Practical work nº 4 - Example of hw-sw co-design with PicoBlaze

As a first example of hw-sw co-design, using the PicoBlaze virtual processor, a simple system will be developed. Using the buttons and switches two eight bit operands will be inputted and added.

Specification

The system under development has:

- 4 buttons (one for reset, one to input operand 1, one to input oper. 2, and one to operate)
- 8 switches to describe a 8 bits natural.
- 4 seven segments to show operands and result.

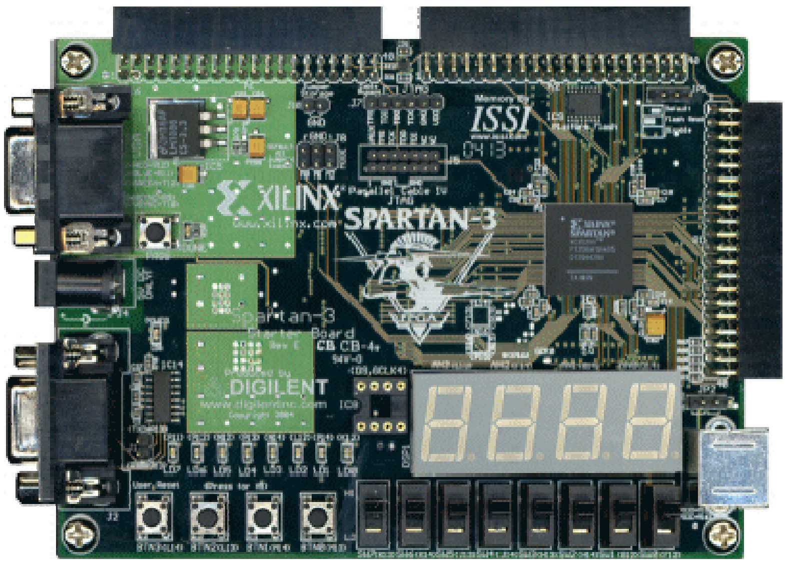


Figure 1 Digilent development board

Hardware platform

The Digilent Spartan-3 development board will be used (http://www.digilentinc.com/). The switches and buttons will be used to input the operands. The design environment contains the *PicoBlaze* virtual processor (a VHDL model).

Hardware – software partitioning

The proposed block diagram is shown in figure 2. The processor is in charge of all operations but the output interface (shows the 4 seven segments).

The *buttons* and *switches* are mapped in different ports. The *in_select* module selects which input read the picpBlaze depending on *port*_id signal.

Five 8 bits registers holds the 4 seven segments graphics and the 8 leds. The 4 registered seven segments are displayed thank to the *7_segement_contrloller* module.

