Emmanuel Rojas Fredini

Trabajo Practico 1: VHDL

El trabajo consistió en el diseño y síntesis de una calculadora, la cual sea capaz de operar sobre 2 números, cada uno de estos de 1 digito. Las operaciones posibles son:

- Suma
- Resta
- Multiplicación

La entrada a la calculadora se realizaran de forma serie.

Análisis y Síntesis

Para realizar la consigna se diseño una maquina de estado finito(circuito secuencial) que llamamos CalculadoraFSM. Esta máquina consiste de varios estados, los cuales pueden agruparse resumidamente en:

1. Lectura de entrada en buffer de entrada.
2. Convertir de código bcd a sistema binario.
3. Realizar la operación requerida mediante el modulo de aritmética.
4. Esperar un reset para volver a usar la calculadora.

La máquina de estados esta sincronizada por un clock que se recibe como entrada y el cambio de estado se da en el flanco creciente de la señal. La maquina además posee un reset asincrónico que le permite volver al estado inicial de lectura de los operandos y operación a realizar.

El esquemático del circuito mencionado es el siguiente:

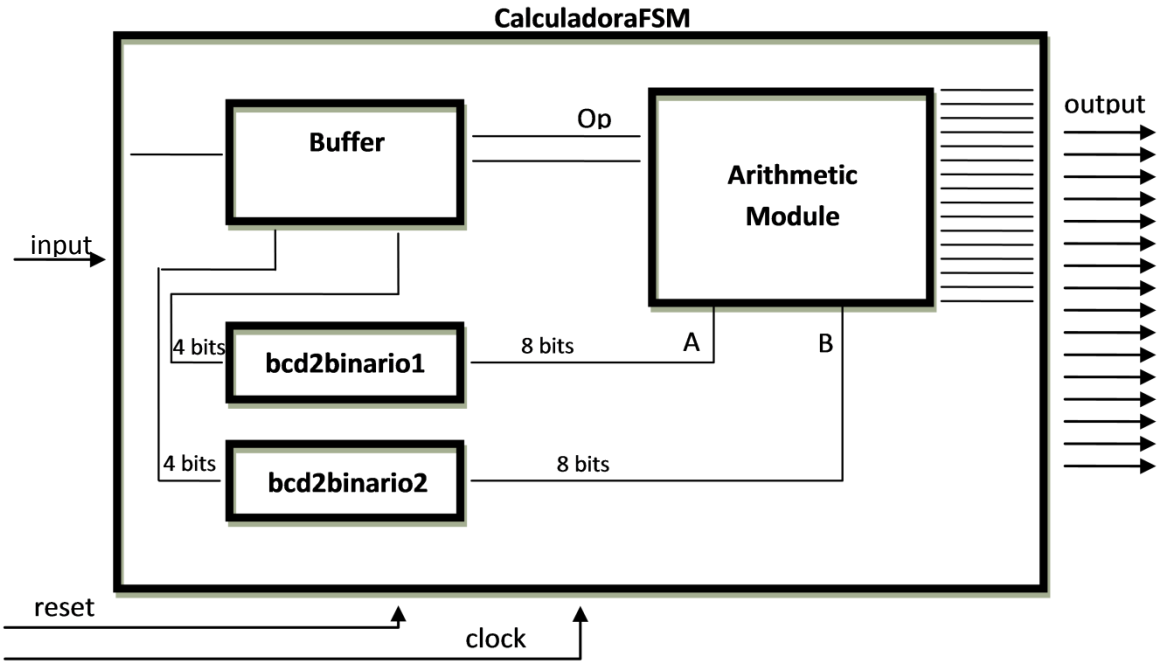


Ilustración 1: Esquemático del circuito

Buffer:

Consiste en un conjunto de señales binarias que almacenaran la entrada de la calculadora que se realiza en forma serie. El orden de los datos será:

4 bits de bcd de operando A → 2 bits de codigo de operacion → 4 bits de bcd de operando B

Los datos se leerán de forma little-endian, es decir que los bits menos significativos se leen primero.

bcd2binario:

Habrá 2 instancias de un modulo de conversión de formato bcd a sistema binario que consiste en un circuito combinacional, este se encargara de convertir los operandos que se reciben en formato bcd de la entrada almacenada en Buffer a sistema binario que es la entrada para el modulo de aritmética.

