Proyecto 1 Cálculadora Tomógrafo

Kevin Josué Ruiz Rodríguez
email: krr248@estudiantec.cr
Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—This paper is about the process of designing and implementing a circuit that receives a number between one four and adds it to an accumuated number in a register which is updated at the press of a button, this number is shown in a seven segment display, and in given intervals the number activates a different circuit to do something.**

***Palabras clave*—Circuitos digitales, lógica combinacional, desacople.**

I. Introducción

Los circuitos digitales son una parte importante de los avances tecnológicos. Estos funcionan con señales binarias, representando los datos como series de ceros y unos. Esto permite la creación de sistemas que pueden realizar varias tareas, como operaciones aritmeticas o lógicas hasta almacenamiento de datos.

Uno de los aspectos del diseño de circuitos digitales es la lógica combinacional, en este tipo de lógica la salida siempre es obtenida en función de únicamenta las entradas actuales. Esto a diferencia de la lógica secuencial, donde la salida depende tanto de las entradas actuales como de los estados pasados, por lo tanto necesitan memoria para operar, cosa que no es necesaria en los circuitos combinacionales. Esto hace a la lógica combinacional una pieza importante para desarrollar sistemas más complejos.

Para tener un buen desempeño y eficiencia en los sistemas digitales, es importante un buen diseño y optimización de los circuitos combinacionales. Para esto existen técnicas, como el álgebra Booleana y los mapas de Karnaugh, también llamados mapas K, estas son utilizadas para simplificar las expresiones logicas, de manera que se puedan reducir las compuertas necesarias para su implementación, minimizando el consumo de potencia y el tiempo de respuesta.

Con el fin profundizar los conceptos mencionados anteriormente, a lo largo de este documento se decsribe el proceso de diseño e implementación de un circuito que, mediante fotosensores, identifique una cantidad para sumar a un acumulado guardado en un registro, además este resultado es mostrado en un display de 7 segmentos. Adicionalmente se desea que en un intervalo del contador, se active un motor. Este proceso de desarrollo parte desde la creación de tablas de verdad, diseño del esquemático, hasta la implementación en protoboard del circuito.

II. Circuito Diseñado

Para empezar el diseño, se parte de una versión más simple en la que simplemente se utilizan switches para representar los dedos, en este proyecto se toma en consideración que la entrada siempre va a ser al menos una entrada, además se desea tomar en cuenta hasta un máximo de 4, por lo que se utilizan 4 switches para interpretar cada fotosensor, dando como resultado un 1 si se detecta una entrada, y un 0 si no.

A. Codificador de Entrada

Se realiza una tabla de verdad para interpretar las entradas, en la cual se le asigna un valor a cada una de las combinaciones posibles, recordando que se espera al menos un dedo y un máximo de 4, por lo que se tienen únicamente 4 combinaciones.

TABLE I
Tabla de Verdad para las entradas.

A	B	C	D	X_1	X_0
1	0	0	0	0	1
1	1	0	0	1	0
1	1	1	0	1	1
1	1	1	1	0	0

Teniendo estos valores asignados, se pueden encontrar las siguientes ecuaciones booleanas para cada dígito utilizando los mintérminos en la tabla I para encontrar la suma de productos.

$$X_1 = AB\overline{C}\,\overline{D} + ABC\overline{D} \tag{1}$$

$$X_0 = A\overline{B}\,\overline{C}\,\overline{D} + ABC\overline{D} \tag{2}$$

Las ecuaciones 1 y 2 se pueden simplificar utilizando álgebra booleana para facilitar la implementación del circuito.

$$X_1 = AB\overline{D} \tag{3}$$

$$X_0 = A\overline{D}\overline{(B \bigoplus C)} \tag{4}$$

Con las ecuaciones 4 y 3 se puede diseñar el siguiente codificador de la entrada.

