Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0523 - Circuitos Digitales II
I ciclo 2023

Tarea 1

Sumador binario sincrónico de 4, 8 y 32 bits

Isaí David Vargas Ovares B88263
Grupo 1

Profesor: Enrique Coen Alfaro

16 de abril de 2023

1. Resumen

Se realizó un sumador binario sincronizado por una señal de reloj, con modos de suma, resta, mantener el valor anterior y limpiar el sumador. Esto teniendo una jerarquización que en un nivel primitivo se tienen los módulos de suma, resta, en un segundo nivel se tiene el módulo sumador de 4 bits, en un tercer nivel se tiene el módulo sumador de 8 bits que utiliza 2 sumadores de 4 bits, además en este nivel se tiene también un sumador de 32 bits que utiliza 8 sumadores de 4 bits.

Se establecieron algunas pruebas, para comprobar el correcto funcionamiento del sumador, en este caso se probaron los distintos modos de funcionamiento del sumador, introduciendo distintos valores a operar, tanto números de 4, 8 y 32 bits, imprimiendo los resultados en terminal para analizarlos y obteniendo los resultados esperados.

Las operaciones bit a bit son esenciales en la programación de sistemas embebidos y en el diseño de circuitos digitales. Estas operaciones se realizan a nivel de bits, lo que significa que se procesa cada bit de los operandos individualmente. Esto hace que las operaciones binarias sean muy eficientes en términos de velocidad y memoria, lo que es importante en los sistemas embebidos que tienen recursos limitados [1].

