Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
IE-0524 Laboratorio de Circuitos Digitales

Anteproyecto Experimento 2

José Carlos Cortés Mora, A72115
Berni Mora Escobar, A94096
Esteban Vargas Arrieta, A96486

Profesor: Diego Valverde

30 de marzo de 2016

1. Solución Propuesta del Problema

1.1. Ejercicio 1

Se ha agregado las dos funciones de multiplicación con y sin signo dentro del archivo MiniAlu.v. La multiplicación sin signo resulta ser más simple, ya que solo se ha utilizado el operador (*) entre dos wires y se ha pasado el resultado al registro result, tal como se observa a continuación.

```
1   `MUL:
2   begin
3     rFFLedEN      <= 1'b0;
4     rBranchTaken <= 1'b0;
5     rWriteEnable <= 1'b1;
6     rResult       <= wSourceData1 * wSourceData0;  //Multiplicacion sin signo
7   end
```

Listing 1: Multiplicación sin signo

Como se observa solo hace falta habilitar la señal de rWriteEnable. Para implementar la multiplicación con signo se requiere un poco más de trabajo. Lo primero es definir dos wires nuevos los cuales se encargaran de almacenar los números que serán considerados como números con signo, tal y como se muestra en 2.

```
1 wire  signed[15:0] wsSourceData0,wsSourceData1;
```

Listing 2: Wires con Signo

Luego de definir los nuevos wires se debe hacer una copia de las entradas originales, y almacenarlas en estos para ser tratados como datos con signo.

```
1 assing  wsSourceData0 =  wSourceData0;
2 assing  wsSourceData1 =  wSourceData1;
```

Listing 3: Convierte los datos de wires sin signo a wires con signo

Al final se vuelve a realizar la multiplicación utilizando el operador (*), pero esta vez se pasa como sources los wires con signo, como se muestra a continuación.

```
1   `SMUL:
2   begin
3     rFFLedEN      <= 1'b0;
4     rBranchTaken <= 1'b0;
5     rWriteEnable <= 1'b1;
6     rsResult      <= wsSourceData1 * wsSourceData0;  //Multiplicacion con signo
7   end
```

Listing 4: Multiplicación con signo

1.2. Ejercicio 2

Al igual que el caso anterior la multiplicación de dos números de 4 bits se debe tener disponible para un buen funcionamiento una cantidad igual al doble de bits de cada operando para representar el resultado, es decir en este caso necesitaremos 8 bits para obtener el resultado. Además para las sumas que representa la multilpicación se debe tener en cuenta el acarreo por lo tanto se define un wire para el carry. La definiciones se muetran en 5.

```

  wire [7:0] wIMULResult;
  wire wCarry;
```

Listing 5: Drivers para IMUL.

```
//-----------------------------------------
  `IMUL:
  begin
    rFFLedEN      <= 1'b0;
    rBranchTaken <= 1'b0;
    rWriteEnable <= 1'b1;
    wResult[0]     <= wSourceData0[0]&wSourceData1[0];

    assign{wCarry, wIMULResult[1]}        <= wSourceData0[1]&wSourceData1[0]+
   wSourceData0[0]&wSourceData1[1];

    assign{wCarry,wIMULResult[2]}        <= wSourceData0[2]&wSourceData1[0]+
   wSourceData0[1]&wSourceData1[1]+wSourceData0[0]&wSourceData1[2]+wCarry;

    assign{wCarry,wIMULResult[3]}        <= wSourceData0[3]&wSourceData1[0]+
   wSourceData0[2]&wSourceData1[1]+wSourceData0[1]&wSourceData1[2]+wSourceData0[0]&
   wSourceData1[3]+wCarry;

    assign{wCarry,wIMULResult[4]}        <= wSourceData0[3]&wSource Data1[1]+
   wSourceData0[2]&wSourceData1[2]+wSourceData0[1]&wSourceData1[3]+wCarry;

    assign{wCarry,wIMULResult[5]}        <= wSourceData0[3]&wSourceData1[2]+
   wSourceData0[2]&wSourceData1[3]+wCarry;

    assign{wCarry,wIMULResult[6]}        <= wSourceData0[3]&wSourceData1[3]+wCarry;

   wIMULResult[7]      <= wCarry;

   rResult  <= 16'd0; //Los bits que no se utilizan en rRewsult deben ser 0
   rResult[7:0]   <= wIMULResult; //El resultado de la multiplicacion se pone en el
    bus de salida rResult
  end
  //---------------------------------------
```

Listing 6: Multiplicación con arrays.