B. Codificador de Suma

Una vez obtenida una forma de interpretar las entradas, se procede a diseñar un codificador que va a realizar una suma según estas entradas y un número acumulado, se van a utilizar números de 2 bits tanto para el acumulado, como para el resultado. Sabiendo esto, se puede realizar una tabla de verdad para el codificador, utilizando los dígitos de ambos números como entrada y los dígitos del resultado como salida.

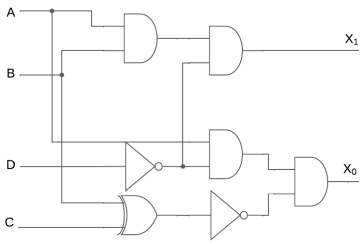

Fig. 1. Codificador de la entrada.

TABLE II
TABLA DE VERDAD PARA EL CODIFICADOR DE SUMA.

X_1	X_0	Y_1	Y_0	S_1	S_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	1	0

La tabla II es un poco compleja, por lo que para analizarla de manera más sencilla se puede realizar un mapa K para cada dígito del resultado.

Analizando el mapa K de la tabla III, se puede obtener la siguiente ecuación booleana para el primer dígito del resultado.

$$Y_0' = \overline{X_0}Y_0 + \overline{Y_0}X_0 \tag{5}$$

De igual manera con la tabla IV se puede obtener una ecuación

TABLE III
MAPA K PARA EL PRIMER DÍGITO DE LA SUMA.

$X_1X_0\backslash Y_1Y_0$	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

TABLE IV
MAPA K PARA EL SEGUNDO DÍGITO DE LA SUMA.

$X_1X_0\backslash Y_1Y_0$	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

para el segundo dígito del resultado

$$Y_1' = X_1\overline{Y_1Y_0} + X_1\overline{X_0Y_1} + \overline{X_1}X_0\overline{Y_1}Y_0 + X_1X_0Y_1Y_0 + X_1Y_1\overline{Y_0} \tag{6}$$

Se puede observar que la ecuación 5 es equivalente a la operación XOR, mientras que la ecacuión 6 se puede simplificar mediante álgebra booleana obteniendo unas ecuaciones simplificadas para cada dígito del resultado.

$$Y_0' = X_0 \oplus Y_0 \tag{7}$$

$$Y_0' = (X_1 \oplus Y_1) \oplus X_0Y_0 \tag{8}$$

Nuevamente, utilizando las ecuaciones 7 y 8, se obtiene el siguiente diseño para el codificador de la suma.

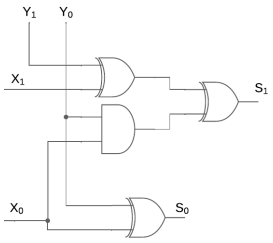

Fig. 2. Codificador de la suma.

El resultado de la suma es actualizado en el registro del acumulador, además de ser enviado a un BCD para obtner la representación necesaria para el display de 7 segmentos, y enviarlo a este mismo.

C. Fase de Desacople

Adicionalmente, se requiere seleccionar un intervalo que active un motor, debido a la cantidad de bits utilizados, se pueden representar 4 números, por que se selecciona únicamente un dígito para activar el motor. En este caso se selecciona el número 3 para la activación, de manera que se pueda utilizar una compuerta AND con cada dígito del acumulador, esta compuerta se conecta a la base de un transistor BJT, el emisor de este transitor se conecta a tierra, mientras que al colector se conecta el motor, con su propia alimentación.

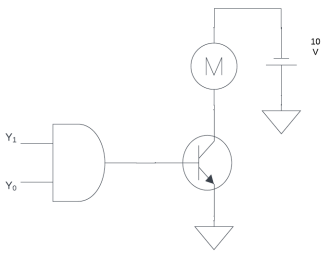

Fig. 3. Etapa de desacople.

Una vez diseñadas las diferentes etapas del sistema se puede proceder a realizar la implementación en protoboard.