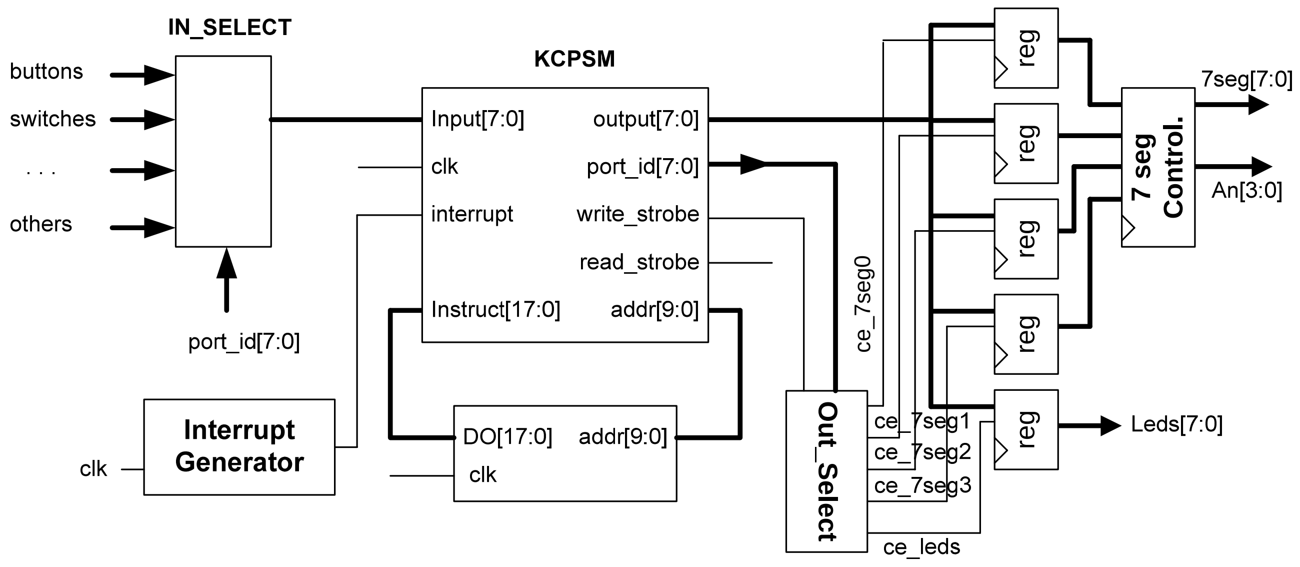


Figure 2 Block diagram

Program generation (software)