2. Descripción Arquitectónica

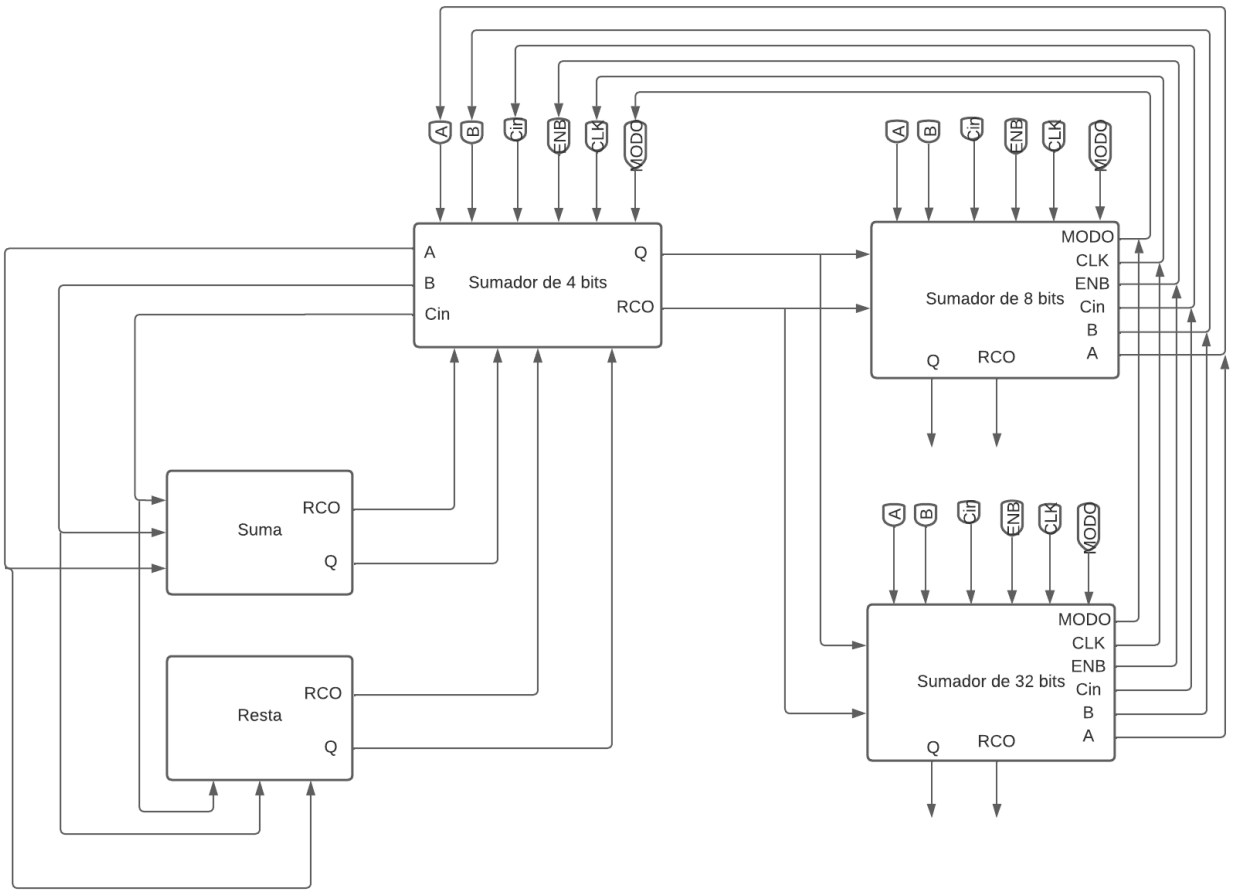

Figura 1: Diagrama de bloques con las señales más importantes.

Se diseña un Sumador binario sincrónico de 4, 8 y 32 bits, capaz de realizar suma, resta, mantener valor anterior y limpiar el sumador. En el diseño planteado se tiene en un primer nivel los módulos de suma y resta, estos reciben los números de 4 bits a operar y tienen a su salida el acarreo y el resultado de la operación, como se pueden ver en la Figura 2, que posteriormente serán instanciados en el módulo sumador de 4 bits.

```
1  module SUMA4 (A,B,cin,cout,suma);
2    input  [3:0] A,B;//Entradas de los nibbles a operar
3    input cin;//Entrada del acarreo inicial
4    output [3:0] suma; //Salida de la operacion de suma
5    output cout;//Salida del rebase
6    assign {cout,suma} = A + B + cin;//Operacion de suma
7  endmodule
8
9  module RESTA4 (A,B,cin,cout,resta);
10   input  [3:0] A,B;//Entradas de los nibbles a operar
11   input cin;//Entrada del acarreo inicial
12   output [3:0] resta; //Salida de la operacion de resta
13   output cout;//Salida del rebase
14   assign {cout,resta} = A - B - cin;//Operacion de resta
15 endmodule
```

Figura 2: Módulos de suma y resta.

Dado que se tiene un segundo nivel en la jerarquía que consta de un módulo sumador de 4 bits, este tiene como entradas:

- CLK – Entrada de reloj del sumador. El flanco activo de la señal CLK es el flanco creciente
- ENB – Entrada de habilitación del sumador.
- MODO[1:0] – Entrada de modo que consta de dos líneas y sirve para definir cuál será el próximo estado del sumador al llegar el flanco activo del reloj en la entrada CLK
- A[3:0] – Entrada de datos A consta de 4 líneas.
- B[3:0] – Entrada de datos B consta de 4 líneas.

Para las salidas se tiene:

- Q[3:0] – Salida Q que consta de cuatro líneas que indican el estado presente del sumador.
- RCO – Salida de llevo “Ripple-Carry Out” que indica cuando el sumador llega a su cuenta límite.

