Universidad de Costa Rica
IE0424: Laboratorio de Circuiros Digitales

Anteproyecto Laboratorio 2

Guillermo Cornejo, Felipe Rincón, Juan Diego Sanchez
B22054, B25530, B16068

23 de marzo de 2015

1. Ejercicio 1: Operación SMUL

Anote la frecuencia que la herramienta de síntesis estima para el experimento 1, anote también el número de Look-up table, Slices y Flip-Flops. El código para multiplicación es similar al implementado en el experimento 1 para resta.

```
'SMUL:
  begin
      rFFLedEN      <= 1'b0;
      rBranchTaken  <= 1'b0;
      rWriteEnable  <= 1'b1;
      rResult       <= wSourceData1 * wSourceData0;
  end
```

Ahora es necesario modificar la declaración de *wires* y *registers* para que acepten valores con signo. Los cambios debe hacerse en `oDataOut0, oDataOut1, iDataIn,wSourceData0, wSourceData1, rResult, wImmediateValue`.

Anote la frecuencia que la herramienta de síntesis estima para esta parte, además el número de LUTs, Slices y Flip-Flops. Modifique el código en la ROM para calcular la multiplicación de varios números tanto con signo como sin signo y despliegue el resultado en los LEDs.

2. Ejercicio 2: Operación IMUL

Se solicita crear un multiplicador de 4 bits array multiplier. Se escribió el módulo *adder*, el cual es un sumador de un bit, con *carry-in* y *carry-out*.

```
module adder(
input a,
input b,
input ci,
output s,
output co
);

    wire a, b, ci;
    reg s, co;

    always @ (*)
    begin
        {co, s} = a + b + ci;
    end

endmodule
```

se crearán 12 instancias de este sumador y se cablearán de manera adecuada, siguiendo el diagrama dado en el enunciado. Las entradas serán `input [3:0] op0` y `input [3:0] op1`.

3. Ejercicio 3: Multiplicador de 16 bits

Para implementar un multiplicador de 16 bits, se utilizará la misma arquitectura que para la operación IMUL de 4 bits. En este caso se tienen que instanciar 240 módulos distribuidos en 15 etapas, por lo cual se utilizará la herramienta **generate** de Verilog, para automatizar este proceso. Esta herramienta facilita el proceso de crear los módulos sumadores, ya que se generan mediante iteraciones, tal y como se muestra en el siguiente código. En este se explica la solución propuesta.

```
module IMUL16 # (parameter NUM_BITS = 16)
(
input wire [NUM_BITS-1:0] A,
input wire [NUM_BITS-1:0] B,
output reg [2*NUM_BITS:0] oResult
);

wire [NUM_BITS-1:0] wCarry [NUM_BITS-1:0]; // matriz para conectar carry's
wire [NUM_BITS-1:0] Temp_results [NUM_BITS-1:0]; // matriz conectar sumadores

genvar CurrentRow, CurrentCol; // iteradores de columnas y filas

assign oResult[0]=A[0]&B[0]; // primer digito de la multiplicacion ya que es fijo
assign Temp_results[0][NUM_BITS-1]=0; //ultimo sumador de la primera fila es siempre 0

generate
    for(CurrentRow=0 CurrentRow<NUM_BITS;CurrentRow=CurrentRow+1);
    begins
        assign wCarry[CurrentRow][0];

        for(CurrentCol=0 CurrentCol<NUM_BITS;CurrentCol=CurrentCol+1);
            begin
            // Se guardan en Temp_results los primeros numeros en entrar a la primera
            //fila de sumadores
                if(CurrentRow==0 && CurrentCol<NUM_BITS-1)
                begin
                    Temp_results[0][CurrentCol]=A[CurrentCol+1]&B[0];
                end

            // Caso de los sumadores intermedios:
                //Se instancian de al forma que tomen sus sumandos de Temp_Result
                //(previos) y se guardan sus resultados tanto en Tem_Result como
                //en wCarry para cablear el resto de sumadores y niveles.
                if(CurrentCol<NUM_BITS-1 && CurrentRow<NUM_BITS-1)
                begin
                    if(CurrentCols!=0)
                    begin
                        adder Circuit_Adders(
                            a.(A[CurrentCol]&B[CurrentRow+1]),
                            b.(Temp_Results[CurrentRow][CurrentCol]),
                            s.(Temp_Results[CurrentRow+1][CurrentCol]),
                            ci.(wCarry[ CurrentRow ][ CurrentCol ] ),
                            co.(wCarry[ CurrentRow ][ CurrentCol + 1])
                        );
                    end
```

```
                    // Caso de los primeros sumadores para cada fila:
                    // En estos se cambian la salida para que se guarde
                    // de una vez en el resultado
                    else
                    begin
                        adder Circuit_Adders(
                            a.(A[CurrentCol]&B[CurrentRow+1]),
                            b.(Temp_Results[CurrentRow][CurrentCol]),
                            s.(oResult[CurrentRow+1]),
                            ci.(wCarry[ CurrentRow ][ CurrentCol ] ),
                            co.(wCarry[ CurrentRow ][ CurrentCol + 1])
                        );
                    end
                end
                //Caso del ultimo bloque sumador por fila:
                // Este es especial ya que su carry out se guarda en los Tem_Results
                // para poder sumarselo al ultimo bloque del siguiente nivel.
                if(CurrentCol==NUM_BITS-1 && CurrentRow<NUM_BITS-1)
                begin
                    adder Circuit_Adders(
                        a.(A[CurrentCol]&B[CurrentRow+1]),
                        b.(Temp_Results[CurrentRow][NUM_BITS-1]),
                        s.(Temp_Results[CurrentRow][NUM_BITS-1]),
                        ci.(wCarry[ CurrentRow ][ NUM_BITS-1 ] ),
                        co.(Temp_Results[ CurrentRow+1 ][ NUM_BITS-1])
                    );
                end

                // El ultimo if es para asignar los resultados finales a oResult

                if(CurrentRow==NUM_BITS-1)
                begin
                    oResult[CurrentCol+NUM_BITS]=Temp_Results[NUM_BITS-1][CurrentRow];
                end
            end
        end
    end
endgenerate
endmodule
```