1 Assembly language program

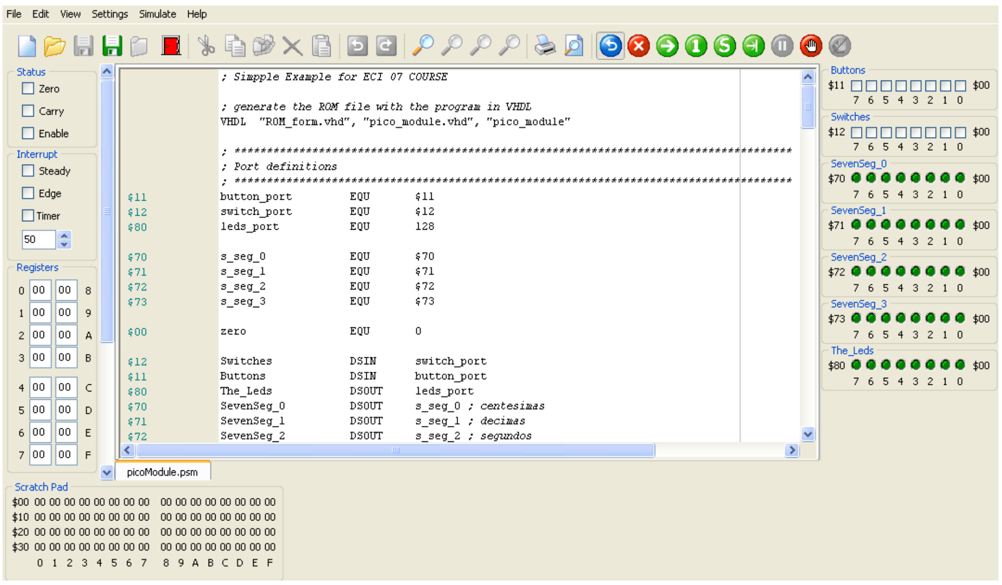
The following program, saved as *picoSimple.asm*, can be assembled and simulated with *pBlaze IDE version: 3.7.4 ß*:Some parts are summarized here

```
; **********************************************************************
; Ports and constant definitions.
; **********************************************************************
button_port         EQU       $11
switch_port         EQU       $
leds_port           EQU       128
s_seg_0             EQU       $70
s_seg_1             EQU       $71
s_seg_2             EQU       $72
s_seg_3             EQU       $73

; **********************************************************************
;  Main program
; **********************************************************************

; Load Scratchpad with seven segments figures
load_scratchpad:    LOAD      sF, $3F            ; cero
                    STORE     sF, 0
                    LOAD      sF, $06            ; uno
                    STORE     sF, 1
                    LOAD      sF, $5B            ; dos
```

```
                     . . .
                    LOAD      sF, $79            ; E
                    STORE     sF, 14
                    LOAD      sF, $71            ; F
                    STORE     sF, 15

                    EINT                       ; eneable interrupt
InitRegisters:
init_Hex0:          LOAD      sA, zero         ; Hexa Digits 1. MSB
init_Hex1:          LOAD      sB, zero         ; Hexa Digits 1. MSB
init_Hex2:          LOAD      sC, zero         ; Hexa Digits 1. MSB
init_Hex3:          LOAD      sD, zero         ; Hexa Digits 1. MSB
init_LEDCount:      LOAD      sE, $FF          ; All LEDs on

                    CALL      send_HOLA        ; Send a Hello message
                    EINT

MainLoop:
read_switches:      IN        s5, switches    ; read Port switches
read_buttons:       IN        s6, buttons     ; read Port buttons
                    COMP      s6, 1           ; if button one is pressed
                    CALL      Z, button_one
                    COMP      s6, 2           ; if button two is pressed
                    CALL      Z, button_two
                    COMP      s6, 4         ; if button three is pressed
                    CALL      Z, button_three
write_leds:         OUT       sE, leds_port   ; out the leds
                    JUMP      MainLoop

show_7segm:
search_scratchpad0: FETCH     s9, sA
display_leds0:      OUT       s9, s_seg_0        ; shows
search_scratchpad1: FETCH     s9, sB
display_leds1:      OUT       s9, s_seg_1        ; shows
search_scratchpad2: FETCH     s9, sC
                    ADD       s9, $80            ; add a dot in 7seg
display_leds2:      OUT       s9, s_seg_2        ;
search_scratchpad3: FETCH     s9, sD
display_leds3:      OUT       s9, s_seg_3        ;
                    RET

button_one:         IN        s6, buttons       ; read buttons again
                    COMP      s6, 0             ; if button one is
                    JUMP      NZ, button_one    ; pressed wait until
                    LOAD      s1, s5            ; button is released
                    LOAD      sA, s1
                    AND       sA, $0F
                    LOAD      sB, s1
                    SR0       sB
                    SR0       sB
                    SR0       sB
                    SR0       sB
                    CALL      show_7segm
                    RET


button_two: ; Similar to button 1
```

```
button_three:       IN        s6, buttons       ; read buttons again
                    COMP      s6, 0             ; if button one is pressed
                    JUMP      NZ, button_three  ; wait until is released
                    LOAD      s3, s1
                    ADD       s3, s2            ; s3 = s1 + s2

Show_result:      LOAD      s9, $40           ; a dash for 7segments
                  OUT       s9, s_seg_0       ; shows a dash
                  LOAD      s8, s3
                  AND       s8, $0F
                  FETCH     s9, s8            ; in scratchpad
                  OUT       s9, s_seg_1       ; shows
                  LOAD      s8, s3
                  SR0       s8
                  SR0       s8
                  SR0       s8
                  SR0       s8
                  FETCH     s9, s8            ; in scratchpad
                  OUT       s9, s_seg_2       ;
                  LOAD      s9, $40
                  OUT       s9, s_seg_3       ; shows a dash
                  RET

; With an interrupt service the 8 leds in the board is shifted
```