III. Resultados

A. Codificador de Entrada

Para comprobar la efectividad del diseño del de codificador de la entrana se implementó el diseño en Sytem Verilog, utilizando las ecuaciones 4 y 3 A esta implementación de la

```
module inDecoder (input logic [3: 0] in,
                  output logic [1: 0] x);
                   //a          //d        //b      //c
   assign x[0] = in[3] & ~in[0] & ~(in[2] ^ in[1]);
                   //a          //b        //d
   assign x[1] = in[3] & in[2] & ~in[0];

endmodule
```

Fig. 4. Implemtación del codificador de entrada en SystemVerilog.

figura 4 se le realizó el siguiente testbench para confirmar el funcionamiento correcto. El resultado del testbench de la

```
module inDecoder_tb;

   logic [3: 0] in;
   logic [1: 0] x;

   inDecoder indec (in, x);

   initial begin
      in = 4'b1000; #10;
      assert (x == 2'b01) else $error("failed 1");
      in = 4'b1100; #10;
      assert (x == 2'b10) else $error("failed 2");
      in = 4'b1110; #10;
      assert (x == 2'b11) else $error("failed 3");
      in = 4'b1111; #10;
      assert (x == 2'b00) else $error("failed 4");
   end
endmodule

```

Fig. 5. Testbench del codificador de entrada en SystemVerilog.

figura 5 se puede observar en la figura 6.

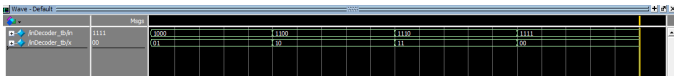

Fig. 6. Resultado del testbench del codificador de entrada en ModelSim.

B. Codificador de Suma

El proceso utilizado para el codificador de entrada, se repite para el codificador de suma, utilizando esta vez las ecuaciones 7 y 8 Nuevamente, a la implementación de 7 se le realiza un

```
module sumDecoder(input logic [1: 0] x, y,
                  output logic [1: 0]s);

   assign s[0] = x[0] ^ y[0];
   assign s[1] = (x[1] ^ y[1]) ^ x[0] & y[0];

endmodule

```

Fig. 7. Implemtación del codificador de suma en SystemVerilog.

testbench para verificar el funcionamiento. El resultado del testbench de la figura 8 se puede observar en la figura 9.

```
module sumDecoder_tb;

   logic [1: 0] x, y, s;

   sumDecoder sumdec (x, y, s);

   initial begin
      y = 2'b00;
      x = 2'b00; #10;
      assert (s === 2'b00) else $error ("failed 0+0");
      x = 2'b01; #10;
      assert (s === 2'b01) else $error ("failed 1+0");
      x = 2'b10; #10;
      assert (s === 2'b10) else $error ("failed 2+0");
      x = 2'b11; #10;
      assert (s === 2'b11) else $error ("failed 3+0");
      y = 2'b01;
      x = 2'b00; #10;
      assert (s === 2'b01) else $error ("failed 0+1");
      x = 2'b01; #10;
      assert (s === 2'b10) else $error ("failed 1+1");
      x = 2'b10; #10;
      assert (s === 2'b11) else $error ("failed 2+1");
      x = 2'b11; #10;
      assert (s === 2'b00) else $error ("failed 3+1");
      y = 2'b10;
      x = 2'b00; #10;
      assert (s === 2'b10) else $error ("failed 0+2");
      x = 2'b01; #10;
      assert (s === 2'b11) else $error ("failed 1+2");
      x = 2'b10; #10;
      assert (s === 2'b00) else $error ("failed 2+2");
      x = 2'b11; #10;
      assert (s === 2'b01) else $error ("failed 3+2");
      y = 2'b11;
      x = 2'b00; #10;
      assert (s === 2'b11) else $error ("failed 0+3");
      x = 2'b01; #10;
      assert (s === 2'b00) else $error ("failed 1+3");
      x = 2'b10; #10;
      assert (s === 2'b01) else $error ("failed 2+3");
      x = 2'b11; #10;
      assert (s === 2'b10) else $error ("failed 3+3");
   end
endmodule

```

Fig. 8. Testbench del codificador de suma en SystemVerilog.

