



EXAMEN DE SISTEMAS EMPOTRADOS

CURSO 2019-20, FINAL, 3 DE SEPTIEMBRE DE 2020

1. (1 punto) Un sistema empotrado dispone de un sistema de memoria central constituido por una memoria principal Mp y un cache Mc. Mp tiene una dimensión de 2M palabras y está estructurada como un conjunto de módulos de 32K palabras con entrelazado de orden inferior. Mc tiene un tamaño de 64 K palabras organizada en conjuntos, con un grado de asociatividad de 8. Se pide:

- Tamaño de línea de Mc para minimizar el tiempo de transferencia entre Mp y Mc.
- Interpretación de los bits de la dirección física del sistema de memoria para Mp
- Interpretación de los bits de la dirección física del sistema de memoria para Mc.
- Si en un determinado instante el conjunto 5 contiene las etiquetas (135, 149, 23, 4) y el conjunto 6 las etiquetas (123, 233, 24, 135) ¿Qué direcciones de Mp están cargados en cada una de dichas líneas de Mc?

2. (0.5 punto) Utilización de criptografía en los sistemas empotrados. Problemas y soluciones

3. (0.50 punto) Bus USB

4. (0.25 puntos) Protocolo ZigBee

5. (0.75 punto) Explica con palabras en qué consiste el diseño del controlador del motor paso a paso de la placa de expansión que utilizamos en el laboratorio.

6. (2 puntos) El siguiente código, correspondiente a un PWM, permite generar una onda (*salida*) de periodo determinado por la señal *numciclos* y que está a 1 un número de ciclos dado por la señal *referencia* y el resto de ciclos del periodo está a 0.

```
-----  
-- PWM --  
-----  
PWM1: process (Bus2IP_Reset, Bus2IP_Clk)  
begin  
    if (Bus2IP_Reset = '1') then  
        cnt <= (others => '0');  
    elsif rising_edge(Bus2IP_Clk) then  
        if (cnt < numciclos) then  
            cnt <= (others => '0');  
        else  
            cnt <= cnt + 1;  
        end if;  
    end if;  
    salida <= '1' when (cnt < referencia) else '0';  
end process;
```

Nombre del Alumno:
Apellidos:

DNI:

`end process;`

En la página siguiente se dispone del código `user_logic.vhd`, generado por la herramienta EDK utilizada en los laboratorios, para añadir un periférico mapeado en memoria con un único registro de L/E (`slv_reg0`). Se pide realizar varias modificaciones sobre el propio código. Puede utilizarse papel anexo pero debe quedar bien claro en qué parte del código inicial se realizan las modificaciones.

- a. Explicar con palabras cómo se utiliza este PWM para controlar la intensidad de un led (0.5 puntos).
- b. Modificar el código para que se utilicen 2 registros de comunicación con microblaze, de solo escritura, que se correspondan con las señales *numciclos* y *referencia* y un puerto de salida que sea la onda *salida*, que supuestamente estará conectada al led (0.5 puntos).
- c. Indicar las modificaciones necesarias para controlar el color de un led RGB (0.5 puntos).
- d. Añadir un puerto de entrada *On_off*, que supuestamente estará conectado a un switch, que permita apagar/encender el led. Modificar el código vhd donde corresponda para que el circuito se comporte según lo especificado (0.5 puntos).

```

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

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

entity user_logic is
  generic
  (
    C_SLV_DWIDTH      : integer      := 32;
    C_NUM_REG          : integer      := 1
  )
  port
  (

    Bus2IP_Clk          : in  std_logic;
    Bus2IP_Reset         : in  std_logic;
    Bus2IP_Data          : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE            : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE          : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE          : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data          : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck         : out std_logic;
    IP2Bus_WrAck         : out std_logic;
    IP2Bus_Error         : out std_logic
  );
end entity user_logic;

architecture IMP of user_logic is

  signal slv_reg0        : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel : std_logic_vector(0 to 0);
  signal slv_reg_read_sel  : std_logic_vector(0 to 0);
  signal slv_ip2bus_data   : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack      : std_logic;
  signal slv_write_ack     : std_logic;

begin

  slv_reg_write_sel <= Bus2IP_WrCE(0 to 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(0 to 0);
  slv_write_ack     <= Bus2IP_WrCE(0);
  slv_read_ack      <= Bus2IP_RdCE(0);

```

Nombre del Alumno:
Apellidos:

DNI:

```
SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
        if Bus2IP_Reset = '1' then
            slv_reg0 <= (others => '0');
        else
            case slv_reg_write_sel is
                when "1" =>
                    slv_reg0 <= Bus2IP_Data;
                end if;
            end loop;

            when others => null;
        end case;
    end if;
end if;
end process SLAVE_REG_WRITE_PROC;

SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0 ) is
begin

    case slv_reg_read_sel is
        when "1" => slv_ip2bus_data <= slv_reg0;

        when others => slv_ip2bus_data <= (others => '0');
    end case;

end process SLAVE_REG_READ_PROC;

IP2Bus_Data <= slv_ip2bus_data when slv_read_ack = '1' else
    (others => '0');

IP2Bus_WrAck <= slv_write_ack;
IP2Bus_RdAck <= slv_read_ack;
IP2Bus_Error <= '0';

end IMP;
```