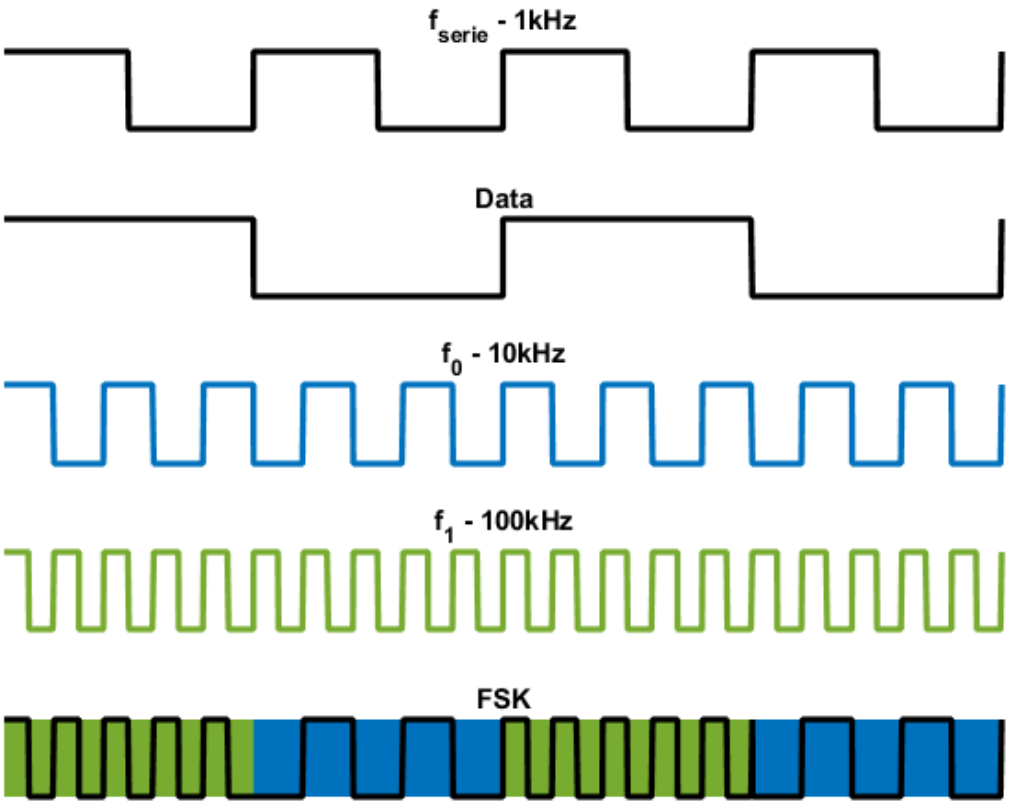
Asignatura: Electrónica Digital (GITI)
Parte 2 – VHDL
Publicación de preactas: 24/07/2020

Fecha: 10/07/2020
Convocatoria: Julio
Revisión: 27/07/2020, 9h

CUESTIÓN ÚNICA (10 puntos)

