is loaded into the program counter from the program flow control using JUMP function.

- *CLK* – Clock signal.
- *Reset* – It resets the program counter if activated.
- *LE* – Load Signal.
- *InterruptSig* – This signal is activated if there is and interruption in the program counter.
- Output address *addressO* – the address returned and used into the ROM Instructions Memory.

4. Program Stack RAM with Counter

This component is represented by two elements:

a. Stack RAM Memory:

The Stack RAM Memory is a RAM Memory constructed a stack. It can store up to 16 address on 8 bits. Its ports are:

```
entity stack is
    port(WE: in std_logic;
    A: in std_logic_vector(7 downto 0);
    SEL: in std_logic_vector(3 downto 0);
    CLK: in std_logic;
    Y: out std_logic_vector(7 downto 0));
end stack;

architecture stack_RAM of stack is
type RAM_cell is array (15 downto 0) of std_logic_vector(7 downto 0);
signal RAM_stack: RAM_cell;
begin
    stiva: process(CLK,WE,A)
    begin
        if(CLK'EVENT) and CLK = '1' then
            if WE = '1' then
                RAM_stack(conv_integer(SEL+1))<=A;
                Y<=RAM_stack(conv_integer(SEL));
            end if;
        end if;
    end process stiva;
end architecture stack_RAM;
```

o Address *A* – The address from the Program Counter which is stored into the stack if the user used either JUMP of CALL instruction.

o *WE* – Write Enable;

o *SEL* – The position where the element is written into the stack.

o *CLK* – Clock Signal;

o *Y* – The address which is returned to the Program Counter if the user uses the RETURN instruction.

b. Counter:

This element generates the SEL signal of the Stack RAM. This counter works only when the CALL or RETURN instructions are used. It is formed by:

- *CE* – Clock Enable of the counter
- *CU* – Clock of the system. It counts up if CU = ’1’ and down if CU = ’0’.
- *Y* – The *SEL* signal used for the Stack RAM.

```
entity counter4bit is
    port(CU: in std_logic;
    CE: in std_logic;
    Y: out std_logic_vector(3 downto 0));
end counter4bit;

architecture numar of counter4bit is
signal intQ: std_logic_vector(3 downto 0):= "0000";
begin
    numara: process(CU,CE)
    begin
        if(CE = '1') then
            if(CU = '1') then
                intQ<=intQ + 1;
            end if;
        end if;
        Y<=intQ;
    end process numara;
end architecture numar;
```

5. Zero & Carry Flag Store

This component stores the signals Zero and Carry from the ALU Unit and being reused in the ALU. This component also communicates with the Interrupt Flag Store. Its structure is represented by:

```
entity flg_store is
    port(reset:in std_logic;
    interrupt:in std_logic;
    inpCarry,inpZero:in std_logic;
    retCarry,retZero:in std_logic;
    instr:in std_logic_vector(15 downto 0);
    clk:in std_logic;
    carry,zero:out std_logic);
end flg_store;
```

- *Reset* – it resets all the flags.
- *Interrupt* – the signal shows whether the interrupt is enabled.
- *inpCarry, inpZero* – Zero & Carry signals returned from the ALU Unit.
- *retCarry, retZero* – Zero & Carry signals returned from the Interrupt Flag store.
- *Instr* – the instruction from the ROM Instructions Memory.
- *CLK* – clock signal;
- *Carry, Zero* – Zero & Carry signals returned to the ALU Unit or Interrupt Flag Store.

6. Interrupt Flag Store

This flag store is activated only when there is an interruption in the program, storing the Zero & Carry signals. The flag store has:

```
entity interrupt_store is
    port(inCarry, inZero:in std_logic;
    clk:in std_logic;
    reset:in std_logic;
    interrupt:in std_logic;
    carry,zero:out std_logic);
end interrupt_store;
```

- *inCarry,inZero* – Carry & Zero signals returned from the Zero & Carry flag store.
- *CLK* – Clock signal;
- *Reset* – Reset signal;
- *Interrupt* – Interrupt signal;
- *Carry,Zero* – Carry & Zero signals returned to the Zero & Carry flag store.

7. ROM Instructions Memory

This component contains all the instructions of the micro-controller. When the user writes the instructions into the input file of the assembler, the assembler returns the memory map of the ROM Instructions Memory. This memory stores 256 instructions, each one represented by a 16-bit number. The ROM Memory ports are:

```
entity ROMMemory is --ROM Instruction Memory
    port(SEL: in std_logic_vector(7 downto 0);
    clk:in std_logic;
    Y: out std_logic_vector(15 downto 0));
end ROMMemory;

architecture ROM of ROMMemory is
type ROM_cell is array (0 to 255) of std_logic_vector(15 downto 0);
signal ROM_Memory:ROM_cell := ("");--Insert Memory Map
signal intY:std_logic_vector(15 downto 0);
begin
    intY<=ROM_Memory(conv_integer(sel));
    Y<=intY when clk'event and clk='1';
end architecture ROM;
```

- *SEL* – the number of the instruction;
- *CLK* – Clock signal;
- *Y* – The output instruction used in the micro-controller.

8. Debouncer

This component is used to filter the manual clock signal from the FPGA board and its signals are:

- *Button* – the signal from the button of the FPGA board.
- *CLK* – Clock signal;
- *Result* – The signal from the button, but filtered.

```
entity debouncer is
    port(button: in std_logic;
    clk: in std_logic;
    result: out std_logic);
end debouncer;

architecture a1 of debouncer is
signal ff: std_logic_vector(1 downto 0);
signal counter_set: std_logic;
signal counter_cont: std_logic_vector(19 downto 0);
begin
    counter_set<=ff(0) xor ff(1);
    process(clk)
    begin
        if clk'event and clk='1' then
            ff(0)<=button;
            ff(1)<=ff(0);
            if counter_set='1' then
                counter_cont<=(others=>'0');
            elsif(counter_cont(19)='0') then
                counter_cont<=counter_cont+1;
            else
                result<=ff(1);
            end if;
        end if;
    end process;
end;
```

9. LED BCD 7-segment Display

This component is used to show the contents of the register from the ALU Unit. It also shows on two LED signals if there are input or output signals, called read_strobe and write_strobe. The signals of the component are:

- *CLK* – Clock Signal
- *SW* – the number which will be on display
- *AN* – the anodes of the display

```
entity led_7_disp is  --LED Display
    port(clk:in std_logic;
    sw:in std_logic_vector(15 downto 0);
    an:out std_logic_vector(3 downto 0);
    seg:out std_logic_vector(0 to 6));
end led_7_disp;
```

- *SEG* – the segments of the display

Instructions

The micro-controller is user-friendly and it is implemented on the Basys-3 FPGA Board, but it can also be implemented on any other boards. The input and output signals of the device are:

- The swtiches V17, V16, W16, W17, W15, V15, W14, W13 represent the 8-bit number inserted by user in the ALU Unit.
- The swtich R2 represents the Interrupt signal.
- The switch T1 represents the Reset signal.
- The LEDs U16, E19, U19, V19, W18, U15, U14, V14 represent the number from the ALU Unit if there is an output instruction.
- The LED P1 represents WS – Write Strobe.
- The LED L1 represents RS – Read Strobe.
- The pins W7, W6, U8, V8, U5, V5, U7 represent the 7 segments of the BCD Display.
- The pins U2, U4, V4, W4 represent the anodes of the 7-segment BCD Display.
- The button U17 represents the Clock Button the user gives.
- The port W5 represents the Clock Quartz Oscillator with the frequency 100 MHz.

In order for the user to use the micro-controller, it has to follow these steps:

Step 1: Use the Assembler. The assembler is a program created using C++ programming language to encode all the instructions for the microcontroller program. The user has a defined set of instructions as input and the encoder will have as an output 256 instructions as 16-bit numbers which serve as the memory map for the Program ROM Memory.

The assembler is represented by: an executable(”AssemblerCPP.exe”), an input text file(”input.txt”) and an output text file(”output.txt”). The input commands have some special conditions:

- Every command must be written in lowercase letters.
- The conditions for the program control group must be written with uppercase letters (C,NZ,Z,NZ).
- The maximum number of commands in the input is 256. If the input text file contains less than 256 commands (e.g. n commands, n<256), the remaining 256-n slots are coded with ”0000000000000000”.

