UPMDIE INDUSTRIALES

Electrónica Digital (GITI)
Prueba de Evaluación Continua (PEC)
Curso 2020/21 – Diciembre

POLITÉCNICA

CUESTIÓN 1 (5 puntos)

Se quiere diseñar un contador síncrono de dos bits con inicialización síncrona (**clear**) y asíncrona (**reset**), señal de habilitación (**enable**) y que ofrezca como salidas una señal de tipo *std_logic_vector* de dos bits con la cuenta (**salida**) y una señal de acarreo (**cout**).

Se piden los siguientes códigos VHDL:

a) Entidad del contador *(0.5 puntos)*.
b) Arquitectura del contador, usando una estructura de contador binario y una señal de tipo *unsigned* como elemento de almacenamiento. *(1 punto)*
c) Arquitectura del contador, de tipo estructural. En este caso se asumirá que existe una entidad y una arquitectura de un contador síncrono de un bit ya diseñada, con el que formar el contador de dos bits pedido. *(1 punto)*
d) Arquitectura del contador usando un contador en anillo como circuito de cuenta, pero que tenga la misma salida que los casos b) y c). *(1 punto)*
e) Arquitectura del contador diseñado mediante una codificación similar a una máquina de estado en la que se use una señal enumerada que describe los estados (E0, E1, E2, …) como el valor de cuenta interno, y haciendo que la señal salida siga una cuenta igual a un código *Gray* (es decir, progresiva y cíclica). *(1,5 punto)*

CUESTIÓN 2 (5 puntos)

Se quiere diseñar un generador de señales digitales configurable basado en un registro de 32 bits que denominaremos acumulador de fase. La salida, que será una señal (**wave**) de 8 bits, se podrá elegir entre cuatro tipos distintos de formas de onda: dientes de sierra, triangular, cuadrada, y PWM. Las formas de onda cuadrada y PWM sólo podrán tener dos valores de salida: 0x00 y 0xff.

El circuito contará con las siguientes entradas de control:
- Incremento (**inc**, 8 bits), con la que se controla la frecuencia de la señal generada. Cada ciclo de reloj, el circuito a diseñar sumará este valor al registro acumulador de fase.
- Selector de forma de onda (**sel**, 2 bits). Queda a vuestra elección la forma de onda que se generará con cada combinación de bits en esta señal.
- Ciclo de trabajo (**duty**, 8 bits) de la señal PWM.

Las distintas formas de onda se generarán de acuerdo con los siguientes criterios:
- La forma de onda de **dientes de sierra** (rampa ascendente) se corresponderá con los 8 bits más significativos del acumulador de fase.
- La forma de onda **triangular** se generará haciendo una operación XOR, bit a bit, entre el bit más significativo del acumulador de fase y cada uno de los 8 bits siguientes (*).
- La forma de onda **cuadrada** se generará con el bit más significativo del acumulador de fase.
- La forma de onda **PWM** se generará comparando los 8 bits más significativos del acumulador de fase con la entrada de control **duty**, que fija el ciclo de trabajo de la señal.

(*) La subida corresponde a los valores en los que el bit más significativo vale ‘0’, y por lo tanto ‘0’ XOR bit(i) = bit(i). La bajada será en los valores en los que el bit más significativo vale ‘1’, por lo que ‘1’ XOR bit(i) = NOT bit(i).

Desarrollar el código VHDL del circuito descrito anteriormente, incluyendo entidad y arquitectura.

Duración del examen: 1 hora 30 minutos

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ej1 is
    port (
        clk    : in  std_logic;
        reset  : in  std_logic;
        clear  : in  std_logic;
        enable : in  std_logic;
        salida : out std_logic_vector(1 downto 0);
        cout   : out std_logic
    );
end ej1;

architecture arch1 of ej1 is

    signal cnt : unsigned(1 downto 0);

begin

    process(clk,reset)
    begin
        if reset = '1' then
            cnt <= (others => '0');
        elsif clk'event and clk = '1' then
            if clear = '1' then
                cnt <= (others => '0');
            else
                if enable = '1' then
                    if cnt = "11" then
                        cnt <= (others => '0');
                    else
                        cnt <= cnt + 1;
                    end if;
                end if;
            end if;
        end if;
    end process;

