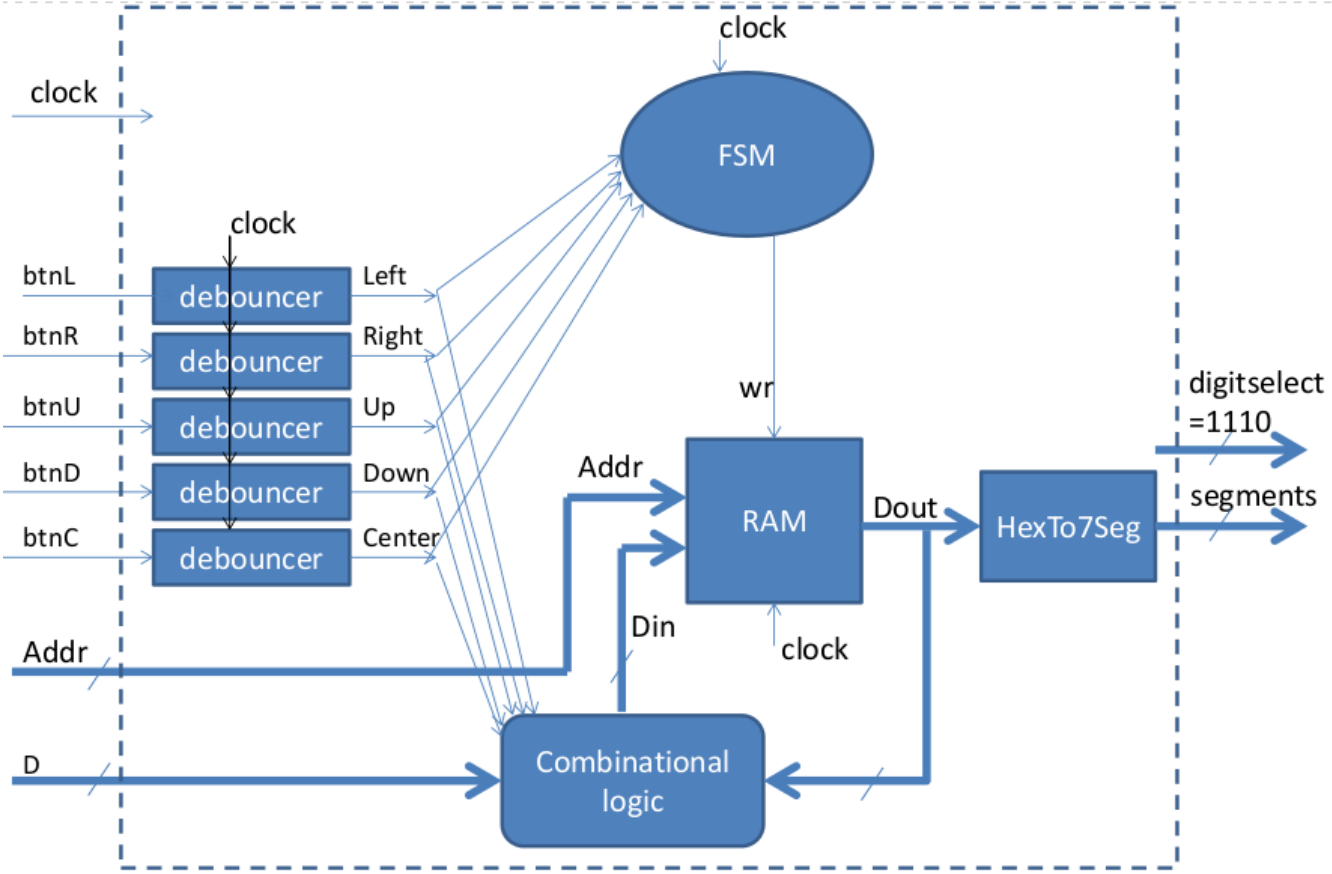
Trabajando con memorias (RAM)

Aprenderá lo siguiente en esta práctica de laboratorio:

- Diseñar una unidad de memoria simple (RAM)
- Más práctica con máquinas de estado (FSM)
- Más práctica con los interruptores anti-rebotes
- Más práctica con pantallas de 7 segmentos

En esta asignación de laboratorio, debe diseñar un sistema que contenga un pequeño módulo RAM y probarlo llenándolo con valores, y leerlos y modificarlos repetidamente. Las entradas de datos a su sistema son: (i) una dirección para seleccionar qué ubicación de memoria se está leyendo/escribiendo; y (ii) datos a escribir en la ubicación de la memoria (si no está escrito). La salida de su sistema es el valor de datos en la ubicación en la que se está abordando la memoria (es decir, el valor de lectura), que se muestra en una pantalla de 7 segmentos.

El siguiente diagrama de bloques muestra el sistema completo. ***Por favor, estudiarlo con cuidado!***



Las entradas y salidas del sistema son las siguientes:

- reloj: entregado a todos los componentes sincronizados dentro del sistema.
- Botones de presión: entradas de los cinco botones de la tarjeta, rotulados btnL, btnR, btnU, btnD y btnC (centro).
- Addr: un valor de 4 bits que proporciona la dirección de la ubicación de memoria a la que se hace referencia. Este valor se ingresa en la tarjeta a través de los cuatro controles deslizantes del extremo izquierdo. Por lo tanto, esta asignación solo necesita una unidad de memoria con 16 ubicaciones.
- D: un valor de 4 bits que proporciona los datos para almacenar en la memoria. Por lo tanto, cada palabra en la memoria es un único dígito hexadecimal [0..F]. Este valor se ingresa en la tarjeta a través de los cuatro controles deslizantes a la derecha.
- segmentos: el valor leído de la memoria se codifica para la salida en la pantalla de 7 segmentos.
- digitselect: Un valor de salida constante que selecciona el extremo derecho de los cuatro caracteres de 7 segmentos.