Arithmetic Module:

El modulo de aritmética es un circuito combinacional que se encarga de realizar las operaciones antes mencionadas(suma, resta y multiplicación) de la calculadora. Recibe dos operandos en sistema binario y un código de operación con el siguiente significado:

- 00 -> Suma
- 01 -> Resta
- 10 -> Multiplicación
- 11 -> No valido produce salida 1111111111111111

Códigos VHDL

CalculadoraFSM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Descripcion:
--La calculadora recibe su entrada en forma serie, recibiendo cada uno de los
--datos en forma little endian. Recibe los datos en el siguiente orden:
--                                          4 bits de un primer operando en formato bcd
--                                          2 bits del codigo del operador
--                                          4 bits de un segundo operando en formato bcd
--Para volver a realizar otra operacion hay que enviar un reset a la calculadora ya que esta
--una vez que produce el resultado, se queda en un estado de espera infinito.
--
--El cambio de estado normal es sincronico pero el reset cambia el estado asincronicamente.
--El ouput de la calculadora antes de que se realize todo el input y luego la operacion
--aritmetica no esta establecido asi que no deberia tenerselo en cuenta.

entity CalculadoraFSM is
    Port ( input : in std_logic;--Entrada seria a la calculadora
           clock : in  std_logic;--Clock de la maquina de estado finito
           reset : in  std_logic;--Reset asincronico de la calculadora
          output : out std_logic_vector(15 downto 0));--resultado de la calculadora
end CalculadoraFSM;

architecture CalculadoraStateMachineArchitecture of CalculadoraFSM is
       type estado is(LeerBit1Op1, LeerBit2Op1, LeerBit3Op1, LeerBit4Op1,
                      LeerOpCodeBit1, LeerOpCodeBit2,
                      LeerBit1Op2, LeerBit2Op2, LeerBit3Op2, LeerBit4Op2,
                      ConvertirBCD2Binario, Calcular, EsperarReset);

       --Estado actual
       signal estado_A: estado := LeerBit1Op1;
       --Estado siguiente
       signal estado_S: estado;

       --Componente de aritmetica
       COMPONENT ArithmeticModule
       PORT(
         A : IN  std_logic_vector(7 downto 0);
         B : IN  std_logic_vector(7 downto 0);
         Op : IN  std_logic_vector(1 downto 0);
         Res : OUT  std_logic_vector(15 downto 0)
        );
       END COMPONENT;

       --ArithmeticModule Inputs
   signal A : std_logic_vector(7 downto 0) := (others => '0');
   signal B : std_logic_vector(7 downto 0) := (others => '0');
   signal Op : std_logic_vector(1 downto 0) := (others => '0');

       --ArithmeticModule Outputs
   signal Res : std_logic_vector(15 downto 0);

       --Componente de conversion de bcd a formato numerico binario
       COMPONENT BCD2Binary
       Port ( bcd : in  STD_LOGIC_VECTOR(3 downto 0);
          binary : out  STD_LOGIC_VECTOR(7 downto 0));
       END COMPONENT;

       --BCD2Binary 1 Inputs
   signal bcd1  : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
```

```
        --BCD2Binary 1 Outputs
    signal binary1 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');

        --BCD2Binary 2 Inputs
    signal bcd2  : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');

        --BCD2Binary 2 Outputs
    signal binary2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');