Se tienen además algunas consideraciones:

- La entrada MODO define si se realiza la suma, resta, mantener valor anterior o limpiar.
  - MODO = 00 → Q
  - MODO = 01 → A+B

- MODO = 10 → A-B
- MODO = 11 → 0

- Si ENB=1 el sumador funciona normalmente, si ENB=0 se mantiene el estado actual.

```
module Sumador(A,B,Cin,CLK,ENB,MODO,Q,RCO);

  input [3:0] A,B;//Entradas de los nibbles a operar
  input Cin,CLK,ENB;//Entrada del acarreo inicial, el reloj y enable
  input [1:0] MODO;//Seleccion del modo de operacion
  output reg [3:0] Q;//Salida de la operacion
  output reg RCO;//Acarreo de salida

  //Cables necesarios
  wire [3:0] out_suma;
  wire [3:0] out_resta;
  wire Carry1;
  wire Carry2;

  //instanciacion de los modulos de suma y resta
  SUMA4 sumar(.A(A), .B(B), .cin(Cin), .cout(Carry1), .suma(out_suma));
  RESTA4 resta(.A(A), .B(B), .cin(Cin), .cout(Carry2), .resta(out_resta));

    //Se ejecuta siempre que se tenga un flanco positivo de reloj
    always @ (posedge CLK) begin
      //Se ejecuta siempre que se tenga ENB=1
      if (ENB==1'b1) begin
        //Se asigna a la salida el resultado segun la operacion seleccionada en MODO
        case (MODO)
          2'b00: begin
            Q = Q;
          end
          2'b01: begin
            Q = out_suma;
            RCO=Carry1;
          end
          2'b10: begin
            Q = out_resta;
            RCO=Carry2;
          end
          2'b11: begin
            Q = 0;
            RCO=0;
          end
        endcase
      end
      else begin
        Q = Q;
      end
    end
endmodule
```

Figura 3: Módulo Sumador de 4 bits.

Dentro de este módulo Sumador de 4 bits se instancian los módulos de suma y resta, donde la salida de estos se asigna a la Salida Q y RCO dependiendo de la señal de ENB, MODO y el flanco del reloj.

```
module Sumador8(A,B,Cin,CLK,ENB,MODO,Q,RCO);

  input [7:0] A,B;//Entradas de los nibbles a operar
  input Cin,CLK,ENB;//Entrada del acarreo inicial, el reloj y enable
  input [1:0] MODO;//Seleccion del modo de operacion
  output reg [7:0] Q;//Salida de la operacion
  output reg RCO;//Acarreo de salida

  //Cables necesarios
  wire [3:0] Q1,Q2;
  wire Carry1,Carry2;

  //instanciacion del modulo de suma de 4 bits 2 veces para operar el total de bits
  //La salida de acarreo del primero se ingresa en el segundo
  //Se le indica cuales bits de A y B trabaja cada instanciacion
  Sumador sumador1(.A(A[3:0]), .B(B[3:0]), .Cin(Cin), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q1), .RCO(Carry1));
  Sumador sumador2(.A(A[7:4]), .B(B[7:4]), .Cin(Carry1), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q2), .RCO(Carry2));

  //Se asigna a la salida el resultado
  always @ (posedge CLK) begin
    RCO=Carry2;
    Q = {Q2, Q1};//Se vuelven a unir los dos Q calculados
  end
endmodule
```

Figura 4: Módulo Sumador de 8 bits.