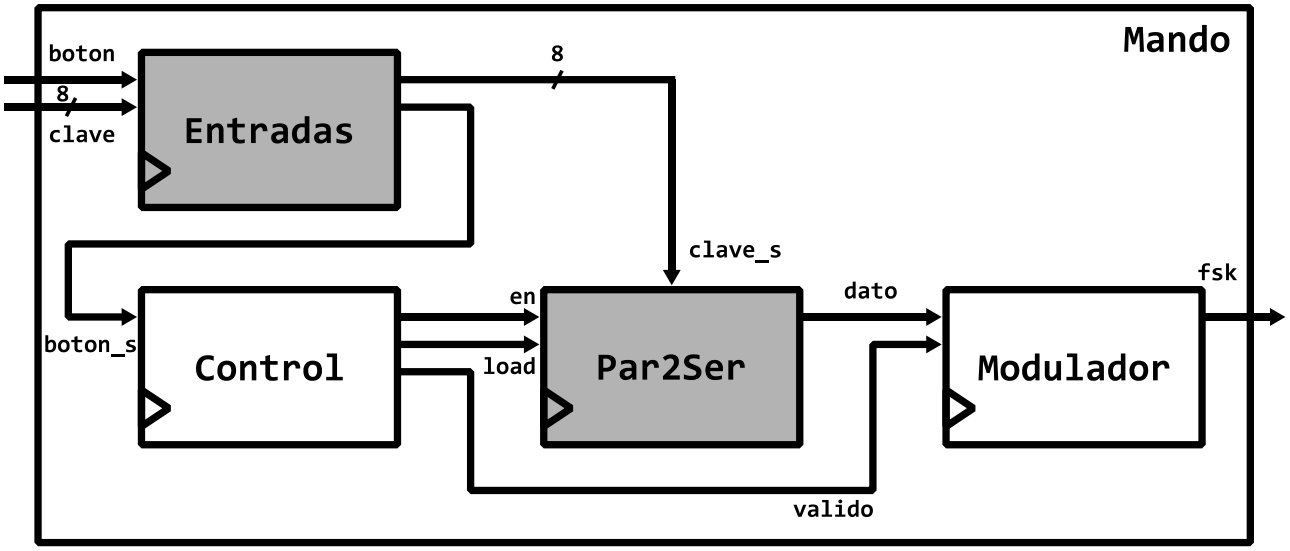
Se pretende diseñar el circuito de control de un mando a distancia universal para puertas de garaje. La funcionalidad del circuito es la que sigue:

- Cuando se pulsa el botón del mando, se deberá almacenar el valor actual de la clave, que a su vez estará indicado por el valor de un *switch* múltiple de 8 bits.
- Una vez almacenado, se deberá transmitir el valor de la clave a través de una línea serie y con una frecuencia base de 1 kHz.
- Dicha línea serie será modulada usando FSK (*Frequency-Shift Keying*), un tipo de modulación de señales que asigna una frecuencia f_0 (10 kHz) al valor '0' y otra frecuencia f_1 (100 kHz) al valor '1' (ver gráficas más abajo).
- Todas las señales de entrada deberán estar debidamente sincronizadas, y se añadirá un antirrebotes (> 2 ms) al botón, con el fin de generar una señal de 1 ciclo de reloj de duración.



En la figura anterior se puede apreciar cómo funciona la modulación FSK: la señal de datos (segunda gráfica), transmitida con una frecuencia base de 1 kHz (primera gráfica), es empleada para seleccionar una de las dos frecuencias portadoras (10 kHz, tercera gráfica; 100 kHz, cuarta gráfica). De esta manera se obtiene como salida la señal modulada (quinta gráfica).

Tras un primer análisis, se ha diseñado el siguiente diagrama de bloques para el circuito:



- El **bloque de entrada** se encarga de sincronizar las señales externas y de filtrar rebotes.
- El **bloque de conversión paralelo a serie** almacena la clave durante la transmisión en un registro de desplazamiento con una señal de habilitación y otra señal de carga en paralelo.
- El **bloque modulador** aplica la modulación FSK a una entrada de datos, pero sólo cuando una señal de control indica que el dato es válido (si no lo es, la salida tiene que valer '0').
- El **bloque de control**, que incluye una máquina de estados y cierta lógica adicional, se usa para gobernar el funcionamiento del circuito.

Se pide:

a) Código VHDL (sólo arquitectura) del **bloque modulador (4 puntos)**.
b) Código VHDL (sólo arquitectura) del **bloque de control (4 puntos)**.
c) Código VHDL (sólo arquitectura) del **bloque *top*** (descripción estructural) **(2 puntos)**.

Notas:

- Aunque no se ha representado, **todos los bloques** del diagrama son secuenciales y síncronos, por lo que **comparten las mismas señales de reloj** (`clk`) **y *reset*** (`reset`). La frecuencia de reloj del sistema es de 100 MHz.
- Se asume que los bloques resaltados en gris en el diagrama han sido diseñados previamente y se dispone de su código VHDL.
- Para que la máquina de estados del bloque de control no sea excesivamente compleja de diseñar y describir, se recomienda utilizar un contador de 0 a 7 con señal de *overflow* como parte de la lógica adicional.