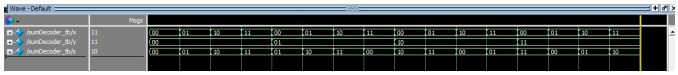

Fig. 9. Resultado del testbench del codificador de suma en ModelSim.

C. Circuito Completo

Finalmente se realiza una implementación del sistema completo, utilizando un registro, con un botón de reinicio y botón de hablitación como entradas, como el de la figura 10. Además de un comparador, que corresponde a la compuerta AND de la figura 3, para enviar la señal que enciende el motor. Utilizando estos módulos, se puede implementar el sistema completo, visto en la figura 12. Al módulo de la figura 12 se le realiza un último testbench, donde se simula que se inicia presionando el botón de reinicio por un tiempo, se presiona y libera constantemente el botón para actualizar el acumulado,

```
module register (input logic rst, en,
                 input logic [1: 0] data,
                 output logic [1: 0] q);

   always_ff @(posedge en or posedge rst)
      if (rst) q = 2'b00;
      else
         if (en) q = data;

endmodule

```

Fig. 10. Módulo de registro en SystemVerilog.

```
module comparator (input [1: 0] a,
                   output equal);

    assign equal = (a[0] & a[1]);

endmodule
```

Fig. 11. Módulo de comparador en SystemVerilog.

```
module main (input logic rst, en,
             input logic [3: 0] in,
             output logic motor_on,
             output logic [1: 0] acc);

    logic [1: 0] x, res;

    inDecoder indec (in, x);

    sumDecoder sum (x, acc, res);

    register res_reg (rst, en, res, acc);

    comparator cmp (acc, motor_on);

endmodule
```

Fig. 12. Implementación del circuito completo en SystemVerilog.

y se mantiene un número de entradas distinto, cada una por cierto tiempo. Finalmente se verifica el comportamiento del

```
module main_tb;

    logic rst, en;
    logic [3: 0] in;
    logic motor_on;
    logic [1: 0] acc;

    main testing (rst, en,
                  in, motor_on, acc);

    always #5 en = ~en;

    initial begin
        en = 1'b0;
        rst = 1'b1; #10;
        rst = 1'b0;
        in = 4'b1000; #40;
        in = 4'b1100; #40;
        in = 4'b1110; #40;
        in = 4'b1111; #40;
        rst = 1'b1;
        #20; $stop;
    end
endmodule
```

Fig. 13. Testbench para el circuito completo.

circuito.

En la figura 14 se puede observar que el comportamiento es el esperado, cuando se tiene una entrada y se presiona el botón de habilitar, el acumulado aumenta en 1, y así respectivamente según el número de entradas, además siempre que el acumulado tiene un valor de 3, se activa la señal para encender el motor.

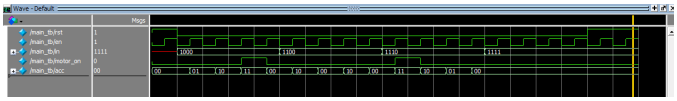

Fig. 14. Resultado del testbench del circuito completo.

A la hora de hacer la implementación en protoboard del diseño propuesto, no se obtuvo ningún tipo de respuesta en los componentes, por lo tanto no se pudo realizar la implementación de manera éxitosa.

IV. Conclusiones

En este proyecto se propuso una solución válida para una cálculadora tomógrafo, ya que según las simulaciones y pruebas realizadas se tenía el comportamiento esperado en el diseño, No obstante, a la hora de realizar la implementación en protoboar, no se pudo verificar el comportamiento, esto pudo haber sido causado debido a errores en los componentes, o en las conexiones realizadas, ya que se trata de un diseño muy complejo con muchas conexiones, por lo que es propenso a errores en la implementación. Aun así, con el desarrollo se logró profundizar los conceptos de circuitos digitales, lógica combinacional y secuencial, además de aplicar técnicas para el diseño como álgebra Booleana y mapas K.

References

[1] Harris, S., & Harris, D. (2015). *Digital design and computer architecture: arm edition*. Morgan Kaufmann.