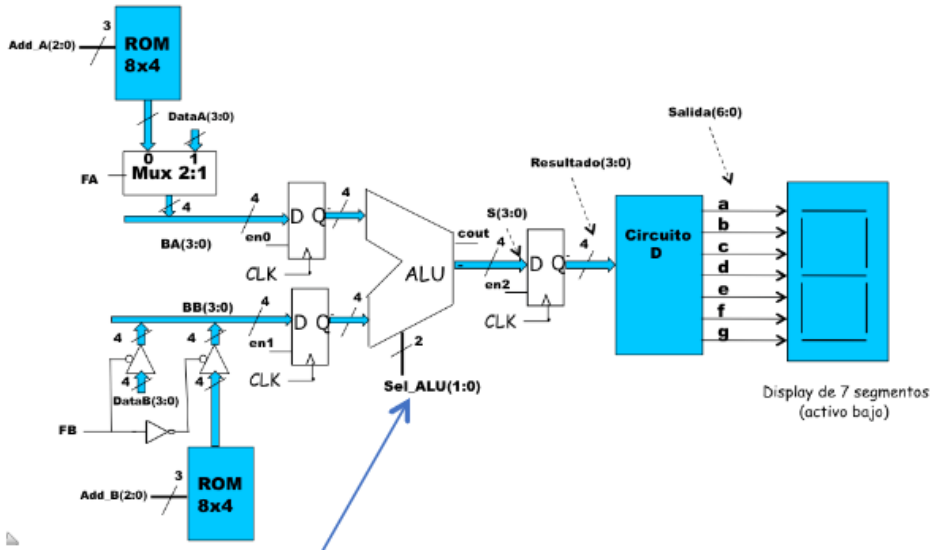
Diseño con Flip Flops

Angel Santiago Graciano Espitia, Kevin Jimenez Rincon
Facultad de ingeniería, Universidad de Antioquia
Medellín, Colombia
angel.graciano@udea.edu.co
kevin.jimenezr@udea.edu.co

Introducción

En este informe se relata el proceso,el circuito con flip flops que tiene como componentes destacados, roms y flip flops.

Circuito flip flops



En este para este circuito se tienen numerosas entradas, varias de control y otras que son las que el sistema va a operar o procesar, principalmente se tiene dos flip flops tipo D que son los flips flops de más fácil funcionamiento. Luego otra novedad en este circuito son las memorias roms que en este circuito hay 2 de estas.

Para este circuito vamos a reciclar código como el del ALU y el del display.
donde vamos a crear nuevos componentes como el de los roms porque cada uno tiene diferente memoria almacenada.

También vamos a regular el tiempo de control de este porque está en el orden de los MHz.

```
if(clk_interno'event and clk_interno='1') then
        if (count_clk = P100MSE) then
           count_clk <= 0;
           CLK_1Hz <= not CLK_1Hz;
    else
   count_clk <= count_clk +1;
```

Donde P100SE tiene un valor de 5000000.
Valores Genéricos

```
generic(
M:integer:=3;
N:integer:=4
);
```

Entradas y Salidas

Entradas de control: sel_alu, EN0, EN1,FA , FB, rst.
-sel_alu es la que nos indica qué operación va hacer el ALU.
-EN0 es la que activa el registro del flip flop del del rom y del driver tri-estado,
-FA y FB señales de control del mux y del driver tri-estado.
-rst pone valores iniciales en el circuito

Entradas para procesar: add_A, add_b, dataA, dataB.
-add_A y add_B señales de la direccion del rom.
-dataA y dataB señales de entrada que se van a controlar.

Reloj: clk.
salida: O display.

```
port(
    FA: in std_logic;
    FB: in std_logic;
    EN0: in std_logic ;
    EN1: in std_logic ;
    sel_alu: in std_logic_vector (2 downto 0);
    add_A: in std_logic_vector (M-1 downto 0);
    add_b: in std_logic_vector (M-1 downto 0);
    dataA: in std_logic_vector (N-1 downto 0);
    dataB: in std_logic_vector (N-1 downto 0);
    clk: in std_logic ;
    rst: in std_logic ;
    O: out std_logic_vector (6 downto 0)
```

Señales Internas del Circuito.

```
signal ba: std_logic_vector(N-1 downto 0);
signal bb: std_logic_vector(N-1downto 0);
signal ra: std_logic_vector(N-1downto 0);
signal rb: std_logic_vector(N-1downto 0);
signal ralu: std_logic_vector(N-1 downto 0);
signal s: std_logic_vector(N-1 downto 0);
signal clk_interno: std_logic;
signal carry: std_logic;
signal Dis: std_logic_vector(6 downto 0);
```

Registros.

El registro lo que busca es guardar los valores que nos entrega primeramente los driver tri-estado y el mux, pero para que estos funcionen el enable necesita estar en activo. por ello el enable de ambos es el mismo porque los datos se van a capturar al mismo tiempo. Para el registro que sale del alu, el enable 1 necesita estar encendido. Recordemos que este trabajo se hace en dos ciclos de reloj que definimos con el divisor de reloj, donde en un ciclo en0 está activo pero el en1 “0”, y en el segundo, en0 en “0” y en1 en “1”.

