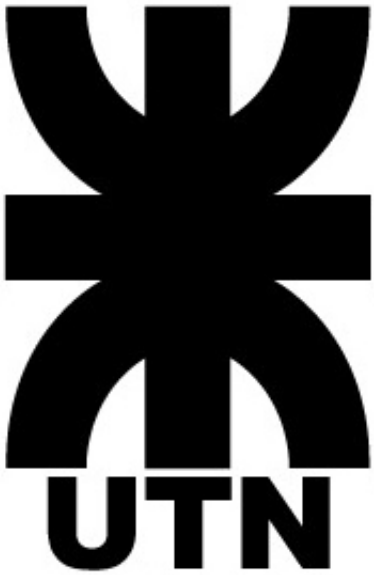


Universidad Tecnológica Nacional
Facultad Regional de Córdoba

Trabajo Práctico De Laboratorio $N°3$

Navarro, Facundo 63809

Curso: 6r4
Grupo $N°5$

Técnicas Digitales IV

Docentes:
Ing. Cayuela, Pablo
Ing. Olmedo, Sergio

23 de junio de 2020

Índice

1. Introducción

En este practico tiene como objetivo embeber un procesador dentro de la FPGA, en este caso se utilizo el softcore “PicoBlaze”. PicoBlaze es la designación de una serie de tres gratuitas procesador suaves (softcore) núcleos de Xilinx para su uso en sus FPGA y CPLD productos. Se basan en una arquitectura RISC de 8-bit RISC y pueden alcanzar velocidades de hasta 100 MIPS en el Virtex 4 FPGA de la familia 7s.

Los procesadores tienen una dirección de 8 bits y puerto de datos para acceso a una amplia gama de periféricos. La licencia de los núcleos permite su uso gratuito, aunque sólo en los dispositivos de Xilinx, y vienen con herramientas de desarrollo . Herramientas de terceros están disponibles en Mediatronix y otros. El PauloBlaze es una implementación VHDL código abierto bajo la licencia Apache .

El diseño original fue nombrado PicoBlaze KCPSM que significa çonstante (K) con código programable Máquina de Estado”(antes ”de Ken Chapman PSM”). Ken Chapman fue el diseñador de sistemas de Xilinx que ideó e implementó el microcontrolador.

2. Desarrollo

Realizar un proyecto donde integre el ejemplo de interrupciones de la documentación del KCPSM6 en VHDL. El archivo principal KCPSM.vhd contiene la descripción de uso libre de un CPU llamado Picoblaze, diseñado por el Ing. Ken Chapman de la empresa Xilinx.

2.1. Estructura del proyecto

El proyecto fue desarrollado en el SW Vivado 2019.2.1 ya que se utilizó la plataforma de entrenamiento de Digilent, Basys3.

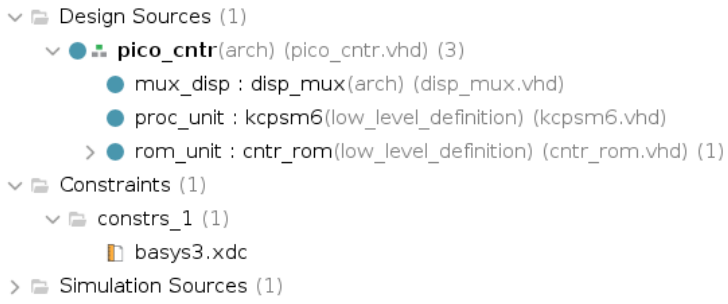
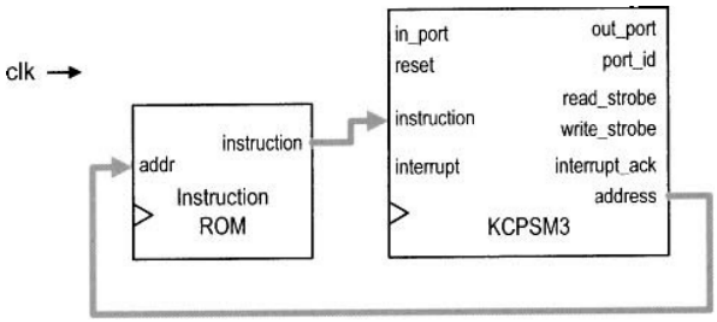


Figura 1: Árbol del proyecto.

2.2. Desarrollo

Para poder utilizar el softcore PicoBlaze es tan simple como instanciar el archivo HDL provisto del sitio web de Xiling, consigo trae una serie de herramientas para generar el archivo ROM.vhd necesario que contendrá el código ensamblador en .psm correspondiente para el microcontrolador. En la figura **2** se puede observar los dos modulos HDL de top-level, el módulo KCPSM6 y la ROM de instrucciones.



```
-- ======================================================
--  KCPSM and ROM instantiation
-- ======================================================
 proc_unit: entity work.kcpsm6(low_level_definition)
 generic map (
                hwbuild => X"00",
       interrupt_vector => X"3FF",
scratch_pad_memory_size => 256
             )
 port map(
                address => address,
            instruction => instruction,
            bram_enable => bram_enable,
                port_id => port_id,
           write_strobe => write_strobe,
         k_write_strobe => k_write_strobe,
               out_port => out_port,
            read_strobe => read_strobe,
                in_port => in_port,
              interrupt => interrupt,
          interrupt_ack => interrupt_ack,
                  sleep => kcpsm6_sleep,
                  reset => kcpsm6_reset,
                    clk => clk
);

 rom_unit: entity work.cntr_rom(low_level_definition)
 generic map(
                C_FAMILY => "7S",
     C_RAM_SIZE_KWORDS => 2,
  C_JTAG_LOADER_ENABLE => 0
  )
 port map(
                address => address,
            instruction => instruction,
                 enable => bram_enable,
                    rdl => rdl,
                    clk => clk);

 -- Unused inputs on processor
 kcpsm6_reset <= '0';
 kcpsm6_sleep <= '0';
 interrupt    <= reset;
```