Duración del examen: 1 hora

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

entity modulador is
    port (
        clk    : in  std_logic;
        reset  : in  std_logic;
        dato   : in  std_logic;
        valido : in  std_logic;
        fsk    : out std_logic
    );
end modulador;

architecture behavioral of modulador is

    -- Divisores de frecuencia
    constant C_MAX_100kHz : integer := (100*10**6)/(100*10**3);
    constant C_MAX_10kHz  : integer := 10;
    signal cnt_100kHz  : integer range 0 to C_MAX_100kHz-1;
    signal ovf_100kHz  : std_logic;
    signal cnt_10kHz   : integer range 0 to C_MAX_10kHz-1;
    signal ovf_10kHz   : std_logic;

    -- Generación de portadoras
    signal wave_100kHz : std_logic;
    signal wave_10kHz  : std_logic;

begin

    ---------------------------------------------
    -- Divisores de frecuencia (10kHz, 100kHz) --
    ---------------------------------------------

    process(clk, reset)
    begin
        if reset = '1' then
            cnt_100kHz <= 0;
        elsif clk'event and clk = '1' then
            if cnt_100kHz = C_MAX_100kHz-1 then
                cnt_100kHz <= 0;
            else
                cnt_100kHz <= cnt_100kHz + 1;
            end if;
        end if;
    end process;

    ovf_100kHz <= '1' when cnt_100kHz = C_MAX_100kHz-1 else '0';

    process(clk, reset)
    begin
        if reset = '1' then
            cnt_10kHz <= 0;
        elsif clk'event and clk = '1' then
            if ovf_100kHz = '1' then
                if cnt_10kHz = C_MAX_10kHz-1 then
                    cnt_10kHz <= 0;
                else
                    cnt_10kHz <= cnt_10kHz + 1;
                end if;
            end if;
        end if;
    end process;

    ovf_10kHz <= '1' when ovf_100kHz = '1' and cnt_10kHz = C_MAX_10kHz-1 else '0';

    ------------------------------
    -- Generación de portadoras --
    ------------------------------

    process(clk, reset)
    begin
        if reset = '1' then
```

```
74              wave_100kHz <= '0';
75              wave_10kHz <= '0';
76          elsif clk'event and clk = '1' then
77              if ovf_100kHz = '1' then
78                  wave_100kHz <= not wave_100kHz;
79              end if;
80              if ovf_10kHz = '1' then
81                  wave_10kHz <= not wave_10kHz;
82              end if;
83          end if;
84      end process;
85
86      --------------------------
87      -- Generación de salida --
88      --------------------------
89
90      process(clk, reset)
91      begin
92          if reset = '1' then
93              fsk <= '0';
94          elsif clk'event and clk = '1' then
95              if valido = '1' then
96                  if dato = '1' then
97                      fsk <= wave_100kHz;
98                  else
99                      fsk <= wave_10kHz;
100                 end if;
101             else
102                 fsk <= '0';
103             end if;
104         end if;
105     end process;
106
107     -- NOTA: esto podría hacerse con lógica combinacional;
108     --       al ser salida final del circuito la registramos.
109     --
110     -- fsk <= wave_100kHz when valido = '1' and dato = '1' else
111     --        wave_10kHz when valido = '1' and dato = '0' else
112     --        '0';
113
114 end behavioral;
115
```

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

entity control is
    port (
        clk    : in  std_logic;
        reset  : in  std_logic;
        boton  : in  std_logic;
        load   : out std_logic;
        en     : out std_logic;
        valido : out std_logic
    );
end control;

architecture behavioral of control is

    -- Divisor de frecuencia (1kHz)
    constant C_MAX_1kHz : integer := (100*10**6)/(10**3);
    signal cnt_1kHz : integer range 0 to C_MAX_1kHz-1;
    signal en_1kHz  : std_logic;
    signal ovf_1kHz : std_logic;

    -- Contador de dígitos
    signal digito     : unsigned(2 downto 0); -- Hay 8 dígitos, cuento de 0 a 7
    signal ovf_digito : std_logic;

    -- FSM de control
    type state_t is (S_WAIT, S_SEND);
    signal state : state_t;

begin

    ----------------------------------
    -- Divisor de frecuencia (1kHz) --
    ----------------------------------

    process(clk, reset)
    begin
        if reset = '1' then
            cnt_1kHz <= 0;
        elsif clk'event and clk = '1' then
            if en_1kHz = '1' then
                if cnt_1kHz = C_MAX_1kHz-1 then
                    cnt_1kHz <= 0;
                else
                    cnt_1kHz <= cnt_1kHz + 1;
                end if;
            end if;
        end if;
    end process;

    ovf_1kHz <= '1' when en_1kHz = '1' and cnt_1kHz = C_MAX_1kHz-1 else '0';

    -------------------------
    -- Contador de dígitos --
    -------------------------

    process(clk, reset)
    begin
        if reset = '1' then
            digito <= (others => '0');
        elsif clk'event and clk = '1' then
            if ovf_1kHz = '1' then
                if digito = 7 then
                    digito <= (others => '0');
                else
                    digito <= digito + 1;
                end if;
            end if;
        end if;
    end process;