Registro al que le entra la señal que sale del mux.

```
process(ClK_1Hz , rst)
  begin
      if(rst='1') then
      ra <= (others => '0');
          elsif(CLK_1Hz'event and CLK_1Hz='1') then
              if(en0 = '1') then
                 RA <= BA;
              end if;
       end if;
end process;
```

Registro al que le entra la señal que sale del driver tri-estado.

```
process(ClK_1Hz , rst)
  begin
      if(rst='1') then
      rb <= (others => '0');
          elsif (CLK_1Hz'event and CLK_1Hz='1') then
        if(en0 = '1') then
        Rb <= Bb;
        end if;
    end if;
end process;
```

Registro al que le entra la señal que sale del driver tri-estado.

```
process(ClK_1Hz , rst)
  begin
      if(rst='1') then
      ralu <= (others => '0');
          elsif (CLK_1Hz'event and CLK_1Hz='1') then
        if(en1 = '1') then
        ralu <= s;
        end if;
    end if;
end process;
```

Diseño ROM

Para este diseño de rom se declara una señal de entrada y de salida, correspondiente a la dirección de memoria a la que queremos ingresar. Y la salida es lo que tenga almacenada esta dirección de memoria. También en esta se utilizan variables genéricas dinámicas que ya tenemos predeterminadas. que se usarán para modelar el tamaño del arreglo y vectores binarios que tienen almacenados.

```
entity ROM_A is
    generic (
        M: integer := 8;
        N: integer := 4
    );
    port (
        add_a : in std_logic_vector(M-1 downto 0);
        outra : out std_logic_vector(N-1 downto 0)
    );
end entity ROM_A;
```

Realmente las 2 roms se construyen de manera similar, lo único en que se diferencia es que tienen diferentes datos almacenados. por ejemplo para este se utiliza el sistema hexadecimal para guardar aquellos vectores binarios. Se utilizan comandos como type y array para construir este y constant para asegurar que este no va a cambiar. Luego se podra indexar a esta.

```
architecture Behavioral of ROM_A is
    type rom_type is array (0 to 2**M-1) of std_logic_vector(N-1 downto 0);
    constant rom : rom_type := (
       X"A",
       X"C",
       X"D",
       X"2",
       X"4",
       X"6",
       X"7",
       X"3"
    );
begin
    process(add_a)
    begin
        Outra <= rom(conv_integer(unsigned(add_a)));
    end process;
end Behavioral;
```

Declaración de Componentes y Conexión De Señales Internas

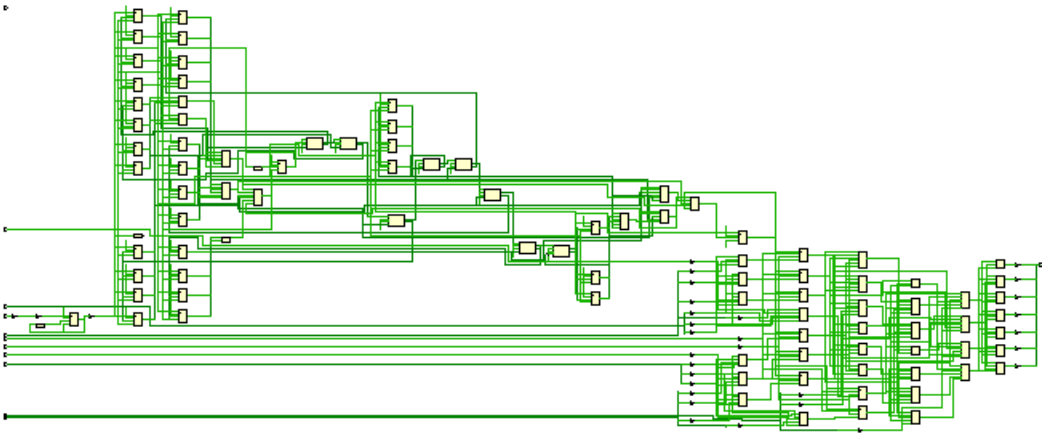
```
component ROM_A is
generic(
M:integer:=3;
N:integer:=4
);
port(
add_A: in std_logic_vector (M-1 downto 0);
outra: out std_logic_vector (N -1 downto 0)
);end component;

component ROM_B is
generic(
M:integer:=3;
N:integer:=4
);
port(
add_B: in std_logic_vector (M-1 downto 0);
outrb: out std_logic_vector (N -1 downto 0)
);
```

```
component ALU is
port(
A, B : in  std_logic_vector(3 downto 0);
          xyz : in  std_logic_vector(2 downto 0);
          cout: out std_logic :='0';
          S : out std_logic_vector(3 downto 0)
);
end component;
```

```
RomA : ROM_A port map(
    add_a => add_a,
    outra => ROMAout
    );
 RomB : ROM_B port map(
    add_b => add_b,
    outrb => ROMbout
    );
ALU1: ALU port  map(
    A => ra,
    B => rb,
    xyz => sel_alu,
    cout => carry,
    s => s
```

Síntesis del Circuito



RTL Análisis