Facultad de Informática
Universidad Complutense de Madrid

[1]

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:

a. Tamaño de línea de Mc para minimizar el tiempo de transferencia entre Mp y Mc.
b. Interpretación de los bits de la dirección física del sistema de memoria para Mp
c. Interpretación de los bits de la dirección física del sistema de memoria para Mc.
d. 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';
```

[1]

```
                    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 vhdl donde corresponda para que el circuito se comporte según lo especificado (0.5 puntos).
.

a) El Pulse Width Modulation es una técnica de modulación de ondas cuadradas que determina la anchura de los pulsos transmitidos, se puede utilizar para controlar la intensidad de encendido de un LED.

La señal numciclos define el **número total de ciclos** que componen un período de la onda cuadrada.
La señal referencia determina el **número de ciclos en los que la señal está en estado '1'** dentro de un período.
La salida salida será:
'1' durante los primeros referencia ciclos (parte activa del duty cycle).
'0' durante el resto de los numciclos (parte inactiva).

La intensidad del LED depende del **duty cycle**:
Cuando referencia es grande (más ciclos en '1'), la intensidad del LED es **alta**.
Cuando referencia es pequeña (más ciclos en '0'), la intensidad del LED es **baja**.
Si referencia = 0, el LED está **apagado**.

```
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);
```

```
 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;
```

1. Mp 2M palabras → 2^{21}

↳ cjto. de módulos de 32K palabras ↳ 2^{15}

Mc 64K palabras → $2^6 \cdot 2^{10} = 2^{16}$

↳ grado 8 asociatividad

b) Interpretación bits Mp:

nº modulos = $\frac{2^{21}}{2^{15}} = 64$ modulos

$\log_2 64 = 6$ bits para modulo

$\log_2 2^{15} = 15$ bits para palabra

a) Tam. línea Mc:

Para minimizar el tiempo de transferencia entre Mp y Mc el tamaño de la línea de Mc debería ser el nº modulos de la Mp.

Tam línea Mc = nº modulos Mp = 64 palabras/línea.

c) Interpretación bits Mc:

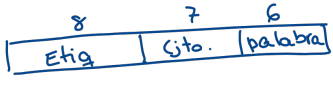
nº líneas = $\frac{2^{16}}{64} = 1024$ líneas en total

nº conjuntos = $\frac{1024}{8} = 128$ conjuntos

$\log_2 64 = 6$ bits para la palabra

$\log_2 128 = 7$ bits para indice cjto.

21 - 6 - 7 = 8 bits para etiqueta.



d) Conjunto 5:

135 → 10000111 | 000010 = 17285 (135 | 5)

149 · 128 + 5 = 19077

23 · 128 + 5 = 2949

4 · 128 + 5 = 517

↳ conj. donde está

↳ nº conjuntos

Conjunto 6:

123 · 128 + 6 = 15750

233 · 128 + 6 = 29830

24 · 128 + 6 = 3078

135 · 128 + 6 = 17286

```vhdl
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 := 6 -- num de registros, 2 para cada color
 port
 (
 switch : in std_logic;
 red : out std_logic;
 green : out std_logic;
 blue : out std_logic;

 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;
```

```vhdl
architecture IMP of user_logic is
 signal slv_reg0, slv_reg1 : std_logic_vector(0 to C_SLV_DWIDTH-1);
 signal slv_reg2, slv_reg3 : std_logic_vector(0 to C_SLV_DWIDTH-1);
 signal slv_reg4, slv_reg5 : std_logic_vector(0 to C_SLV_DWIDTH-1);

 signal slv_reg_write_sel : std_logic_vector(0 to 1);
 signal slv_reg_read_sel : std_logic_vector(0 to 1);
 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;

 signal numciclos_red, referencia_red : std_logic_vector(0 to C_SLV_DWIDTH-1);
 signal numciclos_green, referencia_green : std_logic_vector(0 to C_SLV_DWIDTH-1);
 signal numciclos_blue, referencia_blue : std_logic_vector(0 to C_SLV_DWIDTH-1);

 signal salida_red, salida_blue, salida_green, : std_logic; -- Salida PWM conectada al LED
 signal cnt_red, cnt_green, cnt_blue : std_logic_vector(0 TO C_SLV_DWIDTH - 1)); -- Contadores para PWM
```

```
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) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5);
 slv_read_ack <= Bus2IP_RdCE(0)  or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5);


 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');
            slv_reg1 <= (others => '0');
            slv_reg2 <= (others => '0');
            slv_reg3 <= (others => '0');
            slv_reg4 <= (others => '0');
            slv_reg5 <= (others => '0');
         else
         case slv_reg_write_sel is
            when "001" =>
               slv_reg0 <= Bus2IP_Data; --
            when "010" =>
               slv_reg1 <= Bus2IP_Data;
            when "011" =>
               slv_reg2 <= Bus2IP_Data;
            when "100" =>
               slv_reg3 <= Bus2IP_Data;
            when "101" =>
               slv_reg4 <= Bus2IP_Data;
            when "110" =>
               slv_reg5 <= Bus2IP_Data;
       when others => null;
       end case;
      end if;
      end if;
  end process SLAVE_REG_WRITE_PROC;
 ---------------------------
 -- PWM --
 ---------------------------
 PWM1: process (Bus2IP_Reset, Bus2IP_Clk)
   begin
      if (Bus2IP_Reset = '1') then
           cnt_red <= (others => '0');
           cnt_green <= (others => '0');
           cnt_blue <= (others => '0');
      elsif rising_edge(Bus2IP_Clk) then

         if (cnt_red < numciclos_red) then
            cnt_red <= (others => '0');
         else
         cnt_red <= cnt_red + 1;
         end if;

         if (cnt_green < numciclos_green) then
            cnt_green <= (others => '0');
         else
         cnt_green <= cnt_green + 1;
         end if;

         if (cnt_blue < numciclos_blue) then
            cnt_blue <= (others => '0');
         else
         cnt_blue <= cnt_blue + 1;
         end if;

      end if;

      if switch ='0' then
         salida_red <= '0';
         salida_green <= '0';
         salida_blue <= '0';
      else
         salida_red <= '1' when (cnt_red < referencia_red) else '0';
         salida_green <= '1' when (cnt_green < referencia_green) else '0';
         salida_blue <= '1' when (cnt_blue < referencia_blue) else '0';
      end if;
end process PWM1;
```

```
SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0 ) is
   begin
      case slv_reg_read_sel is
         when "001" => slv_ip2bus_data <= slv_reg0;
         when "010" => slv_ip2bus_data <= slv_reg1;
         when "011" => slv_ip2bus_data <= slv_reg2;
         when "100" => slv_ip2bus_data <= slv_reg3;
         when "101" => slv_ip2bus_data <= slv_reg4;
         when "110" => slv_ip2bus_data <= slv_reg5;
         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';

numciclos_red <= slv_reg0;
referencia_red <= slv_reg1;
numciclos_green <= slv_reg2;
referencia_green <= slv_reg3;
numciclos_blue <= slv_reg4;
referencia_blue <= slv_reg5;

red <= salida_red;
green <= salida_green;
blue <= salida_blue;

end IMP;
```