    salida <= std_logic_vector(cnt);
    cout <= '1' when cnt = "11" and enable = '1' else '0';

end arch1;

architecture arch2 of ej1 is

    component cnt is
        port (
            clk    : in  std_logic;
            reset  : in  std_logic;
            clear  : in  std_logic;
            enable : in  std_logic;
            salida : out std_logic;
            cout   : out std_logic
        );
    end component;

    signal carry : std_logic;

begin

    bit0: cnt
    port map (
        clk    => clk,
        reset  => reset,
        clear  => clear,
        enable => enable,
        salida => salida(0),
        cout   => carry
    );

    bit1: cnt
```

```
74      port map (
75          clk    => clk,
76          reset  => reset,
77          clear  => clear,
78          enable => carry,
79          salida => salida(1),
80          cout   => cout
81      );
82
83  end arch2;
84
85  architecture arch3 of ej1 is
86
87      signal ring : unsigned(3 downto 0);
88
89  begin
90
91      process(clk,reset)
92      begin
93          if reset = '1' then
94              ring <= (0 => '1', others => '0');
95          elsif clk'event and clk = '1' then
96              if clear = '1' then
97                  ring <= (0 => '1', others => '0');
98              else
99                  if enable = '1' then
100                     ring <= ring(2 downto 0) & ring(3);
101                 end if;
102             end if;
103         end if;
104     end process;
105
106     with ring select
107         salida <= "00" when "0001",
108                   "01" when "0010",
109                   "10" when "0100",
110                   "11" when "1000",
111                   "--" when others;
112
113     cout <= '1' when ring = "1000" and enable = '1' else '0';
114
115 end arch3;
116
117 architecture arch4 of ej1 is
118
119     type state_t is (E0, E1, E2, E3);
120     signal state : state_t;
121
122 begin
123
124     process(clk,reset)
125     begin
126         if reset = '1' then
127             state <= E0;
128         elsif clk'event and clk = '1' then
129             if clear = '1' then
130                 state <= E0;
131             else
132                 if enable = '1' then
133                     case state is
134                         when E0 => state <= E1;
135                         when E1 => state <= E2;
136                         when E2 => state <= E3;
137                         when E3 => state <= E0;
138                     end case;
139                 end if;
140             end if;
141         end if;
142     end process;
143
144     with state select
145         salida <= "00" when E0,
146                   "01" when E1,
```

```
147                     "11" when E2,
148                     "10" when E3,
149                     "--" when others;
150 
151     cout <= '1' when state = E3 and enable = '1' else '0';
152 
153 end arch4;
154 
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ej2 is
    port (
        clk   : in  std_logic;
        reset : in  std_logic;
        sel   : in  std_logic_vector(1 downto 0);
        inc   : in  std_logic_vector(7 downto 0);
        duty  : in  std_logic_vector(7 downto 0);
        wave  : out std_logic_vector(7 downto 0)
    );
end ej2;

architecture behavioral of ej2 is

    -- Phase accumulator
    signal acc  : unsigned(31 downto 0);
    -- Waveform: sawtooth
    signal saw  : std_logic_vector(7 downto 0);
    -- Waveform: triangle
    signal msbN : std_logic_vector(7 downto 0);
    signal tri  : std_logic_vector(7 downto 0);
    -- Waveform: square
    signal sqr  : std_logic_vector(7 downto 0);
    -- Waveform: PWM
    signal pwm  : std_logic_vector(7 downto 0);

begin

    -- Phase accumulator
    process(clk, reset)
    begin
        if reset = '1' then
            acc <= (others => '0');
        elsif clk'event and clk = '1' then
            acc <= acc + unsigned(inc);
        end if;
    end process;

    -- Waveform: sawtooth
    saw <= std_logic_vector(acc(31 downto 24));

    -- Waveform: triangle
    msbN <= (others => acc(31));
    tri <= std_logic_vector(acc(30 downto 23)) xor msbN;

    -- Waveform: square
    sqr <= (others => not acc(31));

    -- Waveform: PWM
    pwm <= (others => '1') when acc(31 downto 24) < unsigned(duty) else (others =>
    '0');

    -- Output MUX
    with sel select
        wave <= saw when "00",
                tri when "01",
                sqr when "10",
                pwm when "11",
                (others => '-') when others;

end behavioral;
```