Figura 2: Instansiacion de ROM y Core.

2.3. Codigo en PSM

Debido a su simplicidad, PicoBlaze no puede soportar lenguajes de programacion de alto nivel, y el desarrollo de codigo es en general desarrollado en lenguaje ensamblador. El metodo desarrollado fue el de "divide y conquistaras", de modo que se plantea un programa principal el cual llama a distintas funciones que ejecutan una tarea en particular, tal como se observa en **3**

```
;======================================================
; Main program
;======================================================
;Calling hierarchy:
;
;main
;    - init
;    - read_switch
;    - stop?
;    - sseg_up
;    - counter_up
;    - delay
;    - output_led
;        -hex_to_led
;    - delay

    call init
forever:
    call output_led
    call read_switch
    compare stop_sw, 01
    jump z, continue
    call sseg_up
    call counter_up
    call delay_500ms
continue:
    jump forever
```

Figura 3: Estructura divide and conquer.

Para la generacion del .vhd necesario para instanciar en el Vivado, Xilinx provee una serie de herramientas que se pueden ver en **4**, el compilador kcpsm6 fue ejecutado en Linux a traves de "wine"

Figura 4: Archivos necesarios para compilar una ROM para PicoBlaze.

2.4. Interrupciones

PicoBlaze es capaz de manejar interrupciones para circuitos externos de I/O, en nuestro caso como se vio en la instansiacion del CPU, la memoria de interrupcion esta ubicada en 3FF, tanto la llamada como la inicializacion se ve en la figura **6**

```
;=====================================================
;routine: clr_data_mem
;  function: clear data ram
;  temp register: data
;=====================================================
init:
    enable interrupt
clr_data_mem:
    load stop_sw, 00
    load data_lsb, 00
    load data_msb, 00
    load value_lsb, 00
    load value_msb, 00
    load led0, 00
    load led1, 00
    load led2, 00
    load led3, 00
    load blink, 00
    return

;=====================================================
;routine: interrupt service routine
;=====================================================
int_service_routine:
    load value_msb, 00
    load value_lsb, 00
    load led0, 00
    load led1, 00
    load led2, 00
    load led3, 00
    call output_led
    returni enable

;=====================================================
;interrupt vector
;=====================================================
   address 3FF
   jump int_service_routine
```

Figura 5: Habilitacion y llamada de interrupciones.

2.5. Salida leds siete segmentos

La salida del contador se realiza tanto en los leds provistos por la placa como por los 7 segmentos, la decodificacion del numero en binario a bcd es realizado por el codigo ensamblador. En la figura ??

```
;=========================================================
;routine: hex_to_led
;  function: convert a hex digit to 7-seg led pattern
;  input register: data
;  output register: data
;=========================================================

hex_to_led:
    compare data, 00
    jump nz, comp_hex_1
    load data, C0          ;7seg pattern 0
    jump hex_done
comp_hex_1:
    compare data, 01
    jump nz, comp_hex_2
    load data, F9          ;7seg pattern 1
    jump hex_done
comp_hex_2:
    compare data, 02
    jump nz, comp_hex_3
    load data, A4          ;7seg pattern 2
    jump hex_done
comp_hex_3:
    compare data, 03
    jump nz, comp_hex_4
    load data, B0          ;7seg pattern 3
    jump hex_done
comp_hex_4:
    compare data, 04
    jump nz, comp_hex_5
    load data, 99          ;7seg pattern 4
    jump hex_done
comp_hex_5:
    compare data, 05
    jump nz, comp_hex_6
    load data, 92          ;7seg pattern 5
    jump hex_done
comp_hex_6:
    compare data, 06
    jump nz, comp_hex_7
    load data, 82          ;7seg pattern 6
    jump hex_done
comp_hex_7:
    compare data, 07
    jump nz, comp_hex_8
    load data, F8          ;7seg pattern 7
    jump hex_done
comp_hex_8:
    compare data, 08
    jump nz, comp_hex_9
    load data, 80          ;7seg pattern 8
    jump hex_done
comp_hex_9:
    compare data, 09
    jump nz, comp_hex_a
    load data, 98          ;7seg pattern 9
    jump hex_done
```

Figura 6: Decodificación a nivel de software.

3. Multimedia

Se adjunta un dos links de unos vídeos demostrativos:

- Presentación: `https://youtu.be/RMeyCFNI-uY`
- Demo: `https://youtu.be/nITNLCbS1TU`