Si se desea llamar la instrucción IMUL, se puede realizar agregándose esta instrucción al ROM, tomandose en cuenta que aunque los parámetros de entrada son scr0 y src1 que son de 16 bits, la opración IMUL tal y como fue implementada solo realizara la multiplicación entre los 4 bits más significativos de cada palabra y además el resultado obtenido en rResult solo será válido en los 8 bits meno

```
.
.
.
//SIGUE2
    15: oInstruction = { `SMUL, `R5, `R6, `R7 };
```

Listing 7: Instrucciones de prueba.

1.3. Ejercicio 3

1.4. Ejercicio 4

Para realizar la multiplicación mediante el método LUT, se utilizara el mux mostrado en la figura 1

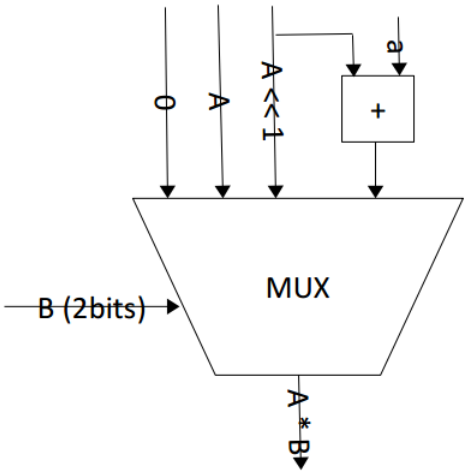

Figura 1: Mux Multiplicación LUT

Este mux permite implementar la multiplicación LUT de 2 bit y sirve de base para el desarrollo de multiplicadores LUT de más bits.

El código de verilog propuesto para la implementación del multiplexor es:

```
1
2  module MULT_LUT_2_BITS
3  (
4  input wire [15:0] iDato_A, //Entrada A - Hasta numeros de 16 Bits
5  input wire [1:0] iDato_B, // Entrada B - 2 Bits
6  output reg [15:0] oResult_Mux //Resultado A*B
7  );
8
9  always @ (*) {
10   case (iDATO_B_2_BITS):
11     0: oResult_Mux = 0; // multiplicacion por 0
12     1: oResult_Mux = iDato_A; // multiplicacion por 1
13     2: oResult_Mux = iDato_A << 1; // multiplicacion por 2
14     3: oResult_Mux = (iDato_A << 1) + iDato_A; // multiplicacion por 3
15 }
16 endmodule
```

Listing 8: Multiplicación LUT 2 Bits

Utilizando el principio de que una multiplicación se puede representar como la suma de dos multiplicaciones, se implementó un módulo multiplicador LUT para números de 4bits y 16 bits, A continuación se muestra el código Verilog para ambos casos.

```
1
2  module MULT_LUT_4_BITS
3  (
```

```
input wire [15:0] iDato_A_4bits; // Entrada A - Hasta numeros de 16 Bits
input wire [3:0] iDato_B_4bits; // Entrada B - 4 Bits
output reg [15:0] oResult_Mux_4bits; // A*B
);

wire [15:0] wBits_down, wBits_up;

// multiplicacion con dos bits menos significativos
MULT_LUT_2_BITS wBits_down
(
  .iDato_A( iDato_A_4bits ),
  .iDato_B( iDato_B_4bits[1:0] ),
  .oResult_Mux( wBits_down )
);

// multiplicacion con dos bits mas significativos
MULT_LUT_2_BITS wBits_up
(
  .iDato_A( iDato_A_4bits ),
  .iDato_B( iDato_B_4bits[3:2] ),
  .oResult_Mux( wBits_up )
);

assign oResult_Mux_4bits = wBits_down + (wBits_up << 2); //resultado A*B LUT 4 bits

endmodule
```

Listing 9: Multiplicación LUT 4 Bits.

Utilizando el mismo principio es posible implementar los módulos para números de 8 bits y de 16 bits. Finalmente se define una instancia del módulo de 16 bits, como se muestra a continuación

```
wire [31:0] Mult_LUT_Result; // Resultado Multiplicacion LUT

MULT_LUT_16_BITS Mult_LUT_Result
(
iDato_A(wSourceData2),
iDato_B(wSourceData1),
oResult_Mux(Mult_LUT_Result)
);
```

Listing 10: Instancia de modulo 16 Bits

```
  `IMUL2: // LUT multiplication
  begin
    rFFLedEN      <= 1'b0;
    rBranchTaken <= 1'b0;
    rDoComplement <= 1'b0;
    rWriteEnable <= 1'b1;
    rResult       <= Mult_LUT_Result;
  end
```

Listing 11: Implementacion de la instrucción IMUL2

2. Observaciones y Recomendaciones

- Consultar al profesor como realizar el ejercicio #3, ya que este no queda muy claro de como se debe implementar.