This microcontroller has different commands to execute. These are grouped in different categories, based on the different features of the device (Note: X, Y, kk, aa, pp take values in hexadecimal form):

- Program Control Group
  - ✓ *jump (Z/NZ/C/NC)* aa – the program counter goes to the address ”aa” (”aa” takes values from 00 to FF in hexadecimal);
  - ✓ *call (Z/NC/C/NC)* aa – the program calls the address ”aa” from the program control stack;
  - ✓ *return (UNC/Z/NC/C/NC)* aa – the program returns the address ”aa” from the program control stack;
- Logical Group
  - ✓ *load sX, (kk/sY)* – the command loads the content from the register Y or the constant kk into the register X;
  - ✓ *and sX, (kk/sY)* – the command does the ”and” logic function between the content of the register X and the constant kk/the content of the register Y;
  - ✓ *or sX, (kk/sY)* – the command does the ”or” logic function between the content of the register X and the constant kk/the content of the register Y;
  - ✓ *xor sX, (kk/sY)* – the command does the ”xor” logic function between the content of the register X and the constant kk/ the content of the register Y;
- Arithmetic Group
  - ✓ *add sX, (kk/sY)* – the command returns the value of the half-adder between the content of the register X with the constant kk/ the content of the register Y;
  - ✓ *addcy sX, (kk/sY)* – the command returns the value of the full-adder between the content of the register X with the constant kk/ the content of the register Y;
  - ✓ *sub sX, (kk/sY)* – the command returns the value of the half-subtractor between the content of the register X with the constant kk/ the content of the register Y;
  - ✓ *subcy sX, (kk/sY)* – the command returns the value of the full-subtractor between the content of the register X with the constant kk/ the content of the register Y;
- Shift and Rotate Group
  - ✓ *sr0 sX* – the command shifts the content of the register X to the right inserting the value ’0’ to the left side and the value out goes into CARRY;
  - ✓ *sr1 sX* – the command shifts the content of the register X to the right inserting the value ’1’ to the left side and the value out goes into CARRY;
  - ✓ *srx sX* – the command shifts the content of the register X to the right doubling the first bit from the left side and the value out goes into CARRY;
  - ✓ *sra sX* – the command shifts the content of the register X to the right, the value out going into CARRY and inserted to the left;
  - ✓ *ror sX* – the command rotates the content of the register X to the right, the first bit from the right side being copied into CARRY;
  - ✓ *sl0 sX* – the command shifts the content of the register X to the left inserting the value ’0’ to the right side and the value out goes into CARRY;

- ✓ *sl1 sX* – the command shifts the content of the register X to the left inserting the value ’1’ to the right side and the value out goes into CARRY;
- ✓ *slx sX* – the command shifts the content of the register X to the left doubling the first bit from the right side and the value out goes into CARRY;
- ✓ *sla sX* – the command shifts the content of the register X to the left, the value out going into CARRY and inserted to the right;
- ✓ *rol sX* – the command rotates the content of the register X to the left, the first bit from the left side being copied into CARRY;

➢ Input/Output Group

- ✓ *input sX*, (*pp/sY*) – the command enables that the value of register X is the constant pp or the value of the register Y;
- ✓ *output sX*, (*pp/sY*) – the command enables that the value of register X can be transferred into an external module with the port pp or the value of register Y;

➢ Interrupt Group

- ✓ *returni enable/disable* – the command returns the interrupted value from the program counter stack(if enabled);
- ✓ *enable/disable interrupt* – the command enables/disables the interrupt process into the program counter;
- ✓ *ff* - the command is the same as the ”jump” command, but for interrupt group.

Step 2: After using the assembler, the output is resembled by the 256 commands coded in 16-bit numbers. These should be copied into the file *proj.vhd* in the ROM Instructions Memory as the memory map.

Step 3: The program will be compiled and implemented on the FPGA Board using programs like Xilinx ISE Design or Vivado (depending on your FPGA Board Chipset and program needed).

Step 4: After compiling, the instructions will be executed by the FPGA Board on the clock cycles given the user. For each instruction, the user must press the Clock Button 2 times in order to an instruction from the ROM Instructions Memory to be executed. The first press gives the pulse to the Program Counter and second press of the button gives to all the other components, based on the instruction.

Step 5: If the interrupt signal is active, the program is interrupted until the interrupt signal is disabled.

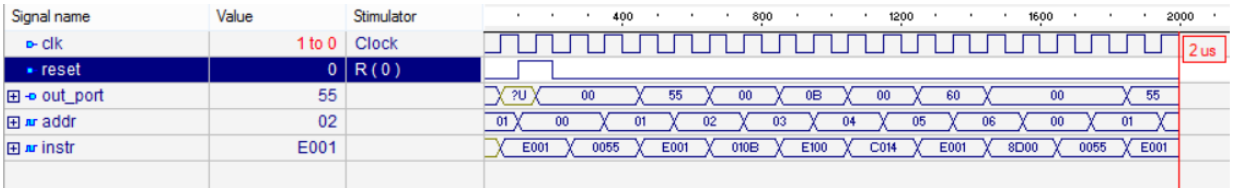
Step 6: If the reset signal is active, all the signals are reset and it will show ”0000” on the board.

Program example

For a better understanding of the microcontroller, we have attached a small program example, complete with simulation waveforms and pictures of the FPGA board running the program.

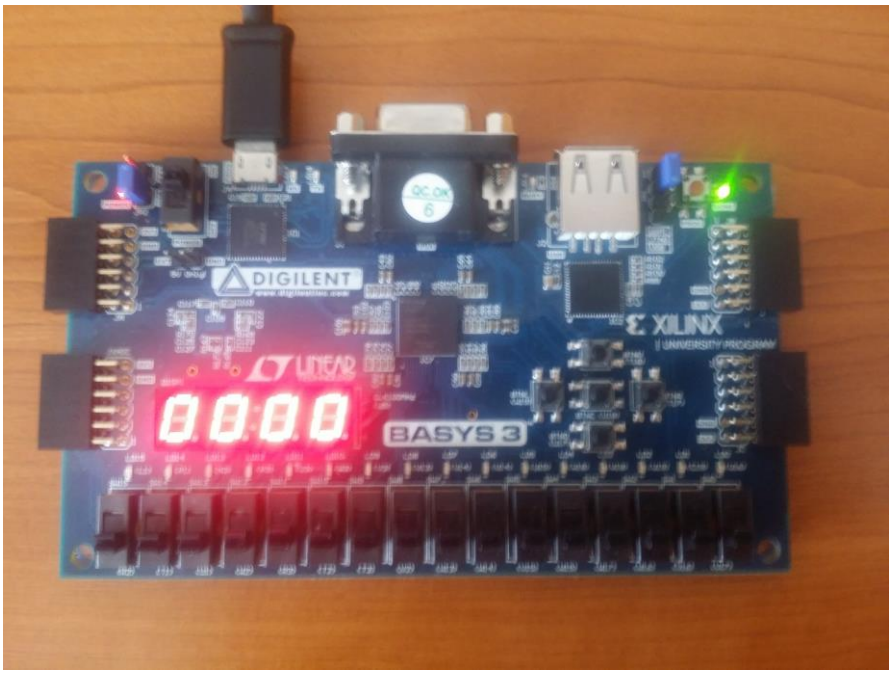
```
load s0 55
output s0 01
load s1 0B
output s1 00
add s0 s1
output s0 01
jump 00
```

This is the exemplified program. It loads 2 registers with values, outputs the values and then adds them and outputs the result. After that, it jumps to the first instruciton and restarts the whole process. To keep it simple, interrupts have not been enabled.

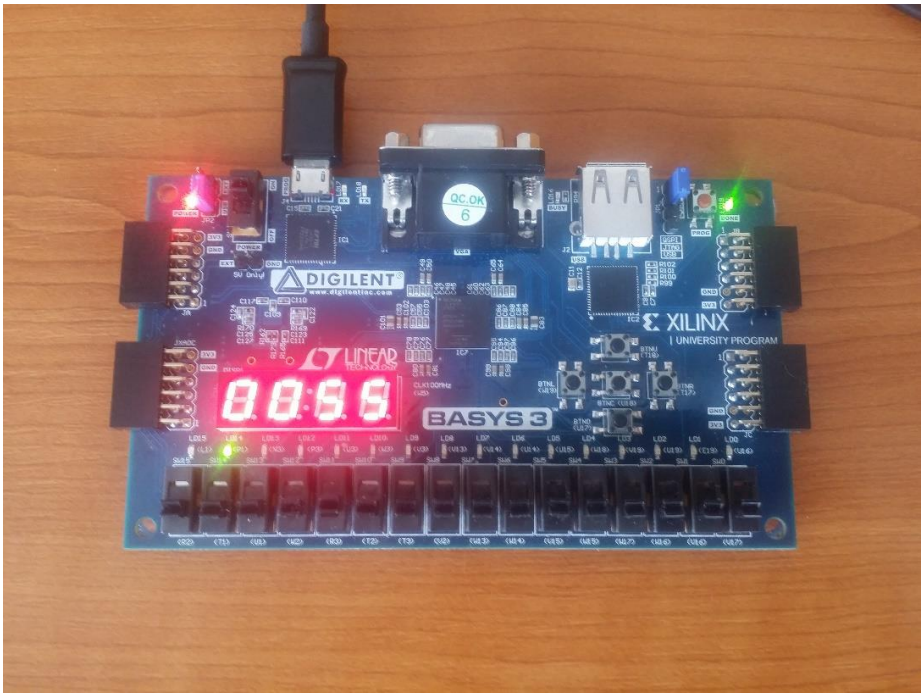


This is the waveform diagram from Active-HDL. As it can be noticed, the program does not work properly before activating the asynchronous reset signal. After that, we can observe a proper two-clock-cycle-per-instruction functionality. As explained before, addresses and instructions are delayed by one clock cycle. The outputs can be observed for two clock cycles on the out_port signal.