        --Almacena la entrada a la calculadora
        --No es totalmente necesario porque se podria cargar directamente en bcd1, bcd2 y Op
        --pero cargarlo en este buffer hace que el proceso sea mas facil de entender
        signal bufferEntrada : STD_LOGIC_VECTOR(9 downto 0);

begin

        -- Instanciamos el Modulo de Aritmetica(AM)
        am: ArithmeticModule PORT MAP (
           A => A,
           B => B,
           Op => Op,
            Res => output
         );

        -- Instanciamos el Conversor de BCD a binario 1(bcd2binario1)
        bcd2binario1: BCD2Binary PORT MAP (
           bcd => bcd1,
           binary => binary1
         );

        -- Instanciamos el Conversor de BCD a binario 2(bcd2binario2)
        bcd2binario2: BCD2Binary PORT MAP (
           bcd => bcd2,
           binary => binary2
         );

        --Proceso del siguiente estado de la maquina de estado finito
        SiguienteEstado: process(estado_A, input)
                begin
                        case estado_A is
                                --Estados de lectura de la entrada a la calculadora

                                when LeerBit1Op1 =>
                                        bufferEntrada(0) <= input;
                                        estado_S <= LeerBit2Op1;
                                when LeerBit2Op1 =>
                                        bufferEntrada(1) <= input;
                                        estado_S <= LeerBit3Op1;
                                when LeerBit3Op1 =>
                                        bufferEntrada(2) <= input;
                                        estado_S <= LeerBit4Op1;
                                when LeerBit4Op1 =>
                                        bufferEntrada(3) <= input;
                                        estado_S <= LeerOpCodeBit1;
                                when LeerOpCodeBit1 =>
                                        bufferEntrada(4) <= input;
                                        estado_S <= LeerOpCodeBit2;
                                when LeerOpCodeBit2 =>
                                        bufferEntrada(5) <= input;
                                        estado_S <= LeerBit1Op2;
                                when LeerBit1Op2 =>
                                        bufferEntrada(6) <= input;
                                        estado_S <= LeerBit2Op2;
                                when LeerBit2Op2 =>
                                        bufferEntrada(7) <= input;
                                        estado_S <= LeerBit3Op2;
                                when LeerBit3Op2 =>
                                        bufferEntrada(8) <= input;
                                        estado_S <= LeerBit4Op2;
```

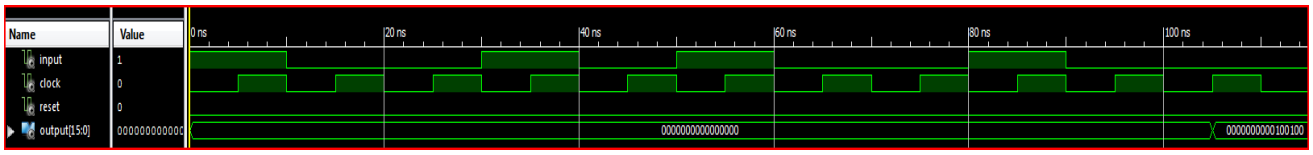
```
                        when LeerBit4Op2 =>
                                bufferEntrada(9) <= input;
                                estado_S <= ConvertirBCD2Binario;
                        --Conversion de los operandos de formato bcd a binario

                        when ConvertirBCD2Binario =>--Convierte los valores
                                bcd1 <= bufferEntrada(3 downto 0);
                                bcd2 <= bufferEntrada(9 downto 6);
                                estado_S <= Calcular;

                        when Calcular =>--Realiza los calculos
                                A <= binary1;
                                OP <= bufferEntrada(5 downto 4);
                                B <= binary2;
                                estado_S <= EsperarReset;

                        when EsperarReset =>--Espera que se resetee
                                estado_S <= EsperarReset;
                end case;
        end process SiguienteEstado;