```

```
74      ovf_digito <= '1' when ovf_1kHz = '1' and digito = 7 else '0';
75
76      --------------------
77      -- FSM de control --
78      --------------------
79
80      process(clk, reset)
81      begin
82          if reset = '1' then
83              state <= S_WAIT;
84          elsif clk'event and clk = '1' then
85              case state is
86                  when S_WAIT =>
87                      if boton = '1' then
88                          state <= S_SEND;
89                      end if;
90                  when S_SEND =>
91                      if ovf_digito = '1' then
92                          state <= S_WAIT;
93                      end if;
94              end case;
95          end if;
96      end process;
97
98      -- Control del temporizador (Moore)
99      en_1kHz <= '1' when state = S_SEND else '0';
100     -- Control de habilitación del registro de desplazamiento (Mealy)
101     en <= '1' when (state = S_WAIT and boton = '1') or (state = S_SEND and ovf_1kHz =
         '1') else '0';
102     -- Control de la carga del registro de desplazamiento (Mealy)
103     load <= '1' when state = S_WAIT and boton = '1' else '0';
104     -- Control del modulador (Moore)
105     valido <= en_1kHz;
106
107 end behavioral;
108
```

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

entity mando is
    port (
        clk   : in  std_logic;
        reset : in  std_logic;
        boton : in  std_logic;
        clave : in  std_logic_vector(7 downto 0);
        fsk   : out std_logic
    );
end mando;

architecture structural of mando is

    component entradas is
        port (
            clk     : in  std_logic;
            reset   : in  std_logic;
            boton   : in  std_logic;
            clave   : in  std_logic_vector(7 downto 0);
            boton_s : out std_logic;
            clave_s : out std_logic_vector(7 downto 0)
        );
    end component;

    component par2ser is
        port (
            clk   : in  std_logic;
            reset : in  std_logic;
            load  : in  std_logic;
            en    : in  std_logic;
            din   : in  std_logic_vector(7 downto 0);
            dout  : out std_logic
        );
    end component;

    component control is
        port (
            clk    : in  std_logic;
            reset  : in  std_logic;
            boton  : in  std_logic;
            load   : out std_logic;
            en     : out std_logic;
            valido : out std_logic
        );
    end component;

    component modulador is
        port (
            clk    : in  std_logic;
            reset  : in  std_logic;
            dato   : in  std_logic;
            valido : in  std_logic;
            fsk    : out std_logic
        );
    end component;

    -- Señales auxiliares
    signal boton_s : std_logic;
    signal clave_s : std_logic_vector(7 downto 0);
    signal load    : std_logic;
    signal enable  : std_logic;
    signal dato    : std_logic;
    signal valido  : std_logic;

begin

    entradas_i: entradas
    port map (
        clk     => clk,
        reset   => reset,
        boton   => boton,
```

```
74          clave   => clave,
75          boton_s => boton_s,
76          clave_s => clave_s
77      );
78
79      par2ser_i: par2ser
80      port map (
81          clk   => clk,
82          reset => reset,
83          load  => load,
84          en    => enable,
85          din   => clave_s,
86          dout  => dato
87      );
88
89      control_i: control
90      port map (
91          clk    => clk,
92          reset  => reset,
93          boton  => boton_s,
94          load   => load,
95          en     => enable,
96          valido => valido
97      );
98
99      modulador_i: modulador
100     port map (
101         clk    => clk,
102         reset  => reset,
103         dato   => dato,
104         valido => valido,
105         fsk    => fsk
106     );
107
108 end structural;
109
```