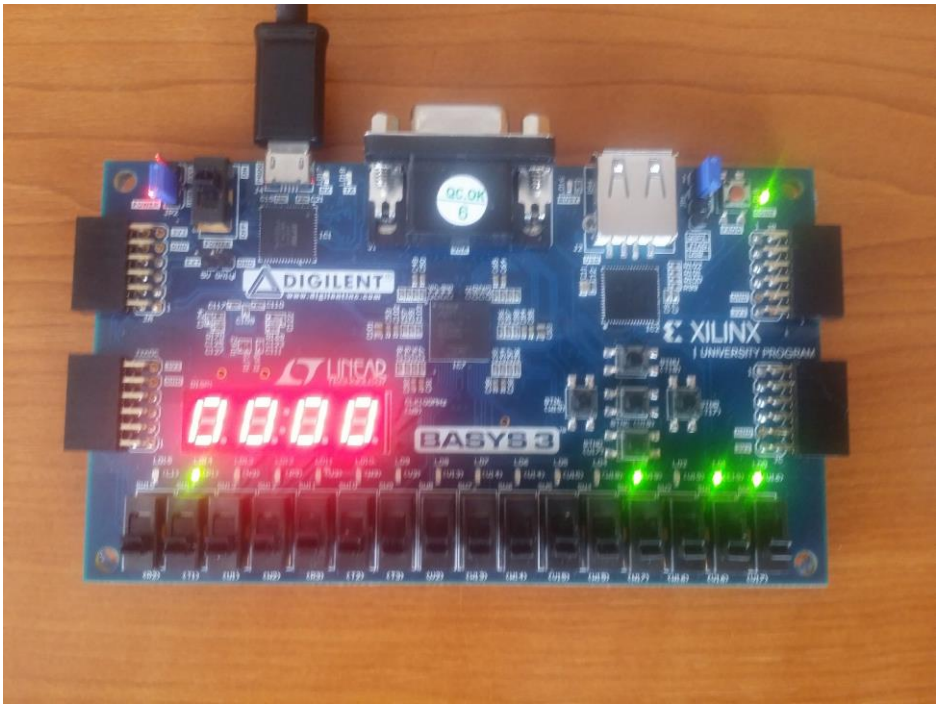
When tested on the FPGA board, this is the initial state. No LEDs are lit and the 7 segment display shows 0000. For initialization, it is not necessary to use the reset button.



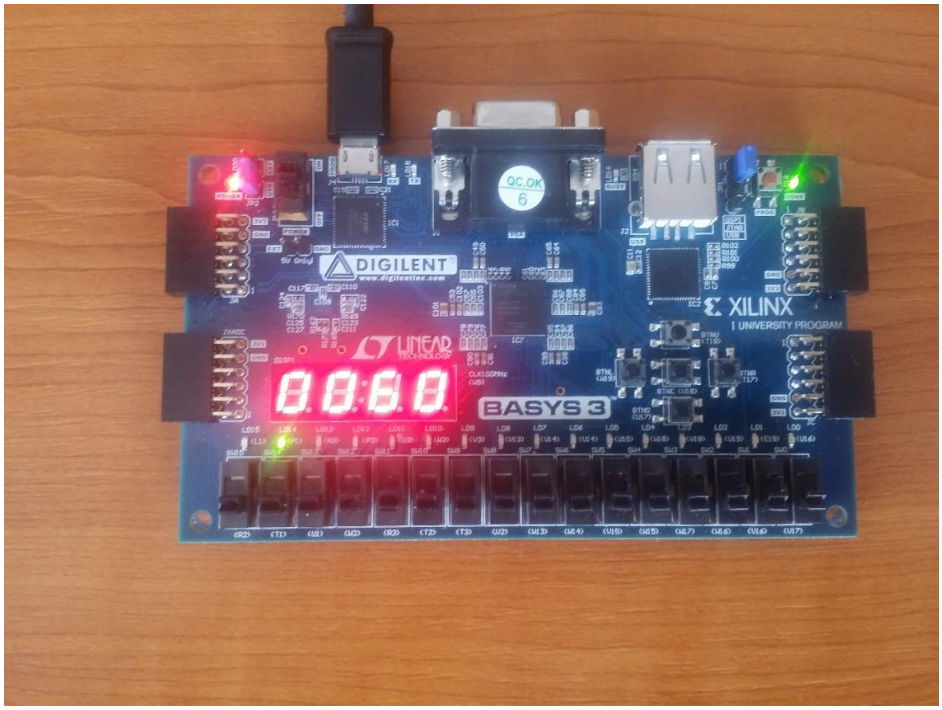
After a number of clock cycles (given by the user by pressing BTND), the first output instruction takes place and the content of the register s0 can be observed on the 7-segment-display (port 01). Notice that the Write Strobe is lit. The outputs on the 7-segment-display are only relevant when that LED is on. This was implemented in order to distinguish between the standby mode an the output instruction when 00 is displayed.



The second output instruction displays the contents of s1 on the board LEDs (port 00). The Write Strobe LED is active again



At this step, the sum between the contents of s0 and s1 is displayed on the 7-segment-display (port 01). The sum is stored in s0. The Write Strobe is lit again.



After this, the whole process starts again (because of the „jump 00” instruction, which causes the program counter to start from 00 again).

Solution justification

For this project we chose to stay as truthful as possible to the provided documentation, and because of that almost everything works as it would work in the PicoBlaze module provided by Xilinx. The microcontoller uses the same encoding for instructions as PicoBlaze and the instructions syntax is similar. In developing the microcontroller we adopted a bottom-up strategy, starting from the most basic components and working our way up. This lead to us using a behavioral description for basic components and a structural description for linking the modules together. Ocasionally we used used both descriptions, according to our needs. The behavioral description allows for an easy understanding of the functionalities of the basic modules, while the structural description shows better how the modules are linked. By doing this we made it easier both for us to design (and possibly improve) the code and for the user to understand it. For a better understanding we recommend checking the provided schematics and block diagrams.

The main development idea was that, while everything might work concurently, it does not mean that it should do that. Achieving reconfigurable sequential functionalities in a concurrent machine is not an easy task, but we did it with a lot of condition cheking. These conditions make sure that no operation has more effects than it should. For example, operations concerning address handling do not affect the values stored in the RAM memory, nor do they change the values stored in the flag store.

The assembler translates the instructions given by the user in bit strings (16-bit numbers) using basic string handling functions and a function that translates hexadecimal numbers in bit strings.

Possible improvements

- A possible improvement would be to optimize the microcontroller in order to make it efficient, increasing the working speed to that of the FPGA board (for Basys 3: 100 MHz). By achieving this, the microcontroller could easily work in compatibility with other modules in the FPGA board and the user would no longer be required to press a button in order to provide a clock signal.
- Another improvement could be to make interrupt response faster. In the current version it takes up to 3 clock cycles for the interrupt routine to intiate.

Annex 1 – *proj.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX2_1 is
	generic(pathwidth: integer);
	port(I0: in std_logic_vector(pathwidth-1 downto 0);
	I1: in std_logic_vector(pathwidth-1 downto 0);
	S0: in std_logic;
	O: out std_logic_vector(pathwidth-1 downto 0));
end MUX2_1;

architecture arh_MUX2_1 of MUX2_1 is
begin
	process(I0,I1,S0)
	variable intQ: std_logic_vector(pathwidth-1 downto 0);
	begin
		if S0='0' then
			intQ:=I0;
		else
			intQ:=I1;
		end if;
		O<=intQ;
	end process;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity I4_O1 is
	port(A: in std_logic_vector(3 downto 0);
	O: out std_logic);
end I4_O1;

architecture And_gate_test_1101 of I4_O1 is
begin
	O<=A(3) and A(2) and not(A(1)) and A(0);
end;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ent_ALU is
	port(A,B,Const: in std_logic_vector(7 downto 0);
	Instr: in std_logic_vector(15 downto 0);
	CIN: in std_logic;
	Y: out std_logic_vector(7 downto 0);
	Zero,Carry: out std_logic);
end;
```

```
architecture arh_ALU of ent_ALU is

component MUX2_1 is
        generic(pathwidth: integer:=1);
        port(I0: in std_logic_vector(pathwidth-1 downto 0);
        I1: in std_logic_vector(pathwidth-1 downto 0);
        S0: in std_logic;
        O: out std_logic_vector(pathwidth-1 downto 0));
end component;

component I4_O1 is
        port(A: in std_logic_vector(3 downto 0);
        O: out std_logic);
end component;

component ALU2 is
        port(A,B: in std_logic_vector(7 downto 0);
        CIN: in std_logic;
        SEL: in std_logic_vector(2 downto 0);
        Y: out std_logic_vector(7 downto 0);
        COUT: out std_logic;
        ZERO: out std_logic);
end     component;

component shift_reg is
        port(A:in std_logic_vector(7 downto 0);
        sel:in std_logic_vector(2 downto 0);
        B3:in std_logic;
        CarryI:in std_logic;
        R:out std_logic_vector(7 downto 0);
        Zero,CarryO:out std_logic);
end component;

signal V1,V2,V3: std_logic_vector(7 downto 0);
signal intZero0, intZero1: std_logic;
signal intCarry0, intCarry1: std_logic;
signal intSel,intSel1: std_logic;
signal intSel3b,intSel3b2: std_logic_vector(2 downto 0);

begin
        U0:Mux2_1
        generic map(pathwidth=>8)
        port map(Const,B,Instr(15),V1);
        U1:Mux2_1
        generic map(pathwidth=>3)
        port map(Instr(14 downto 12),Instr(2 downto 0),Instr(15),intSel3b);
        U2:Mux2_1
        generic map(pathwidth=>8)
        port map(V3,V2,intSel,Y);
        U3:Mux2_1
        generic map(pathwidth=>1)
        port map(I0(0)=>intZero0,I1(0)=>intZero1,S0=>intSel,O(0)=>Zero);
        U4:Mux2_1
        generic map(pathwidth=>1)
        port map(I0(0)=>intCarry0,I1(0)=>intCarry1,S0=>intSel,O(0)=>Carry);
        U5:I4_O1
        port map(Instr(15 downto 12),intSel);
        U6:ALU2
        port map(A,V1,CIN,intSel3b,V3,intCarry0,intZero0);
```