Para el módulo sumador de 8 bits, se tiene en un tercer nivel de jerarquía, ya que este instancia dos veces al módulo de 4 bits, con esto se logra la operación de 8 bits, con un primer módulo de 4 bits se operan los primeros 4 bits y la salida de acarreo de este se asigna como entrada al siguiente módulo que opera los últimos 4 bits. Teniendo al final un solo acarreo de importancia que es el que sale del segundo módulo instanciado y se tienen dos salidas Q (Una de cada instanciación, Q1 y Q2), estas se unen en una sola salida Q para obtener el resultado de 8 bits completo.

```
module Sumador32(A,B,Cin,CLK,ENB,MODO,Q,RCO);

  input [31:0] A,B;//Entradas de los nibbles a operar
  input Cin,CLK,ENB;//Entrada del acarreo inicial, el reloj y enable
  input [1:0] MODO;//Seleccion del modo de operacion
  output reg [31:0] Q;//Salida de la operacion
  output reg RCO;//Acarreo de salida

  //Cables necesarios
  wire [3:0] Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8;
  wire Carry1,Carry2,Carry3,Carry4,Carry5,Carry6,Carry7,Carry8;

  //instanciacion del modulo de suma de 4 bits 8 veces para operar el total de bits
  //La salida de acarreo del primero se ingresa en el segundo
  //Se le indica cuales bits de A y B trabaja cada instanciacion
  Sumador sumador1(.A(A[3:0]), .B(B[3:0]), .Cin(Cin), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q1), .RCO(Carry1));
  Sumador sumador2(.A(A[7:4]), .B(B[7:4]), .Cin(Carry1), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q2), .RCO(Carry2));
  Sumador sumador3(.A(A[11:8]), .B(B[11:8]), .Cin(Carry2), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q3), .RCO(Carry3));
  Sumador sumador4(.A(A[15:12]), .B(B[15:12]), .Cin(Carry3), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q4), .RCO(Carry4));
  Sumador sumador5(.A(A[19:16]), .B(B[19:16]), .Cin(Carry4), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q5), .RCO(Carry5));
  Sumador sumador6(.A(A[23:20]), .B(B[23:20]), .Cin(Carry5), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q6), .RCO(Carry6));
  Sumador sumador7(.A(A[27:24]), .B(B[27:24]), .Cin(Carry6), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q7), .RCO(Carry7));
  Sumador sumador8(.A(A[31:28]), .B(B[31:28]), .Cin(Carry7), .CLK(CLK), .ENB(ENB), .MODO(MODO), .Q(Q8), .RCO(Carry8));

  //Se asigna a la salida el resultado
  always @ (posedge CLK) begin
    RCO=Carry8;
    Q = {Q8, Q7, Q6, Q5, Q4, Q3, Q2, Q1};//Se vuelven a unir los dos Q calculados
  end
endmodule
```

Figura 5: Módulo Sumador de 32 bits.

Para el módulo sumador de 32 bits, se tiene en un tercer nivel de jerarquía, ya que este instancia ocho veces al módulo de 4 bits, con esto se logra la operación de 32 bits, con un primer módulo de 4 bits se operan los primeros 4 bits y la salida de acarreo de este se asigna como entrada al siguiente módulo que opera los siguientes 4 bits, así sucesivamente hasta completar los 32 bits. Teniendo al final un solo acarreo de importancia que es el que sale del ultimo módulo

instanciado y se tienen ocho salidas Q (Una de cada instanciación), estas se unen en una sola salida Q para obtener el resultado de 32 bits completo.

3. Plan de Pruebas

En esta ocasión este fue dado en el enunciado de la tarea.