Ensure the correct functional behavior of program using PicoBlaxe IDE simulation capabilities.



```
                ; Simpple Example for ECI 07 COURSE

                ; generate the ROM file with the program in VHDL
                VHDL  "ROM_form.vhd", "pico_module.vhd", "pico_module"

                ; ************************************************************************
                ; Port definitions
                ; ************************************************************************
$11             button_port       EQU      $11
$12             switch_port       EQU      $12
$80             leds_port         EQU      128

$70             s_seg_0           EQU      $70
$71             s_seg_1           EQU      $71
$72             s_seg_2           EQU      $72
$73             s_seg_3           EQU      $73

$00             zero              EQU      0

$12             Switches          DSIN     switch_port
$11             Buttons           DSIN     button_port
$80             The_Leds          DSOUT    leds_port
$70             SevenSeg_0        DSOUT    s_seg_0 ; centesimas
$71             SevenSeg_1        DSOUT    s_seg_1 ; decimas
$72             SevenSeg_2        DSOUT    s_seg_2 ; segundos
```

```
Registers
0  00  00  8
1  00  00  9
2  00  00  A
3  00  00  B
4  00  00  C
5  00  00  D
6  00  00  E
7  00  00  F

Scratch Pad
$00  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
$10  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
$20  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
$30  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
      0  1  2  3  4  5  6  7    8  9  A  B  C  D  E  F

Buttons     $11  □ □ □ □ □ □ □ □  $00
                 7 6 5 4 3 2 1 0
Switches    $12  □ □ □ □ □ □ □ □  $00
                 7 6 5 4 3 2 1 0
SevenSeg_0  $70  ● ● ● ● ● ● ● ●  $00
                 7 6 5 4 3 2 1 0
SevenSeg_1  $71  ● ● ● ● ● ● ● ●  $00
                 7 6 5 4 3 2 1 0
SevenSeg_2  $72  ● ● ● ● ● ● ● ●  $00
                 7 6 5 4 3 2 1 0
SevenSeg_3  $73  ● ● ● ● ● ● ● ●  $00
                 7 6 5 4 3 2 1 0
The_Leds    $80  ● ● ● ● ● ● ● ●  $00
                 7 6 5 4 3 2 1 0
```

Figure 3 Picoblaze IDE

2 Generation of the program memory model

The PicoBlaze assembler *kcpsm.exe* generates the VHDL model of the program memory. Some modifications should be done to use the command line version of compiler. An easier way to do that is generate the ROM directly from PicoBlaze IDE.

In order to generate the ROM content the following code line should be added at the beginning:

```
VHDL  "ROM_form.vhd", "name_of_module.vhd", "name_of_entity"
```

Note: The Picoblaze IDE executable, the files *ROM_form.vhd* and *ROM_form.coe* and the *source.asm* should be at the same directory.

Circuit generation (hardware)

For generating the circuit of figure 2, the following models are available:

PicoBlaze (*kcpsm.vhd*),
the program memory (*Pico_simple.vhd*).
The in_select module (*in_select.vhd*)
The out_select module (*out_select.vhd*)
The seven segment controller (sseg_control.vhd)

It remains to add the four seven segment registers. The register will be modeled by a process within the circuit architecture. A summary of VHDL code:

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity picoblaze_simple is
   Port (
     clk_board : in std_logic;
     reset : in std_logic;
     switches : in std_logic_vector(7 downto 0); -- Switches
     buttons: in std_logic_vector(2 downto 0);  -- Buttons
     LEDS : out std_logic_vector(7 downto 0);   -- LEDs
     DIS : out std_logic_vector(3 downto 0);    -- 7 seg selector
     D7S : out std_logic_vector(7 downto 0));   -- 7 segments and Point
   end picoblaze_simple;

architecture Behavioral of picoblaze_simple is
-- declaration of components: KCPSM3, program ROM (pico_Simple).
-- output selector, interrupt generator, segments controller
-- input selector. And declaration of system signals

begin
  processor: kcpsm3
    port map(      address => address,
               instruction => instruction,
                   port_id => port_id,
              write_strobe => write_strobe,
                  out_port => out_port,
               read_strobe => read_strobe,
                   in_port => in_port,
                 interrupt => interrupt,
             interrupt_ack => interrupt_ack,
                     reset => reset,
                       clk => clk);

  program_rom: pico_simple
    port map(      address => address,
               instruction => instruction,
                       clk => clk);
  gen_int: gen_interrupt port map(clk => clk, interrupt => interrupt);

  out_sel: out_select port map( port_id => port_id,
              write_strobe => write_strobe, ce_leds => ce_leds,
                 ce_7seg_0 => ce_7seg_0, ce_7seg_1 => ce_7seg_1,
                 ce_7seg_2 => ce_7seg_2, ce_7seg_3 => ce_7seg_3 );