```
	U7:Shift_reg
	port map(A,Instr(2 downto 0),Instr(3),CIN,V2,intZero1,intCarry1);
	U8:Mux2_1
	generic map(pathwidth=>3)
	port map(intSel3b,"000",intSel1,intSel3b2);
	intSel1<='1' when instr(15 downto 13)="100" or instr(15 downto 13)="111" or
instr(15 downto 13)="101" else '0';
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity debouncer is
	port(button: in std_logic;
	clk: in std_logic;
	result: out std_logic);
end debouncer;

architecture a1 of debouncer is
signal ff: std_logic_vector(1 downto 0);
signal counter_set: std_logic;
signal counter_cont: std_logic_vector(19 downto 0);
begin
	counter_set<=ff(0) xor ff(1);
	process(clk)
	begin
		if clk'event and clk='1' then
			ff(0)<=button;
			ff(1)<=ff(0);
			if counter_set='1' then
				counter_cont<=(others=>'0');
			elsif(counter_cont(19)='0') then
				counter_cont<=counter_cont+1;
			else
				result<=ff(1);
			end if;
		end if;
	end process;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ALU2 is -- ALU2
	port(A,B: in std_logic_vector(7 downto 0);
	CIN: in std_logic;
	SEL: in std_logic_vector(2 downto 0);
	Y: out std_logic_vector(7 downto 0);
	COUT: out std_logic;
	ZERO: out std_logic);
end ALU2;

architecture operatii of ALU2 is
signal sum: std_logic_vector(8 downto 0);
signal intY: std_logic_vector(7 downto 0);
begin
```

```
	op: process(A,B,CIN,SEL)
	variable sum: std_logic_vector(8 downto 0);
	begin
		case SEL is
			when "000" => intY <= B;
			COUT<='0';
			when "001" => intY <= A and B;
			COUT<='0';
			when "010" => intY <= A or B;
			COUT<='0';
			when "011" => intY <= A xor B;
			COUT<='0';
			when "100" => sum := ('0'&A) + ('0'&B);
			COUT<=sum(8);
			intY<=sum(7 downto 0);
			when "101" => sum := ('0'&A) + ('0'&B) + CIN;
			COUT<=sum(8);
			intY<=sum(7 downto 0);
			when "110" => sum := ('0'&A) - ('0'&B);
			COUT<=sum(8);
			intY<=sum(7 downto 0);
			when "111" => sum := ('0'&A) - ('0'&B) - CIN;
			COUT<=sum(8);
			intY<=sum(7 downto 0);
			when others => sum := "000000000";
		end case;
	end process op;
	Y<=intY;
	Zero<='1' when intY="00000000" else '0';
end architecture operatii;

library ieee;
use ieee.std_logic_1164.all;

entity MUX8_1 is
	port(I0,I1,I2,I3,I4,I5,I6,I7:in std_logic;
	sel:in std_logic_vector(2 downto 0);
	R:out std_logic);
end MUX8_1;

architecture arh1 of MUX8_1 is
begin
	process(I0,I1,I2,I3,I4,I5,I6,I7,sel)
	begin
		case sel is
			when "000"=>R<=I0;
			when "001"=>R<=I1;
			when "010"=>R<=I2;
			when "011"=>R<=I3;
			when "100"=>R<=I4;
			when "101"=>R<=I5;
			when "110"=>R<=I6;
			when "111"=>R<=I7;
			when others=>R<='0';
		end case;
	end process;
end architecture arh1;

library ieee;
```

```
use ieee.std_logic_1164.all;

entity shift_reg is        --Shift-Register
        port(A:in std_logic_vector(7 downto 0);
        sel:in std_logic_vector(2 downto 0);
        B3:in std_logic;
        CarryI:in std_logic;
        R:out std_logic_vector(7 downto 0);
        Zero,CarryO:out std_logic);
end shift_reg;

architecture arh2 of shift_reg is

component MUX2_1 is
        generic(pathwidth: integer);
        port(I0: in std_logic_vector(pathwidth-1 downto 0);
        I1: in std_logic_vector(pathwidth-1 downto 0);
        S0: in std_logic;
        O: out std_logic_vector(pathwidth-1 downto 0));
end component;

component MUX8_1 is
        port(I0,I1,I2,I3,I4,I5,I6,I7:in std_logic;
        sel:in std_logic_vector(2 downto 0);
        R:out std_logic);
end component;

signal iR1,iR2,intCarry0,intCarry1:std_logic;
signal iR3,iR4,intR:std_logic_vector(7 downto 0);

begin
        U0:MUX8_1 port map(CarryI,'0',A(0),'0',A(7),'0','0','1',sel,iR1);        --shift right
        U1:MUX8_1 port map(CarryI,'0',A(7),'0',A(0),'0','0','1',sel,iR2);        --shift left
        iR3(6 downto 0)<=A(7 downto 1);   --shift left
        iR3(7)<=iR1;
        iR4(7 downto 1)<=A(6 downto 0);       --shift left
        iR4(0)<=iR2;
        U2:MUX2_1
        generic map(pathwidth=>8)
        port map(iR4,iR3,B3,intR);
        U3:MUX2_1
        generic map(pathwidth=>1)
        port map(I0(0)=>intCarry0,I1(0)=>intCarry1,S0=>B3,O(0)=>CarryO);
        intCarry0<=A(7);
        intCarry1<=A(0);
        R<=intR;
        process(intR)
        begin
                if intR="00000000" then Zero<='1'; else Zero<='0'; end if;
        end process;
end architecture arh2;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity num_princ is     --Program Counter
        port(clk: in std_logic;
```

```
        reset: in std_logic;
        le: in std_logic;
        interruptSig: in std_logic;
        addressI: in std_logic_vector(7 downto 0);
        addressO: out std_logic_vector(7 downto 0));
end num_princ;

architecture arh1 of num_princ is
signal intQ: std_logic_vector(7 downto 0):="00000000";
begin
        process(clk,reset)
        begin
                if reset='1' then
                        intQ<="00000000";
                else
                        if clk'event and clk='1' then
                                if le='1' then
                                        intQ<=addressI;
                                elsif interruptSig='1' then
                                        intQ<="11111111";
                                else
                                        intQ<=intQ+1;
                                end if;
                        end if;
                end if;
        end process;
        addressO<=intQ;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX4_1 is
        generic(N:integer);
        port(I0,I1,I2,I3:in std_logic_vector(N-1 downto 0);
        S:in std_logic_vector(1 downto 0);
        O:out std_logic_vector(N-1 downto 0));
end MUX4_1;

architecture arh of MUX4_1 is
begin
        process(I0,I1,I2,I3,S)
        variable intQ:std_logic_vector(N-1 downto 0);
        begin
                case S is
                        when "00"=>intQ:=I0;
                        when "01"=>intQ:=I1;
                        when "10"=>intQ:=I2;
                        when others=>intQ:=I3;
                end case;
                O<=intQ;
        end process;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;

entity cond_check is
        port(Carry,Zero:in std_logic;
        instr:in std_logic_vector(15 downto 0);
        Enable:out std_logic);
end cond_check;

architecture arh1 of cond_check is
component MUX2_1 is
        generic(pathwidth: integer);
        port(I0: in std_logic_vector(pathwidth-1 downto 0);
        I1: in std_logic_vector(pathwidth-1 downto 0);
        S0: in std_logic;
        O: out std_logic_vector(pathwidth-1 downto 0));
end component;
component MUX4_1 is
        generic(N:integer);
        port(I0,I1,I2,I3:in std_logic_vector(N-1 downto 0);
        S:in std_logic_vector(1 downto 0);
        O:out std_logic_vector(N-1 downto 0));
end component;
signal S1,S2,S3:std_logic;
begin
        U1:MUX2_1
        generic map(pathwidth=>1)
        port map(I0(0)=>'1',I1(0)=>S1,S0=>instr(12),O(0)=>Enable);
        U2:MUX4_1
        generic map(N=>1)
        port    map(I0(0)=>Zero,I1(0)=>S2,I2(0)=>Carry,I3(0)=>S3,S=>instr(11    downto
10),O(0)=>S1);
        S2<=not(Zero);
        S3<=not(Carry);
end architecture arh1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity num_complet is
        port(instr:in std_logic_vector(15 downto 0);
        addrExt,addrSt:in std_logic_vector(7 downto 0);
        Carry,Zero:in std_logic;
        reset:in std_logic;
        interruptSig: in std_logic;
        clk:in std_logic;
        Output:out std_logic_vector(7 downto 0));
end num_complet;

architecture arh1 of num_complet is

component cond_check is
        port(Carry,Zero:in std_logic;
        instr:in std_logic_vector(15 downto 0);
        Enable:out std_logic);
end component;

component num_princ is
```

```
        port(clk: in std_logic;
        reset:in std_logic;
        le: in std_logic;
        interruptSig: in std_logic;
        addressI: in std_logic_vector(7 downto 0);
        addressO: out std_logic_vector(7 downto 0));
end component;

component MUX2_1 is
        generic(pathwidth: integer);
        port(I0: in std_logic_vector(pathwidth-1 downto 0);
        I1: in std_logic_vector(pathwidth-1 downto 0);
        S0: in std_logic;
        O: out std_logic_vector(pathwidth-1 downto 0));
end component;

signal intAddr:std_logic_vector(7 downto 0);
signal S2,S3,S4:std_logic;

begin
        U1:MUX2_1
        generic map(pathwidth=>8)
        port map(addrSt,addrExt,instr(8),intAddr);
        U2:cond_check port map(carry,Zero,instr,S2);
        U3:num_princ port map(Clk,reset,S3,interruptSig,intAddr,Output);
        S4<=instr(15) and not(instr(14)) and not(instr(13));
        S3<=S4 and S2 when instr/="1000000000110000" and
instr/="1000000000010000" else '0';
end architecture arh1;

library ieee;
use ieee.std_logic_1164.all;

entity clk_div is
        port(clk:in std_logic;
        O0,O1:out std_logic);
end clk_div;

architecture arh1 of clk_div is
signal intQ:std_logic:='0';
begin
        process(clk)
        begin
                if clk'event and clk='1' then
                        intQ<=not(intQ);
                end if;
        end process;
        O0<=intQ;
        O1<=not(intQ);
end architecture arh1;

library ieee;
use ieee.std_logic_1164.all;

entity storage is
        port(instr:in std_logic_vector(15 downto 0);
        clk: in std_logic;
        reset:in std_logic;
        addr:in std_logic_vector(7 downto 0);
```