El comportamiento del sistema es el siguiente:

- El usuario selecciona la ubicación de la memoria (0 a 15) deslizando los interruptores correspondientes a *Addr*.
- Cuando el usuario proporciona *Addr*, la unidad de memoria genera el valor almacenado en esa ubicación, *mem[Addr]*. Este es un valor de 4 bits que se muestra como un carácter en la visualización del segmento.
- Si el usuario presiona uno de los botones: Se modifica el valor en la ubicación de la memoria de referencia. Su nuevo valor depende de qué botón se presionó, de la siguiente manera:

    - botón central: el valor de la memoria se establece en cero
    - botón arriba: el valor de la memoria se incrementa en 1
    - botón Abajo: el valor de la memoria se reduce en 1
    - botón izquierdo: el valor de la memoria es AND bit a bit con el valor D
    - botón derecho: el valor de la memoria es OR bit a bit con el valor D
    - NOTA: Para cada uno de estos escenarios, la operación de escritura de memoria debe tener lugar cuando se presiona el botón; posteriormente, su máquina de estado debe esperar hasta que se suelte el botón antes de volver a su estado inicial.

Su diseño debe seguir estas pautas:

- Diseñe el módulo de codificador HexTo7Seg.
- Use el módulo antirrebote suministrado. Necesitará cinco instancias separadas.
- Se proporciona un código para ram_module.
- Para el FSM, primero dibuje un diagrama con estados y transiciones. No debe necesitar más de seis estados: un estado inicial y un estado para representar cada uno de los cinco botones presionados. El FSM debería volver a su estado inicial una vez que se suelte el botón. ¡Esto es importante! De otra manera, terminará escribiendo la memoria miles de veces durante cada presión de botón.
- Para la lógica combinacional en la parte inferior del diagrama de bloques, escriba una pieza simple

de código que genere el valor correcto que se almacenará en la memoria, es decir, 0, o el valor (actual + 1), o el valor (actual - 1), o ... etc.

Qué entregar:

- Una copia impresa del diagrama de estado para el FSM.
- Una copia impresa de su fuente de VHDL para el FSM y el bloque lógico combinacional.
- Muestre una demostración funcional de su diseño.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity debounce is
    Port ( clk,reset : in  STD_LOGIC;
           sw : in  STD_LOGIC;
           db_level,db_tick : out  STD_LOGIC);
end debounce;

architecture Behavioral of debounce is
   constant N: integer:=21; --filtro de 2^N*20ns = 41ms (si el clk=50MHz)
   type estados is (cero,espera0,uno,espera1);
   signal estado_reg,estado_next:estados;
   signal q_reg,q_next:unsigned(N-1 downto 0);
begin
process(clk,reset)
begin
    if(reset='1') then
        estado_reg <= cero;
        q_reg <= (others=>'0');
    elsif(clk'event and clk='1') then
        estado_reg <= estado_next;
        q_reg <= q_next;
    end if;
end process;

process(estado_reg,q_reg,sw,q_next)
begin
        estado_next <= estado_reg;
        q_next <= q_reg;
        db_tick <= '0';
        case estado_reg is
            when cero =>
                db_level <= '0';
                if(sw = '1') then
                    estado_next <= espera1;
                    q_next <= (others=>'1');
                end if;
            when espera1 =>
                db_level <= '0';
                if(sw = '1') then
                    q_next <= q_reg - 1;
                    if(q_next = 0) then
                        estado_next <= uno;
                        db_tick <= '1';
                    end if;
                else    -- sw='0'
                    estado_next <= cero;
                end if;
            when uno =>
                db_level <= '1';
                if(sw = '0') then
                    estado_next <= espera0;
                    q_next <= (others=>'1');
                end if;
            when espera0 =>
                db_level <= '1';
                if(sw = '0') then
                    q_next <= q_reg - 1;
                    if(q_next = 0) then
                        estado_next <= cero;
                    end if;
                else -- sw='1'
                    estado_next <= uno;
                end if;
        end case;
end process;

end Behavioral;
```

```
-- Simple generic RAM Model
--
-- +-----------------------------+
-- |    Copyright 2008 DOULOS    |
-- |   designer :  JK            |
-- +-----------------------------+

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;

entity sync_ram is
  port (
    clock   : in  std_logic;
    we      : in  std_logic;
    address : in  std_logic_vector(3 downto 0);
    datain  : in  std_logic_vector(3 downto 0);
    dataout : out std_logic_vector(3 downto 0)
  );
end entity sync_ram;

architecture RTL of sync_ram is

   type ram_type is array (0 to (2**address'length)-1) of
std_logic_vector(datain'range);
   signal ram : ram_type;
   signal read_address : std_logic_vector(address'range);

begin

  RamProc: process(clock) is

  begin
    if rising_edge(clock) then
      if we = '1' then
        ram(to_integer(unsigned(address))) <= datain;
      end if;
      read_address <= address;
    end if;
  end process RamProc;

  dataout <= ram(to_integer(unsigned(read_address)));

end architecture RTL;
```