  in_sel: in_select PORT MAP(
      clk => clk, read_strobe => read_strobe, port_id => port_id,
      buttons => buttons, switches => switches, in_port => in_port );

   registers: process (CLK, reset)
   begin
    if reset = '1' then
        reg_leds <= (others => '0');
        reg_7seg_0 <= (others => '0'); reg_7seg_1 <= (others => '0');
        reg_7seg_2 <= (others => '0'); reg_7seg_3 <= (others => '0');
    elsif CLK='1' and CLK'event then
      if ce_leds = '1' then reg_leds <= out_port; end if;
      if ce_7seg_0 = '1' then reg_7seg_0 <= out_port; end if;
      if ce_7seg_1 = '1' then reg_7seg_1 <= out_port; end if;
      if ce_7seg_2 = '1' then reg_7seg_2 <= out_port; end if;
      if ce_7seg_3 = '1' then reg_7seg_3 <= out_port; end if;
    end if;
   end process;

   sseg_cont: sseg_control port map( clk => clk,
         reg_7seg_0 => reg_7seg_0, reg_7seg_1 => reg_7seg_1,
         reg_7seg_2 => reg_7seg_2, reg_7seg_3 => reg_7seg_3,
         ssel => DIS, sseg => D7S);

   LEDS <= reg_LEDS;

end Behavioral;
```

Complete system simulation

The VHDL model is simulated with ModelSim. For that, the input stimuli can be previously defined and stored within a *do file*; or generate a test bench wave form.

Assuming that the previous file as been saved as *do_file.txt*, its execution is controlled from the VSIM command line (VSIM···>) with do `do_file.txt`. The simulation allows to completely debug both the hardware and the software.

Synthesis and implementation

The programs XST (Xilinx Synthesis Technology) and ISE (Integrated System Environment) are available within the *Xilinx – Project Navigator* package.

The following ping assignment should be ensured:

```
  NET "clk_board"  LOC = "T9";
  NET "reset"  LOC = "l14"; # the buttons<3>
  NET "buttons<2>"  LOC = "l13"  ;
  NET "buttons<1>"  LOC = "m14"  ;
  NET "buttons<0>"  LOC = "m13"  ;
  NET "D7S<7>"  LOC = "p16"  ;
  NET "D7S<6>"  LOC = "n16"  ;
  NET "D7S<5>"  LOC = "f13"  ;
  NET "D7S<4>"  LOC = "r16"  ;
  NET "D7S<3>"  LOC = "p15"  ;
  NET "D7S<2>"  LOC = "n15"  ;
  NET "D7S<1>"  LOC = "g13"  ;
  NET "D7S<0>"  LOC = "e14"  ;

  NET "DIS<3>"  LOC = "e13"  ;
  NET "DIS<2>"  LOC = "f14"  ;
  NET "DIS<1>"  LOC = "g14"  ;
  NET "DIS<0>"  LOC = "d14"  ;

  NET "LEDS<0>"  LOC = "k12"  ;
  NET "LEDS<1>"  LOC = "p14"  ;
  NET "LEDS<2>"  LOC = "l12"  ;
  NET "LEDS<3>"  LOC = "n14"  ;
  NET "LEDS<4>"  LOC = "p13"  ;
  NET "LEDS<5>"  LOC = "n12"  ;
  NET "LEDS<6>"  LOC = "p12"  ;
  NET "LEDS<7>"  LOC = "p11"  ;

  NET "switches<7>"  LOC = "k13"  ;
  NET "switches<6>"  LOC = "k14"  ;
  NET "switches<5>"  LOC = "j13"  ;
  NET "switches<4>"  LOC = "j14"  ;
  NET "switches<3>"  LOC = "h13"  ;
  NET "switches<2>"  LOC = "h14"  ;
  NET "switches<1>"  LOC = "g12"  ;
  NET "switches<0>"  LOC = "f12"  ;
```

Test

The *pico_simple.bit* file, generated at the end of the implementation, is downloaded to the Digilent Spartan-3 development board using the impact program include in Xilinx ISE.