```
        interruptEnable: out std_logic);
end storage;

architecture arh1 of storage is
signal intQ:std_logic;
begin
        process(clk,reset)
        begin
                if reset='1' then
                        intQ<='0';
                else
                        if clk'event and clk='1' then
                                if                instr="1000000011110000"                or
instr="1000000000110000" then
                                        intQ<='1';
                                elsif                instr="1000000011010000"                or
instr="1000000000010000" or addr=”11111111” then
                                        intQ<='0';
                                end if;
                        end if;
                end if;
        end process;
        interruptEnable<=intQ;
end architecture arh1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter4bit is  --Counter from stack
        port(CU: in std_logic;
        CE: in std_logic;
        reset:in std_logic;
        clk: in std_logic;
        Y: out std_logic_vector(3 downto 0));
end counter4bit;

architecture numar of counter4bit is
signal intQ: std_logic_vector(3 downto 0):= "0000";
begin
        numara: process(clk,reset,ce)
        begin
                if reset='1' then
                        intQ<="0000";
                else
                        if clk'event and clk='1' then
                                if(CE = '1') then
                                        if(CU = '1') then
                                                intQ<=intQ + 1;
                                        else
                                                intQ<=intQ - 1;
                                        end if;
                                end if;
                        end if;
                end if;
        end process numara;
        Y<=intQ;
end architecture numar;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity stack is    --Stack RAM
        port(WE: in std_logic;
        A: in std_logic_vector(7 downto 0);
        SEL: in std_logic_vector(3 downto 0);
        CLK: in std_logic;
        Y: out std_logic_vector(7 downto 0));
end stack;

architecture stack_RAM of stack is
type RAM_cell is array (15 downto 0) of std_logic_vector(7 downto 0);
signal RAM_stack: RAM_cell;
begin
        stiva: process(CLK,WE,A)
        begin
                if(CLK'EVENT) and CLK = '1'  then
                        if WE = '1' then
                                RAM_stack(conv_integer(SEL+1))<=A;
                        end if;
                Y<=RAM_stack(conv_integer(SEL));
                end if;
        end process stiva;
end architecture stack_RAM;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity complete_stack is                --Program Counter Stack
        port(instr:in std_logic_vector(15 downto 0);
        Cont:in std_logic_vector(7 downto 0);
        Carry,Zero:in std_logic;
        reset:in std_logic;
        interrupt:in std_logic;
        clk:in std_logic;
        Output: out std_logic_vector(7 downto 0));
end complete_stack;


architecture arh1 of complete_stack is
component stack is
        port(WE: in std_logic;
        A: in std_logic_vector(7 downto 0);
        SEL: in std_logic_vector(3 downto 0);
        CLK: in std_logic;
        Y: out std_logic_vector(7 downto 0));
end component;
component CE is
        port(A,B,C,D,E: in std_logic;
        Y: out std_logic);
end component;
component counter4bit is
```

```
        port(CU: in std_logic;
        CE: in std_logic;
        reset:in std_logic;
        clk: in std_logic;
        Y: out std_logic_vector(3 downto 0));
end component;
component cond_check is
        port(Carry,Zero:in std_logic;
        instr:in std_logic_vector(15 downto 0);
        Enable:out std_logic);
end component;
signal s1,s3,s4,s5,s6,s7,s8:std_logic;
signal s2:std_logic_vector(3 downto 0);
begin
        U0:CE port map(instr(15),instr(14),instr(13),instr(9),instr(8),s1);
        U1:counter4bit port map(s6,s8,reset,clk,s2);
        U2:stack port map(s5,Cont,s2,clk,Output);
        U3:cond_check port map(Carry,Zero,instr,s3);
        s4<=s1 and s3;
        s5<=instr(9) or interrupt;
        s6<=instr(8) or interrupt;
        check_returni:process(instr)
        begin
                if instr="1000000011110000" or instr="1000000011010000" then
                        s7<='1';
                else
                        s7<='0';
                end if;
        end process;
        s8<=s4 or interrupt or s7;
end architecture arh1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ROMMemory is --ROM Instruction Memory
        port(SEL: in std_logic_vector(7 downto 0);
        clk:in std_logic;
        Y: out std_logic_vector(15 downto 0));
end ROMMemory;

architecture ROM of ROMMemory is
type ROM_cell is array (0 to 255) of std_logic_vector(15 downto 0);
signal ROM_Memory:ROM_cell := ();
signal intY:std_logic_vector(15 downto 0);
begin
        intY<=ROM_Memory(conv_integer(sel));
        Y<=intY when clk'event and clk='1';
end architecture ROM;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity program_counter is     -- Program Flow Control
        port(instr:in std_logic_vector(15 downto 0);
```

```
        Carry,Zero:in std_logic;
        reset:in std_logic;
        interrupt:in std_logic;
        clk:in std_logic;
        addr:out std_logic_vector(7 downto 0));
end program_counter;

architecture arh of program_counter is
component complete_stack is
        port(instr:in std_logic_vector(15 downto 0);
        Cont:in std_logic_vector(7 downto 0);
        Carry,Zero:in std_logic;
        reset:in std_logic;
        interrupt:in std_logic;
        clk:in std_logic;
        Output: out std_logic_vector(7 downto 0));
end component;

component num_complet is
        port(instr:in std_logic_vector(15 downto 0);
        addrExt,addrSt:in std_logic_vector(7 downto 0);
        Carry,Zero:in std_logic;
        reset:in std_logic;
        interruptSig: in std_logic;
        clk:in std_logic;
        Output:out std_logic_vector(7 downto 0));
end component;

signal s1,s2,s3:std_logic_vector(7 downto 0);
begin
        U0:num_complet port map(instr,instr(7 downto
0),s3,Carry,Zero,reset,interrupt,clk,s2);
        U1:complete_stack port map(instr,s2,Carry,Zero,reset,interrupt,clk,s1);
        addr<=s2;
        s3<=s1+1;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity clk200Hz is
    Port (
        clk_in : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end clk200Hz;

architecture Behavioral of clk200Hz is
    signal temporal: STD_LOGIC;
    signal counter : integer range 0 to 124999 := 0;
begin
    frequency_divider: process (reset, clk_in) begin
        if (reset = '1') then
            temporal <= '0';
            counter <= 0;
        elsif rising_edge(clk_in) then
```

```
            if (counter = 124999) then
                temporal <= NOT(temporal);
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    clk_out <= temporal;
end Behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX4_1 is
        generic(N:integer);
        port(I0,I1,I2,I3:in std_logic_vector(N-1 downto 0);
        S:in std_logic_vector(1 downto 0);
        O:out std_logic_vector(N-1 downto 0));
end MUX4_1;

architecture arh of MUX4_1 is
begin
        process(I0,I1,I2,I3,S)
        variable intQ:std_logic_vector(N-1 downto 0);
        begin
                case S is
                        when "00"=>intQ:=I0;
                        when "01"=>intQ:=I1;
                        when "10"=>intQ:=I2;
                        when others=>intQ:=I3;
                end case;
                O<=intQ;
        end process;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity conv_to_bcd is
        port(nrIN1,nrIN2,nrIN3,nrIN4: in std_logic_vector(3 downto 0);
        nrOut1,nrOut2,nrOut3,nrOut4: out std_logic_vector(6 downto 0));
end conv_to_bcd;

architecture arh of conv_to_bcd is
type cont is array(0 to 15) of std_logic_vector(0 to 6);
signal                                                                          intQ:
cont:=("0000001","1001111","0010010","0000110","1001100","0100100","0100000","00
01111","0000000","0000100","0001000","1100000","0110001","1000010","0110000","01
11000");
begin
        nrOut1<=intQ(conv_integer(nrIN1));
        nrOut2<=intQ(conv_integer(nrIN2));
        nrOut3<=intQ(conv_integer(nrIN3));
```