Anote la frecuencia que la herramienta de síntesis estima para esta parte, además el numero de LUTs, Slices y Flip-Flops.

Explique ¿Qué ocurre con el periodo del reloj si se añaden más etapas de lógica combinatoria?

¿Qué ocurre con la frecuencia del circuito si añade Latches entre cada etapa de sumadores?

Modifique el código de la ROM para calcular la multiplicación de varios números sin signo y despliegue el resultado en los LEDs. Como solo se cuenta con 8 LEDs se buscará una forma alternativa de mostrar el resultado.

4. Ejercicio 4: IMUL con Look-up Table (IMUL2)

- ¿Cuántas filas y columnas tendría una LUT para multiplicar números de 32 bits?
  Para implementar una LUT binaria de multiplicación se necesitan 2^n filas (y columnas), donde $n = \#bits$, para este caso particular serían $2^{32} = 4294967296$ filas y columnas.

Ahora bien para implementar un circuito que multiplique dos números A y B de 4 dígitos se puede utilizar un circuito como el mostrado en la figura 1.

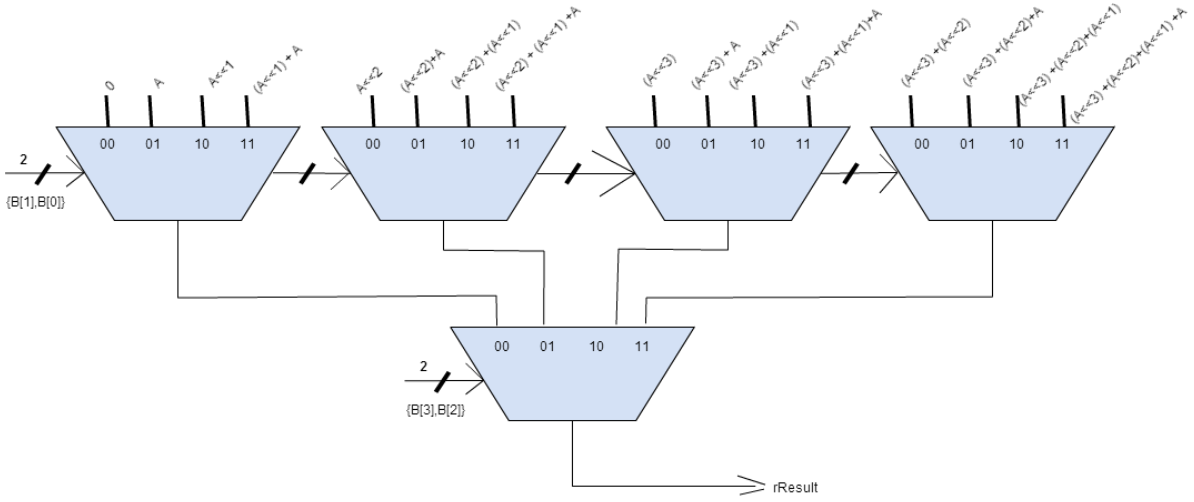

Figura 1: Circuito multiplicador para 2 números de 4 bits

La función se puede implementar en la MiniAlu, mediante el código:

```
'IMUL2:
begin
        rFFLedEN      <= 1'b0;
        rBranchTaken <= 1'b0;
    rWriteEnable <= 1'b1;
        rResult <= wSourceData1<< (3*wSourceData0[3]) +
        wSourceData1<<(2* wSourceData0[2])+ wSourceData1 <<(1*wSourceData0[1])+
        wSourceData1 * wSourceData0[0];
end
```

Por otra parte, si se desea multiplicar números de 16 bits solamente hay que agregar los bits [15:4] a la expresión anterior es decir:

```
'IMUL2:
begin
        rFFLedEN      <= 1'b0;
        rBranchTaken <= 1'b0;
        rWriteEnable <= 1'b1;
        rResult <= wSourceData1<< (15*wSourceData0[15])+
        wSourceData1<<(14*wSourceData0[14])+  ... +
        SourceData1 <<(1*wSourceData0[1])+
        wSourceData1 * wSourceData0[0];
end
```

5. Ejercicio 5

La función genérica IMUL implementada, permite la multiplicación de números de 4 bits y también de 2 bits. La función implementada con muxes de 4 bits se muestra en la figura 1.