Facultad de Informática
Universidad Complutense de Madrid

EXAMEN DE SISTEMAS EMPOTRADOS

CURSO 2016-17, FINAL, 7 DE FEBRERO DE 2017

1. (1 punto) Un sistema empotrado dispone de un sistema de memoria central constituido por una memoria principal Mp y una cache Mc. Mp tiene una dimensión de 4M palabras y está estructurada como un conjunto de módulos de 128K palabras con entrelazado de orden inferior. Mc tiene un tamaño de 64 K palabras organizada en 256 conjuntos. Se pide:

a. Calcular el tamaño de línea para minimizar el tiempo de transferencia de bloques entre Mp y Mc. Calcular para dicho tamaño de línea el grado de asociatividad. (**0. 5 puntos**)
b. Interpretación de los bits de la dirección física del sistema de memoria para Mp y Mc (**0.25 puntos**)
c. Si en un determinado instante el conjunto 1 contiene una línea con la etiqueta 133 (en decimal) y el conjunto 2 tiene una línea con la etiqueta 241 (en decimal), ¿qué direcciones de Mp están cargados en cada una de dichas líneas de Mc? (**0.25 puntos**)

2. (0.75 puntos) Bus I2C

3. (0.5 puntos) Explica cómo funciona una fotoresistencia LDR y un sensor de aceleración. ¿Cómo se conectan dicho sensores a un sistema empotrado?

4. (0.75 puntos) Optimización del consumo de un sistema empotrado a nivel de aplicación/sistema

5. (**2 puntos**) 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 accesible en lectura y escritura a través de un registro 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. Modificar el código para instanciar un motor unipolar cuya entity es la siguiente (1 punto).

```
entity motorstep is
Port ( clk : in std_logic ;
    rst : in std_logic ;
    dir : in std_logic ;
    stop : in std_logic ;
    halfstep : in std_logic ;
    motor : out std_logic_vector (3 downto 0);
    step : out std_logic_vector (2 downto 0));
    end motorstep ;
```

La señal *motor* son las líneas de salida de control del motor. Atacan directamente al driver externo del motor. La señal *step* es el paso actual del motor generada por el

driver interno, *dir* es la dirección de giro, stop es la señal de parada y *halfstep* es una señal para determinar si se quiere que funcione con medio paso o no.
Añadir las entradas/salidas necesarias en user_logic.

b. La comunicación entre el motor y microblaze se realiza a través de dos registros, uno de *control* y otro de *estado*. El registro de *control* tiene la siguiente información: dirección de giro bit 31, stop bit 30, paso bit 29 , y número de pasos que se desea que gire el motor, los bits del 25 al 28. El registro de *estado* indica el número de pasos que ha girado desde que se dio la orden de empezar.
Modificar el código actualizando los procesos de lectura y escritura para que existan estos dos registros, y cada vez que se escriba en el registro de *control* desde microblaze, se vuelque los datos correspondientes sobre las entradas del driver interno del motor.
(**0.5 puntos**).

c.- Añadir un contador que divida la frecuencia del reloj entre 4, y dicho reloj sea la entrada de reloj del driver del motor.
(**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);
```

```
 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 4M palabras → 2^{22}

↳ módulos de 128K palabras → 2^{17}

Mc 64K palabras → 2^{16}

↳ 256 conjuntos

a) Para minimizar el tiempo de transferencia de bloques entre Mc y Mp, el tamaño de línea debe ser igual al nº módulos de Mp.

$$nº\ módulos = \frac{2^{22}}{2^{17}} = 32\ módulos$$

Tam. línea = 32 palabras/línea.

$$Grado\ asoc. = \frac{nº\ líneas}{nº\ cjtos.} = \frac{\frac{2^{16}}{32}}{256} = 8$$

b) Interpretación bits dir. física:

Mp → 2^{22}

nº módulos = 32

$\log_2 32$ = 5 bits para módulo

$\log_2 2^{17}$ = 17 bits para palabra

5	17
Módulo	Palabra

Mc → 2^{16}

$\log_2 256$ = 8 bits para índice cjto.

$\log_2 32$ = 5 bits para palabra

22 − 5 − 8 = 9 bits para etiqueta

9	8	5
Etiq	cjto.	Palabra

c) Conjunto 1: 133 → 133 · 256 + 1 = 34049

Conjunto 2: 241 → 241 · 256 + 2 = 61698

Se multiplica por el nº conjuntos y se suma el conjunto en el que estoy.

34049 → 01000101 00000001 (rango: 00000 - 11111 → 1089568 - 1089599)

```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; -- Ancho de datos
 C_NUM_REG : integer := 2      -- Dos registros: control y estado
 port
 (

 motor         : out std_logic_vector(3 downto 0); -- Salidas de control al driver
 step          : out std_logic_vector(2 downto 0); -- Paso actual del motor
 clk_motor     : out std_logic;                    -- Reloj para el driver del motor
 dir           : out std_logic;                    -- Dirección del motor
 stop          : out std_logic;                    -- Señal de parada del motor
 halfstep      : out std_logic                     -- Medio paso o paso completo


 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_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;

 signal slv_reg0 : std_logic_vector(0 to C_SLV_DWIDTH-1); -- Registro de control
 signal slv_reg1 : std_logic_vector(0 to C_SLV_DWIDTH-1); -- Registro de estado

-- Señales de control para el motor
 signal num_steps : std_logic_vector(3 downto 0); -- Número de pasos (bits 25-28)
 signal step_count : std_logic_vector(31 downto 0); -- Contador de pasos realizados
 signal clk_div    : std_logic; -- Reloj dividido

-- Contador para dividir la frecuencia del reloj
 signal clk_div_counter : integer range 0 to 3 := 0;
```

```
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);
 slv_read_ack <= Bus2IP_RdCE(0) or Bus2IP_WrCE(1);


 CLK_DIVIDER: process(Bus2IP_Clk)
   begin
      if rising_edge(Bus2IP_Clk) then
         if clk_div_counter = 3 then
               clk_div <= not clk_div;
               clk_div_counter <= 0;
         else
               clk_div_counter <= clk_div_counter + 1;
         end if;
      end if;
   end process;

   clk_motor <= clk_div;

 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');
         else
         case slv_reg_write_sel is
            when "01" => slv_reg0 <= Bus2IP_Data;
            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 "01" => slv_ip2bus_data <= slv_reg0;
         when "10" => slv_ip2bus_data <= slv_reg1;
         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';

-- Extracción de señales del registro de control
dir      <= slv_reg0(31); -- Dirección de giro (bit 31)
stop     <= slv_reg0(30); -- Parada (bit 30)
halfstep <= slv_reg0(29); -- Medio paso o paso completo (bit 29)
num_steps <= slv_reg0(25 downto 28); -- Número de pasos (bits 25-28)
```

```
-- Contador de pasos realizados
STEP_COUNTER: process(clk_div, Bus2IP_Reset, stop)
   begin
      if Bus2IP_Reset = '1' then
         step_count <= (others => '0');
      elsif rising_edge(clk_div) then
         if stop = '1' then
               step_count <= step_count; -- Detener el conteo
         else
               step_count <= step_count + 1; -- Incrementar pasos realizados
         end if;
      end if;
end process;

slv_reg1 <= step_count;

-- Instancia del motor
motor_inst: entity work.motorstep
    port map(
        clk       => clk_div,     -- Reloj dividido
        rst       => Bus2IP_Reset,
        dir       => dir,
        stop      => stop,
        halfstep  => halfstep,
        motor     => motor,
        step      => step
    );

end IMP;
```