```
	nrOut4<=intQ(conv_integer(nrIN4));
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity num2bit is
	port(clk: in std_logic;
	Q:out std_logic_vector(1 downto 0));
end num2bit;

architecture arh of num2bit is
signal intQ: std_logic_vector(1 downto 0):="00";
begin
	process(clk)
	begin
		if(clk'event and clk='1') then
			intQ<=intQ+1;
		end if;
	end process;
	Q<=intQ;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity DMUX1_4 is
	port(I:in std_logic;
	sel:in std_logic_vector(1 downto 0);
	O:out std_logic_vector(3 downto 0));
end DMUX1_4;

architecture arh1 of DMUX1_4 is
begin
	process(I,sel)
	begin
		if(sel="00") then
			O(0)<=I;
			O(3 downto 1)<="000";
		elsif(sel="01") then
			O(0)<='0';
			O(1)<=I;
			O(3 downto 2)<="00";
		elsif(sel="10") then
			O(1 downto 0)<="00";
			O(2)<=I;
			O(3)<='0';
		elsif(sel="11") then
			O(2 downto 0)<="000";
			O(3)<=I;
		end if;
	end process;
end architecture arh1;

library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity led_7_disp is  --LED Display
        port(clk:in std_logic;
        sw:in std_logic_vector(15 downto 0);
        an:out std_logic_vector(3 downto 0);
        seg:out std_logic_vector(0 to 6));
end led_7_disp;

architecture arh of led_7_disp is
component conv_to_bcd is
        port(nrIN1,nrIN2,nrIN3,nrIN4: in std_logic_vector(3 downto 0);
        nrOut1,nrOut2,nrOut3,nrOut4: out std_logic_vector(6 downto 0));
end component;
component num2bit is
        port(clk: in std_logic;
        Q:out std_logic_vector(1 downto 0));
end component;
component DMUX1_4 is
        port(I:in std_logic;
        sel:in std_logic_vector(1 downto 0);
        O:out std_logic_vector(3 downto 0));
end component;
component MUX4_1 is
        generic(N:integer);
        port(I0,I1,I2,I3:in std_logic_vector(N-1 downto 0);
        S:in std_logic_vector(1 downto 0);
        O:out std_logic_vector(N-1 downto 0));
end component;
component clk200Hz is
    Port (
        clk_in : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end component;
signal int:std_logic;
signal intQ0,intQ1,intQ2,intQ3:std_logic_vector(0 to 6);
signal sel:std_logic_vector(1 downto 0);
signal an1:std_logic_vector(3 downto 0);
begin
        U0:conv_to_bcd port map(sw(15 downto 12),sw(11 downto 8),sw(7 downto
4),sw(3 downto 0),intQ3,intQ2,intQ1,intQ0);
        U1:num2bit port map(int,sel);
        U2:DMUX1_4 port map('1',sel,an1);
        U3:MUX4_1
        generic map(N=>7)
        port map(intQ0,intQ1,intQ2,intQ3,sel,seg);
        U4:clk200Hz port map(clk,'0',clk_out=>int);
        an<=not(an1);
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;

entity read_strobe is
        port(instr:in std_logic_vector(15 downto 0);
```

```
        O:out std_logic);
end read_strobe;

architecture arh of read_strobe is
begin
        O<='1' when instr(15 downto 13)="101" else '0';
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;

entity write_strobe is
        port(instr:in std_logic_vector(15 downto 0);
        O:out std_logic);
end write_strobe;

architecture arh of write_strobe is
begin
        O<='1' when instr(15 downto 13)="111" else '0';
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


package reg_package is
type reg_cell is array (15 downto 0) of std_logic_vector(7 downto 0);
end package reg_package;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.reg_package.all;

entity register_ent is
        port(input: in std_logic_vector(7 downto 0);
        reg_select1: in std_logic_vector(3 downto 0);
        reg_select2: in std_logic_vector(3 downto 0);
        clk: in std_logic;
        reset:in std_logic;
        output1: out std_logic_vector(7 downto 0);
        output2: out std_logic_vector(7 downto 0));
end register_ent;

architecture arh1 of register_ent is
signal internal_cont: reg_cell;
begin
        process(clk,reset)
        begin
                if reset='1' then
                        internal_cont<=(others=>"00000000");
                else
                        if clk'event and clk='1' then
                                internal_cont(conv_integer(reg_select1))<=input;
                        end if;
                end if;
```

```
        end process;
        output1<=internal_cont(conv_integer(reg_select1));
        output2<=internal_cont(conv_integer(reg_select2));
end architecture arh1;

library ieee;
use ieee.std_logic_1164.all;

entity mem_alu is
        port(in_port:in std_logic_vector(7 downto 0);
        instr:in std_logic_vector(15 downto 0);
        Cin:in std_logic;
        clk:in std_logic;
        reset:in std_logic;
        zero:out std_logic;
        carry:out std_logic;
        out_port:out std_logic_vector(7 downto 0);
        port_id:out std_logic_vector(7 downto 0));
end mem_alu;

architecture arh of mem_alu is
component register_ent is
        port(input: in std_logic_vector(7 downto 0);
        reg_select1: in std_logic_vector(3 downto 0);
        reg_select2: in std_logic_vector(3 downto 0);
        clk: in std_logic;
        reset:in std_logic;
        output1: out std_logic_vector(7 downto 0);
        output2: out std_logic_vector(7 downto 0));
end component;
component ent_ALU is
        port(A,B,Const: in std_logic_vector(7 downto 0);
        Instr: in std_logic_vector(15 downto 0);
        CIN: in std_logic;
        Y: out std_logic_vector(7 downto 0);
        Zero,Carry: out std_logic);
end component;
component MUX2_1 is
        generic(pathwidth: integer);
        port(I0: in std_logic_vector(pathwidth-1 downto 0);
        I1: in std_logic_vector(pathwidth-1 downto 0);
        S0: in std_logic;
        O: out std_logic_vector(pathwidth-1 downto 0));
end component;
signal A,B,I,Y:std_logic_vector(7 downto 0);
signal Sem:std_logic_vector(3 downto 0);
signal en,S3,S4:std_logic;
begin
        U0:ent_ALU port map(A,B,instr(7 downto 0),instr,Cin,Y,zero,carry);
        U1:MUX2_1
        generic map(pathwidth=>8)
        port map(instr(7 downto 0),B,instr(12),port_id);
        en<='1' when instr(15 downto 13)="111" else '0';
        out_port<=A when en='1' else "00000000";
        U5:MUX2_1
        generic map(pathwidth=>8)
        port map(Y,in_port,S3,I);
        S3<='1' when instr(15 downto 13)="101" else '0';
        U6:MUX2_1
```

```
	generic map(pathwidth=>4)
	port map(instr(11 downto 8),instr(7 downto 4),S4,Sem);
	S4<='0' when (instr(15 downto 13)="100" or instr(15 downto 13)="101" or
instr(15 downto 13)="111") else '1';
	U7:register_ent port map(I,instr(11 downto 8),Sem,clk,reset,A,B);
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;

entity flg_store is
	port(reset:in std_logic;
	interrupt:in std_logic;
	inpCarry,inpZero:in std_logic;
	retCarry,retZero:in std_logic;
	instr:in std_logic_vector(15 downto 0);
	clk:in std_logic;
	carry,zero:out std_logic);
end flg_store;

architecture arh of flg_store is
component MUX2_1 is
	generic(pathwidth: integer);
	port(I0: in std_logic_vector(pathwidth-1 downto 0);
	I1: in std_logic_vector(pathwidth-1 downto 0);
	S0: in std_logic;
	O: out std_logic_vector(pathwidth-1 downto 0));
end component;
signal ce, ce1, intC, intZ, sig1, iC, iZ:std_logic;
begin
	process(clk,ce,reset,interrupt)
	begin
		if reset='1' or interrupt='1' then
			intC<='0';
			intZ<='0';
		else
			if interrupt='1' then
				if clk'event and clk='1' then
					intC<='0';
					intZ<='0';
				end if;
			elsif ce='1' then
				if clk'event and clk='1' then
					intC<=iC;
					intZ<=iZ;
				end if;
			end if;
		end if;
	end process;
	U0:Mux2_1
	generic map(pathwidth=>1)
	port map(I0(0)=>inpCarry,I1(0)=>retCarry,S0=>sig1,O(0)=>iC);
	U1:Mux2_1
	generic map(pathwidth=>1)
	port map(I0(0)=>inpZero,I1(0)=>retZero,S0=>sig1,O(0)=>iZ);
	carry<=intC;
	zero<=intZ;
	sig1<='1' when instr="1000000011110000" or instr="1000000011010000" else
'0';
```

```
	ce<=ce1 or sig1;
	ce1<='0' when instr(15 downto 13)="100" or instr(15 downto 13)="111" or instr(15 downto 13)="101" or instr(15 downto 12)="0000" or (instr(15 downto 12)="1100" and instr(3 downto 0)="0000") else '1';
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;

entity interrupt_store is
	port(inCarry, inZero:in std_logic;
	clk:in std_logic;
	reset:in std_logic;
	interrupt:in std_logic;
	carry,zero:out std_logic);
end interrupt_store;

architecture arh of interrupt_store is
signal ce, intC, intZ:std_logic;
begin
	process(reset,ce,clk)
	begin
		if reset='1' then
			intC<='0';
			intZ<='0';
		else
			if ce='1' then
				if clk'event and clk='1' then
					intC<=inCarry;
					intZ<=inZero;
				end if;
			end if;
		end if;
	end process;
	carry<=intC;
	zero<=intZ;
	ce<=interrupt;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;

entity comb is
	port(clk:in std_logic;
	reset:in std_logic;
	interrupt:in std_logic;
	instr:in std_logic_vector(15 downto 0);
	inCarry,inZero:in std_logic;
	carry,zero:out std_logic);
end comb;

architecture arh of comb is
component interrupt_store is
	port(inCarry, inZero:in std_logic;
	clk:in std_logic;
	reset:in std_logic;
	interrupt:in std_logic;
	carry,zero:out std_logic);
end component;
```