- Prueba 1, suma de 4 bits. El sumador inicia en cero y se definen valores para las entradas A y B. Se envía un flanco activo de reloj y el sumador realiza la operación correspondiente. Se debe repetir este proceso para varias posibles sumas, incluyendo casos que generen acarreo, para verificar el correcto funcionamiento de la señal RCO. Un ejemplo de cómo se puede estructurar la prueba se muestra en la siguiente secuencia:
  - Establecer MODO[1:0]=11 para limpiar el contador.
  - Poner ENB=1.
  - Enviar flanco activo en CLK. Con esto se pone el contador a cero.
  - Establecer MODO[1:0]=01. Pone modo de suma.
  - Enviar flanco activo en CLK. El estado de contador debería pasar a Q=A+B.
  - Variar los valores de A y B para observar distintos resultados, incluyendo casos con acarreo donde RCO=1.
- Prueba 2, resta de 4 bits. El sumador inicia en cero y se definen valores para las entradas A y B para ejecutar la resta correspondiente. También se debe verificar que la señal RCO se pone en 1 cuando se cumple la condición de rebase, ¿en qué estado debería ponerse RCO cuando A es menor que B?
- Prueba 3, mantener el valor en modo 00. El sumador realiza una operación inicial y se modifica el modo para ponerlo en 00. Luego se envían varios flancos positivos del reloj y se verifica que la salida se mantiene con el resultado de la última operación realizada.
- Prueba 4, mantener el valor cuando ENB = 0. El sumador realiza una operación inicial y después de completada se pone ENB=0. Luego se envían varios flancos positivos del reloj y se verifica que la salida se mantiene con el resultado de la última operación realizada.
- Prueba 5, limpiar el contador. Verificar que si el contador se encuentra en cualquier estado inicial posible, es posible regresar su salida a cero poniendo la señal de MODO=2'b11 y sosteniéndola durante un ciclo de reloj.
- Prueba 6, sumador de 8 bits. Construya un sumador de 8 bits utilizando dos contadores de 4 bits de los ya probados. Verifique que el sumador de 8 bits funciona en todos los modos de funcionamiento del sumador de 4 bits. Para esto diseñe una prueba mínima dado que las pruebas de la 1 a la 5 ya cubren una buena porción de la funcionalidad esperada.
- Prueba 7, sumador de 32 bits. Construya un sumador de 32 bits utilizando ocho contadores de 4 bits de los ya probados o ien 4 contadores de 8 bits. Verifique que el sumador de 32 bits funciona en todos los modos de funcionamiento del sumador de 4 bits. Para esto diseñe una prueba mínima dado que las pruebas de la 1 a la 5 ya cubren una buena porción de la funcionalidad esperada.

4. Instrucciones de utilización de la simulación

Uno de los principales aspectos del proyecto, reside en la generación de resultados visuales para generar un mayor entendimiento de los diseños efectuados. A continuación, se detallan los pasos para poder verificar el funcionamiento del código.

- Si se desea utilizar EDAplayground, basta con cargar los archivos *testbench.sv* en el sitio respectivo para este (Izquierdo) y *desing.sv,probador.v* en el sitio respectivo para este (Derecho).
- Una vez verificado lo anterior, se procede a utilizar *Icarus Verilog*, que es una implementación del compilador de lenguaje de descripción de hardware Verilog que genera netlists en el formato deseado (EDIF) [2]. En la sección de Tools and Simulators de EDAplayground se debe seleccionar *Icarus Verilog 0.9.7* y compilar(Botón Run)

5. Resultados

- **Prueba 1**: Se puede notar al inicio no se tienen cargados los valores de A y B, por esto se muestra x, después de realizar una limpieza (MODO=11) se ponen 0, se ingresan A y B y se ejecuta una suma (MODO=01), la cual da el resultado correcto, luego se cambia el acarreo obteniendo el resultado esperado.:

```
Prueba #1, suma de 4 bits.
Operando A: xxxx
Operando B: xxxx
Acarreo de entrada: x
ENB = 1
Resultado de la operacion 11: 0000
Rebase: 0

Operando A: 1110
Operando B: 1001
Acarreo de entrada: 0
ENB = 1
Resultado de la operacion 01: 0111
Rebase: 1

Operando A: 0110
Operando B: 1001
Acarreo de entrada: 1
ENB = 1
Resultado de la operacion 01: 0000
Rebase: 1
```

Figura 6: Prueba 1. Resultados en terminal.

- **Prueba 2**: Se inicia con realizar una limpieza, se ponen 0 los resultados, se ingresan A y B y se ejecuta una resta, la cual da el resultado correcto.

```
Prueba #2, resta de 4 bits.
Operando A: 0110
Operando B: 1001
Acarreo de entrada: 1
ENB = 1
Resultado de la operacion 11: 0000
Rebase: 0

Operando A: 1110
Operando B: 1001
Acarreo de entrada: 0
ENB = 1
Resultado de la operacion 10: 0101
Rebase: 0
```

Figura 7: Prueba 2. Resultados en terminal.