    --Proceso de cambio de estado de la maquina de estado finito
    CambioEstadoSincronico: process
        begin
                wait until reset='1' or rising_edge(clock);
                if reset='1' then
                        estado_A <= LeerBit1Op1;--resetea la maquina de estado finito
                else
                        estado_A <= estado_S;--actualiza el estado actual al siguiente
                end if;
        end process CambioEstadoSincronico;


end CalculadoraStateMachineArchitecture;
```

Pruebas

1. **Multiplicación:**

   Entrada: bcd 1001b -> Ósea 9 Op 10b bcd 0100b -> Ósea 4

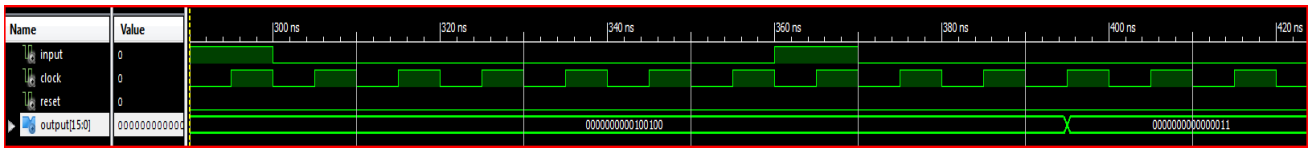
   Resultado esperado 100100b -> Ósea 36



2. **Suma:**

   Entrada: bcd 0001b -> Ósea 1 Op 00b bcd 0010b -> Ósea 2

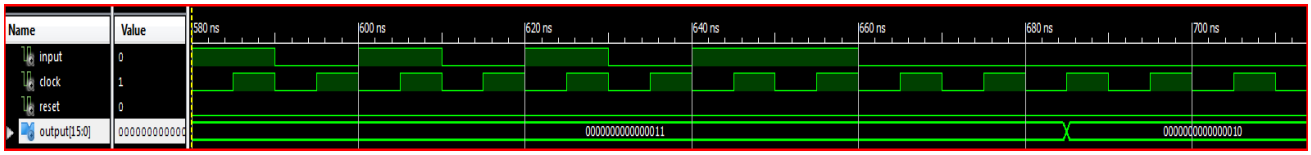
   Resultado esperado 0011b -> Ósea 3



3. **Resta:**

   Entrada: bcd 0101b -> Ósea 5 Op 01b bcd 0011b -> Ósea 3

   Resultado esperado 0010b -> Ósea 2



ArithmeticModule:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity ArithmeticModule is
    Port ( A : in  STD_LOGIC_VECTOR(7 downto 0);--Primer operando(valor binario)
           B : in  STD_LOGIC_VECTOR(7 downto 0);--Segundo operando(valor binario)
           Op : in  STD_LOGIC_VECTOR(1 downto 0);--Operador
           Res : out  STD_LOGIC_VECTOR(15 downto 0));--Resultado de la operacion
end ArithmeticModule;

architecture ArithmeticArchitecture of ArithmeticModule is
begin

	PROCESS(A, B, Op)
	BEGIN
		CASE Op IS
			WHEN "00" => --Suma
				Res(7 downto 0) <= A+B;
				Res(15 downto 8) <= "00000000";
			WHEN "01" => --Resta
				Res(7 downto 0) <= A-B;
				Res(15 downto 8) <= "00000000";
			WHEN "10" =>--Multiplicacion
				Res <= A*B;
			WHEN "11" =>
				Res <= "1111111111111111";
			when others =>
				Res <= "1111111111111111";
		END CASE;
	END PROCESS;


end ArithmeticArchitecture;
```

BCD2Binary:

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

entity BCD2Binary is
    Port ( bcd : in  STD_LOGIC_VECTOR(3 downto 0);
           binary : out  STD_LOGIC_VECTOR(7 downto 0));
end BCD2Binary;

architecture BCD2BinaryArchitecture of BCD2Binary is
begin
       binary(3 downto 0) <= bcd;
       binary(7 downto 4) <= "0000";
end BCD2BinaryArchitecture;
```