```
component flg_store is
	port(reset:in std_logic;
	interrupt:in std_logic;
	inpCarry,inpZero:in std_logic;
	retCarry,retZero:in std_logic;
	instr:in std_logic_vector(15 downto 0);
	clk:in std_logic;
	carry,zero:out std_logic);
end component;
signal retCarry, retZero, outCarry, outZero:std_logic;
begin
	U0:flg_store port
map(reset,interrupt,inCarry,inZero,retCarry,retZero,instr,clk,outCarry,outZero);
	U1:interrupt_store port
map(outCarry,outZero,clk,reset,interrupt,retCarry,retZero);
	carry<=outCarry;
	zero<=outZero;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity micro is
	port(in_port:in std_logic_vector(7 downto 0);
	clk:in std_logic;
	interrupt:in std_logic;
	reset:in std_logic;
	port_id:out std_logic_vector(7 downto 0);
	out_port:out std_logic_vector(7 downto 0);
	RS:out std_logic;
	WS:out std_logic);
end micro;

architecture arh of micro is
component mem_alu is
	port(in_port:in std_logic_vector(7 downto 0);
	instr:in std_logic_vector(15 downto 0);
	Cin:in std_logic;
	clk:in std_logic;
	reset:in std_logic;
	zero:out std_logic;
	carry:out std_logic;
	out_port:out std_logic_vector(7 downto 0);
	port_id:out std_logic_vector(7 downto 0));
end component;
component program_counter is
	port(instr:in std_logic_vector(15 downto 0);
	Carry,Zero:in std_logic;
	reset:in std_logic;
	interrupt:in std_logic;
	clk:in std_logic;
	addr:out std_logic_vector(7 downto 0));
end component;
component clk_div is
	port(clk:in std_logic;
	O0,O1:out std_logic);
end component;
```

```
component storage is
	port(instr:in std_logic_vector(15 downto 0);
	clk: in std_logic;
	reset:in std_logic;
	addr:in std_logic_vector(7 downto 0);
	interruptEnable: out std_logic);
end component;
component ROMMemory is
	port(SEL: in std_logic_vector(7 downto 0);
	clk:in std_logic;
	Y: out std_logic_vector(15 downto 0));
end component;
component read_strobe is
	port(instr:in std_logic_vector(15 downto 0);
	O:out std_logic);
end	component;
component write_strobe is
	port(instr:in std_logic_vector(15 downto 0);
	O:out std_logic);
end component;
component comb is
	port(clk:in std_logic;
	reset:in std_logic;
	interrupt:in std_logic;
	instr:in std_logic_vector(15 downto 0);
	inCarry,inZero:in std_logic;
	carry,zero:out std_logic);
end component;
signal addr:std_logic_vector(7 downto 0);
signal clk1,clk2,C,Z,oZ,oC,interruptEnable,INTinterrupt:std_logic;
signal instr:std_logic_vector(15 downto 0);
begin
	U0:ROMMemory port map(addr,clk1,instr);
	U1:mem_alu port map(in_port,instr,C,clk1,reset,oZ,oC,out_port,port_id);
	U2:program_counter port map(instr,C,Z,reset,INTinterrupt,clk2,addr);
	U3:clk_div port map(clk,clk1,clk2);
	U4:storage port map(instr,clk1,reset,instr,interruptEnable);
	U5:read_strobe port map(instr,RS);
	U6:write_strobe port map(instr,WS);
	U7:comb port map(clk1,reset,INTinterrupt,instr,oC,oZ,C,Z);
	INTinterrupt<=interruptEnable and interrupt;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity DMUXspec2_1 is
	port(I:in std_logic_vector(7 downto 0);
	S:in std_logic;
	O0:out std_logic_vector(7 downto 0);
	O1:out std_logic_vector(7 downto 0));
end DMUXspec2_1;

architecture arh of DMUXspec2_1 is
begin
	process(I,S)
	begin
```

```
		if S='0' then
			O0<=I;
			O1<="00000000";
		else
			O0<="00000000";
			O1<=I;
		end if;
	end process;
end architecture arh;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity project is  --The project itself
	port(clk:in std_logic;
	button:in std_logic;
	sw:in std_logic_vector(7 downto 0);
	interrupt:in std_logic;
	reset:in std_logic;
	RS,WS:out std_logic;
	led:out std_logic_vector(7 downto 0);
	an:out std_logic_vector(3 downto 0);
	seg:out std_logic_vector(0 to 6));
end project;

architecture arh of project is
component micro is
	port(in_port:in std_logic_vector(7 downto 0);
	clk:in std_logic;
	interrupt:in std_logic;
	reset:in std_logic;
	port_id:out std_logic_vector(7 downto 0);
	out_port:out std_logic_vector(7 downto 0);
	RS:out std_logic;
	WS:out std_logic);
end component;
component DMUXspec2_1 is
	port(I:in std_logic_vector(7 downto 0);
	S:in std_logic;
	O0:out std_logic_vector(7 downto 0);
	O1:out std_logic_vector(7 downto 0));
end component;
component MUX2_1 is
	generic(pathwidth: integer);
	port(I0: in std_logic_vector(pathwidth-1 downto 0);
	I1: in std_logic_vector(pathwidth-1 downto 0);
	S0: in std_logic;
	O: out std_logic_vector(pathwidth-1 downto 0));
end component;
component debouncer is
	port(button: in std_logic;
	clk: in std_logic;
	result: out std_logic);
end component;
component led_7_disp is
	port(clk:in std_logic;
	sw:in std_logic_vector(15 downto 0);
```

```
        an:out std_logic_vector(3 downto 0);
        seg:out std_logic_vector(0 to 6));
    end component;
    signal port_id, in_port, out_port, switch: std_logic_vector(7 downto 0);
    signal intCLK: std_logic;
    begin
        U0:MUX2_1
        generic map(pathwidth=>8)
        port map(sw,"00001111",port_id(0),in_port);
        U1:debouncer port map(button,clk,intCLK);
        U2:micro port map(in_port,intCLK,interrupt,reset,port_id,out_port,RS,WS);
        U3:DMUXspec2_1 port map(out_port,port_id(0),led,switch);
        U4:led_7_disp port map(clk,sw(7 downto 0)=>switch,sw(15 downto
    8)=>"00000000",an=>an,seg=>seg);
    end architecture arh;

    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
    use ieee.std_logic_unsigned.all;

    entity CE is
        port(A,B,C,D,E: in std_logic;
        Y: out std_logic);
    end CE;

    architecture A of CE is
    signal int1: std_logic;
    signal int2: std_logic;
    begin
        int1<=A and (B nor C);
        int2<=(not D) nand E;
        Y<=int1 and int2;
    end architecture A;
```

Annex 2 – *constr.xdc*

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
```

```
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN T1 [get_ports reset]
        set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN R2 [get_ports interrupt]
        set_property IOSTANDARD LVCMOS33 [get_ports interrupt]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN P1 [get_ports WS]
        set_property IOSTANDARD LVCMOS33 [get_ports WS]
set_property PACKAGE_PIN L1 [get_ports RS]
        set_property IOSTANDARD LVCMOS33 [get_ports RS]

##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
```

```
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
set_property PACKAGE_PIN U17 [get_ports button]
	set_property IOSTANDARD LVCMOS33 [get_ports button]
```

Annex 3 – *assembler.cpp*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* convFromHex(char str[], int length);

int main()
{
   FILE *pf1;
   FILE *pf2;
   int i;
   pf1=fopen("input.txt","r");
   pf2=fopen("output.txt","w");
   char instr[256][17];
   char cons[17];
   strcpy(cons,"0000000000000000");
   for(i=0;i<256;i++){
      strcpy(instr[i],cons);
   }
   int line=-1;
   char str1[16],str2[16],str3[16];
   while(fscanf(pf1,"%s",str1)>0){
      line++;
      if(line>255){
         printf("Error: Too many instructions!");
         getchar();
         exit(1);
      }
      if(strcmp(str1,"jump")==0){
         strcpy(instr[line],"100");
         fscanf(pf1,"%s",str2);

if((strcmp(str2,"Z")==0)||(strcmp(str2,"NZ")==0)||(strcmp(str2,"C")==0)||(strcmp(str2,"NC"
)==0)){
            strcat(instr[line],"1");
            fscanf(pf1,"%s",str3);
            strcpy(str3,convFromHex(str3,2));
            if(strcmp(str2,"Z")==0)
               strcat(instr[line],"00");
            else if(strcmp(str2,"NZ")==0)
               strcat(instr[line],"01");
            else if(strcmp(str2,"C")==0)
               strcat(instr[line],"10");
```

```
            else if(strcmp(str2,"NC")==0)
                strcat(instr[line],"11");
            strcat(instr[line],"01");
            strcat(instr[line],str3);
        }
        else{
            strcat(instr[line],"011");
            strcpy(str2,convFromHex(str2,2));
            strcat(instr[line],"01");
            strcat(instr[line],str2);
        }
        instr[line][16]='\0';
    }
    else if(strcmp(str1,"call")==0){
        strcpy(instr[line],"100");
        fscanf(pf1,"%s",str2);

if((strcmp(str2,"Z")==0)||(strcmp(str2,"NZ")==0)||(strcmp(str2,"C")==0)||(strcmp(str2,"NC"
)==0)){
            strcat(instr[line],"1");
            fscanf(pf1,"%s",str3);
            strcpy(str3,convFromHex(str3,2));
            if(strcmp(str2,"Z")==0)
                strcat(instr[line],"00");
            else if(strcmp(str2,"NZ")==0)
                strcat(instr[line],"01");
            else if(strcmp(str2,"C")==0)
                strcat(instr[line],"10");
            else if(strcmp(str2,"NC")==0)
                strcat(instr[line],"11");
            strcat(instr[line],"11");
            strcat(instr[line],str3);
        }
        else{
            strcat(instr[line],"011");
            strcpy(str2,convFromHex(str2,2));
            strcat(instr[line],"11");
            strcat(instr[line],str2);
        }
    }
    else if(strcmp(str1,"return")==0){
        strcpy(instr[line],"100");
        fscanf(pf1,"%s",str2);
        if(strcmp(str2,"UNC")==0)
            strcat(instr[line],"0110010000000");
        else{
            strcat(instr[line],"1");
            if(strcmp(str2,"Z")==0)
                strcat(instr[line],"00");
            else if(strcmp(str2,"NZ")==0)
                strcat(instr[line],"01");
            else if(strcmp(str2,"C")==0)
                strcat(instr[line],"10");
            else if(strcmp(str2,"NC")==0)
                strcat(instr[line],"11");
            strcat(instr[line],"0010000000");
        }
    }
    else if(strcmp(str1,"returni")==0){
```