- **Prueba 3**: Se inicia con realizar una suma, se cambia el modo a 00 (mantener el valor anterior), la cual da el resultado correcto.

```
Prueba #3, mantener el valor en modo 00.
Operando A: 1110
Operando B: 1001
Acarreo de entrada: 0
ENB = 1
Resultado de la operacion 01: 0111
Rebase: 1

Operando A: 1110
Operando B: 1001
Acarreo de entrada: 0
ENB = 1
Resultado de la operacion 00: 0111
Rebase: 1
```

Figura 8: Prueba 3. Resultados en terminal.

- **Prueba 4**: Se inicia con realizar una suma, se cambia el ENB a 0 (Se debe mantener el valor anterior), la cual da el resultado correcto.

```
Prueba #4, mantener el valor cuando ENB = 0.
Operando A: 1110
Operando B: 1001
Acarreo de entrada: 0
ENB = 1
Resultado de la operacion 01: 0111
Rebase: 1

Operando A: 1110
Operando B: 1001
Acarreo de entrada: 0
ENB = 0
Resultado de la operacion 01: 0111
Rebase: 1
```

Figura 9: Prueba 4. Resultados en terminal.

- **Prueba 6**: En este caso se prueban todos los modos para A y B de 8 bits, obteniendo los resultados esperados.:

```
Prueba #6, Pruebas para el modulo sumador de 8 bits
Operando A: 11100000
Operando B: 10010000
Acarreo de entrada: 0
Resultado de la memoria: xxxxxxxx
Rebase: x
Resultado de la suma: 01110000
Rebase: 1
Resultado de la resta: 01010000
Rebase: 0
Resultado de la memoria: 01010000
Rebase: 0
Resultado del limpiar: 00000000
Rebase: 0
Resultado de la memoria: 00000000
Rebase: 0
```

Figura 10: Prueba 6. Resultados en terminal.

- **Prueba 7**: En este caso se prueban todos los modos para A y B de 32 bits, obteniendo los resultados esperados.

```
Prueba #7, Pruebas para el modulo sumador de 32 bits
Operando A: 11100000111000001110000011100000
Operando B: 10010000100100001001000010010000
Acarreo de entrada: 0
Resultado de la memoria: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Rebase: x
Resultado de la suma: 01110001011100010111000101110000
Rebase: 1
Resultado de la resta: 01010000101000001010000001010000
Rebase: 0
Resultado de la memoria: 01010000101000001010000001010000
Rebase: 0
Resultado del limpiar: 00000000000000000000000000000000
Rebase: 0
Resultado de la memoria: 00000000000000000000000000000000
Rebase: 0
```

Figura 11: Prueba 7. Resultados en terminal.

6. Conclusiones

- Se logró obtener los resultados esperados para los distintos modos de operación, estos fueron comprobados con impresiones en terminal.
- Por medio de la unidad de pruebas del bloque, se puede apreciar la gran cantidad de resultados que este puede generar debido a la gran variedad de entradas que este puede recibir. Esto al poder operar números de 4,8 y 32 bits.
- Con modularización de la suma y resta se logra tener un código más ordenado. Incluso la subdivisión del problema ayuda con el entendimiento de este y a la hora de ejecutar.
- Se tuvieron problemas con respecto a los cambios de modos, esto debido a que se dificultó el entendimiento de este, por lo que se optó por ver este cambio como un multiplexor que pone en la salida el resultado pedido según el modo.
- Se ralentizó el proceso de creación del banco de pruebas, esto debido a que no se lograba entender como probar varios módulos a la vez y principalmente porque estos tienen los mismos nombres en las variables, por lo que este código se podría mejorar cambiando el nombre de las variables de los módulos de 8 y 32 bits, para así tener un orden claro.

- Para el módulo de 32 bits, se podría haber utilizado 4 instanciaciones del módulo de 8 bits para reducir unas cuantas líneas de código y agilizar la ejecución.

Referencias

[1] J. A. G. Garza, *Sistemas digitales y electrónica digital, prácticas de laboratorio.* Pearson Educación, 2006.

[2] J. Chávez, "Manual de verilog," 1999.