INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Trabajo Práctico Nº 1

Electrónica III - 2019

Grupo 1:

Farall, Facundo
Gaytan, Joaquín
Kammann, Lucas
Maselli, Carlos

August 18, 2019

1 Ejercicio 2

1.1 Expresión en mintérminos

La subsección presente tratará la siguiente expresión caracterizada como:

$$f(e,d,c,b,a) = \sum m(0,2,4,7,8,10,12,16,18,20,23,24,25,26,27,28)$$

1.1.1 Simplificación con Álgebra booleana

En el siguiente desarrollo primero se escribe de forma completa la suma de productos que se describe en la expresión consignada, donde cada producto se compone de la combinación de entradas en cada estado indicado, de forma tal que el resultado de tal producto sea un estado lógico activo, esto es, un '1' binario. Con el objetivo de proveer la mayor claridad posible en el desarrollo, se subindican y supraindican los términos, en donde el subíndice describe la identificación de un término y luego el supraíndice describe el conjunto de términos de los cuáles deriva.

$$\begin{aligned} f(e,d,c,b,a) = {} & \overline{e}\cdot\overline{d}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + \overline{e}\cdot\overline{d}\cdot\overline{c}\cdot b\cdot\overline{a} + \overline{e}\cdot\overline{d}\cdot c\cdot\overline{b}\cdot\overline{a} + \overline{e}\cdot d\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + \overline{e}\cdot\overline{d}\cdot c\cdot b\cdot a + \overline{e}\cdot d\cdot\overline{c}\cdot b\cdot\overline{a} \\ & + \overline{e}\cdot d\cdot c\cdot\overline{b}\cdot\overline{a} + e\cdot\overline{d}\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} + e\cdot\overline{d}\cdot\overline{c}\cdot b\cdot\overline{a} + e\cdot\overline{d}\cdot c\cdot\overline{b}\cdot\overline{a} + e\cdot\overline{d}\cdot c\cdot b\cdot a + e\cdot d\cdot\overline{c}\cdot\overline{b}\cdot\overline{a} \\ & + e\cdot d\cdot\overline{c}\cdot\overline{b}\cdot a + e\cdot d\cdot\overline{c}\cdot b\cdot\overline{a} + e\cdot d\cdot\overline{c}\cdot b\cdot a + e\cdot d\cdot c\cdot\overline{b}\cdot\overline{a} \end{aligned}$$

1.1.2 Simplificación con mapas de Karnaugh

1.1.3 Expresión desarrollada usando NOR

1.1.4 Circuitos lógicos

1.2 Expresión en maxtérminos

1.2.1 Simplificación con Álgebra booleana

1.2.2 Simplificación con mapas de Karnaugh

1.2.3 Expresión desarrollada usando NOR

1.2.4 Circuitos lógicos

2 Ejercicio 3

De los dos módulos pedidos para implementar en Verilog se tomó la decisión, de forma completamente arbitraria, de realizar uno mediante las compuertas lógicas que lo componen, y otro a través de la descripción de su comportamiento (behavioural). La razón detrás de esta decisión fue pura y exclusivamente para hacer uso de las variantes provistas por Verilog, e interiorizarnos en su estilo de programación.

2.1 Decoder de 4 entradas

Recibe una entrada de 2 bits, que determinan cual de las cuatro salidas accionar. Su implementación se realizó a través de lógica de compuertas, donde únicamente la combinación correcta de los dos bits de entrada, ponen a la salida con ese número en 1 lógico. Las relaciones lógicas son relativamente sencillas y *straight-forward*, por lo cual no se considera necesario demostrarlas mediante una tabla de verdad o mapa de Karnaugh. A continuación se presenta el código:

```
module decoder4out(coded, y0, y1, y2, y3);
    input [1:0] coded;
    output y0, y1, y2, y3;

    assign y0 = ~coded[1] & ~coded[0];
    assign y1 = ~coded[1] & coded[0];
    assign y2 = coded[1] & ~coded[0];
    assign y3 = coded[1] & coded[0];

endmodule
```

2.2 MUX DE 4 ENTRADAS

Recibe una entrada de 4 bits con las 4 "fuentes", otra entrada de 2 bits que hace las veces de selector, y cuenta con una salida de 1 bit. La salida copiará el valor de la entrada determinada por el selector. El módulo es logró mediante una descripción del comportamiento del mismo, en el cual se le especificó qué debía realizar ante cambios en alguna de sus entradas. A continuación se presenta el código:

```
module mux4in (x, sel, y);
    input [3:0] x;
    input [1:0] sel;                    // sel selects the exit (x[3], x[2], x[1], x[0]).
    output reg y;

    always @(sel or x) begin
        if (sel == 0)
            assign y = x[0];
        else if (sel == 1)
            assign y = x[1];
        else if (sel == 2)
            assign y = x[2];
        else if (sel == 3)
            assign y = x[3];

    end

endmodule
```

2.3 TEST-BENCH

Se sometió a los dos módulos a un testeo de su respuesta a cada una de las posibles entradas, y los resultados fueron los esperados. Los mismos pueden ser replicados ejecutando los comandos:

```
user@computer: path/to/EJ_3/folder$ make
user@computer: path/to/EJ_3/folder$ ./run
```