```
    fscanf(pf1,"%s",str2);
    if(strcmp(str2,"enable")==0)
        strcpy(instr[line],"1000000011110000");
    else
        strcpy(instr[line],"1000000011010000");
}
else if(strcmp(str1,"enable")==0){
    fscanf(pf1,"%s",str2);
    if(strcmp(str2,"interrupt")==0)
        strcpy(instr[line],"1000000000110000");
}
else if(strcmp(str1,"disable")==0){
    fscanf(pf1,"%s",str2);
    if(strcmp(str2,"interrupt")==0)
        strcpy(instr[line],"1000000000010000");
}
else if(strcmp(str1,"load")==0){
    fscanf(pf1,"%s",str2);
    fscanf(pf1,"%s",str3);
    if(str3[0]=='s'){
        strcpy(instr[line],"1100");
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(str3,convFromHex(str3+1,1));
        strcat(instr[line],str2);
        strcat(instr[line],str3);
        strcat(instr[line],"0000");
    }
    else{
        strcpy(instr[line],"0000");
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(str3,convFromHex(str3,2));
        strcat(instr[line],str2);
        strcat(instr[line],str3);
    }
}
else if(strcmp(str1,"and")==0){
    fscanf(pf1,"%s",str2);
    fscanf(pf1,"%s",str3);
    if(str3[0]=='s'){
        strcpy(instr[line],"1100");
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(str3,convFromHex(str3+1,1));
        strcat(instr[line],str2);
        strcat(instr[line],str3);
        strcat(instr[line],"0001");
    }
    else{
        strcpy(instr[line],"0001");
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(str3,convFromHex(str3,2));
        strcat(instr[line],str2);
        strcat(instr[line],str3);
    }
}
else if(strcmp(str1,"or")==0){
    fscanf(pf1,"%s",str2);
    fscanf(pf1,"%s",str3);
    if(str3[0]=='s'){
        strcpy(instr[line],"1100");
```

```
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3+1,1));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
            strcat(instr[line],"0010");
        }
        else{
            strcpy(instr[line],"0010");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3,2));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
        }
    }
    else if(strcmp(str1,"xor")==0){
        fscanf(pf1,"%s",str2);
        fscanf(pf1,"%s",str3);
        if(str3[0]=='s'){
            strcpy(instr[line],"1100");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3+1,1));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
            strcat(instr[line],"0011");
        }
        else{
            strcpy(instr[line],"0011");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3,2));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
        }
    }else if(strcmp(str1,"add")==0){
        fscanf(pf1,"%s",str2);
        fscanf(pf1,"%s",str3);
        if(str3[0]=='s'){
            strcpy(instr[line],"1100");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3+1,1));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
            strcat(instr[line],"0100");
        }
        else{
            strcpy(instr[line],"0100");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3,2));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
        }
    }
    else if(strcmp(str1,"addcy")==0){
        fscanf(pf1,"%s",str2);
        fscanf(pf1,"%s",str3);
        if(str3[0]=='s'){
            strcpy(instr[line],"1100");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3+1,1));
            strcat(instr[line],str2);
```

```
            strcat(instr[line],str3);
            strcat(instr[line],"0101");
        }
        else{
            strcpy(instr[line],"0101");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3,2));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
        }
    }
    else if(strcmp(str1,"sub")==0){
        fscanf(pf1,"%s",str2);
        fscanf(pf1,"%s",str3);
        if(str3[0]=='s'){
            strcpy(instr[line],"1100");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3+1,1));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
            strcat(instr[line],"0110");
        }
        else{
            strcpy(instr[line],"0110");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3,2));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
        }
    }
    else if(strcmp(str1,"subcy")==0){
        fscanf(pf1,"%s",str2);
        fscanf(pf1,"%s",str3);
        if(str3[0]=='s'){
            strcpy(instr[line],"1100");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3+1,1));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
            strcat(instr[line],"0111");
        }
        else{
            strcpy(instr[line],"0111");
            strcpy(str2,convFromHex(str2+1,1));
            strcpy(str3,convFromHex(str3,2));
            strcat(instr[line],str2);
            strcat(instr[line],str3);
        }
    }
    else if(strncmp(str1,"sr",2)==0||strncmp(str1,"rr",2)==0){
        fscanf(pf1,"%s",str2);
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(instr[line],"1101");
        strcat(instr[line],str2);
        strcat(instr[line],"00001");
        switch(str1[2]){
            case '0':
                strcat(instr[line],"110");
                break;
```

```
        case '1':
            strcat(instr[line],"111");
            break;
        case 'x':
            strcat(instr[line],"100");
            break;
        case 'a':
            strcat(instr[line],"000");
            break;
        case '\0':
            strcat(instr[line],"010");
            break;
    }
}
else if(strncmp(str1,"sl",2)==0||strncmp(str1,"rl",2)==0){
    fscanf(pf1,"%s",str2);
    strcpy(str2,convFromHex(str2+1,1));
    strcpy(instr[line],"1101");
    strcat(instr[line],str2);
    strcat(instr[line],"00000");
    switch(str1[2]){
        case '0':
            strcat(instr[line],"110");
            break;
        case '1':
            strcat(instr[line],"111");
            break;
        case 'x':
            strcat(instr[line],"100");
            break;
        case 'a':
            strcat(instr[line],"000");
            break;
        case '\0':
            strcat(instr[line],"010");
            break;
    }
}
else if(strcmp(str1,"input")==0){
    fscanf(pf1,"%s",str2);
    fscanf(pf1,"%s",str3);
    if(str3[0]=='s'){
        strcpy(instr[line],"1011");
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(str3,convFromHex(str3+1,1));
        strcat(instr[line],str2);
        strcat(instr[line],str3);
        strcat(instr[line],"0000");
    }
    else{
        strcpy(instr[line],"1010");
        strcpy(str2,convFromHex(str2+1,1));
        strcpy(str3,convFromHex(str3,2));
        strcat(instr[line],str2);
        strcat(instr[line],str3);
    }
}
else if(strcmp(str1,"output")==0){
    fscanf(pf1,"%s",str2);
```

```
            fscanf(pf1,"%s",str3);
            if(str3[0]=='s'){
                strcpy(instr[line],"1111");
                strcpy(str2,convFromHex(str2+1,1));
                strcpy(str3,convFromHex(str3+1,1));
                strcat(instr[line],str2);
                strcat(instr[line],str3);
                strcat(instr[line],"0000");
            }
            else{
                strcpy(instr[line],"1110");
                strcpy(str2,convFromHex(str2+1,1));
                strcpy(str3,convFromHex(str3,2));
                strcat(instr[line],str2);
                strcat(instr[line],str3);
            }
        }
        else if(strcmp(str1,"ff")==0){
            fscanf(pf1,"%s",str2);
            strcpy(instr[255],"10001101");
            strcpy(str2,convFromHex(str2,2));
            strcat(instr[255],str2);
            line--;
        }
    }
    for(i=0;i<255;i++){
        fprintf(pf2,"\"%s\", \n",instr[i]);
    }
    fprintf(pf2,"\"%s\"",instr[255]);
    return 0;
}

char* convFromHex(char str[], int length)
{
    char bin[8];
    switch(str[0]){
        case '0':
            strcpy(bin,"0000");
            break;
        case '1':
            strcpy(bin,"0001");
            break;
        case '2':
            strcpy(bin,"0010");
            break;
        case '3':
            strcpy(bin,"0011");
            break;
        case '4':
            strcpy(bin,"0100");
            break;
        case '5':
            strcpy(bin,"0101");
            break;
        case '6':
            strcpy(bin,"0110");
            break;
        case '7':
            strcpy(bin,"0111");
```

```
        break;
    case '8':
        strcpy(bin,"1000");
        break;
    case '9':
        strcpy(bin,"1001");
        break;
    case 'A':
        strcpy(bin,"1010");
        break;
    case 'B':
        strcpy(bin,"1011");
        break;
    case 'C':
        strcpy(bin,"1100");
        break;
    case 'D':
        strcpy(bin,"1101");
        break;
    case 'E':
        strcpy(bin,"1110");
        break;
    case 'F':
        strcpy(bin,"1111");
        break;
}
if(length>1)
switch(str[1]){
    case '0':
        strcat(bin,"0000");
        break;
    case '1':
        strcat(bin,"0001");
        break;
    case '2':
        strcat(bin,"0010");
        break;
    case '3':
        strcat(bin,"0011");
        break;
    case '4':
        strcat(bin,"0100");
        break;
    case '5':
        strcat(bin,"0101");
        break;
    case '6':
        strcat(bin,"0110");
        break;
    case '7':
        strcat(bin,"0111");
        break;
    case '8':
        strcat(bin,"1000");
        break;
    case '9':
        strcat(bin,"1001");
        break;
    case 'A':
```

```
            strcat(bin,"1010");
            break;
        case 'B':
            strcat(bin,"1011");
            break;
        case 'C':
            strcat(bin,"1100");
            break;
        case 'D':
            strcat(bin,"1101");
            break;
        case 'E':
            strcat(bin,"1110");
            break;
        case 'F':
            strcat(bin,"1111");
            break;
    }
    bin[8]='\0';
    strcpy(str,bin);
    return str;
}
```

Summary