Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE-0323 Circuitos Digitales I
Grupo 04

Profesor:
Esteban Badilla Alvarado

Proyecto final

Estudiantes:
Marianne Escalante Sánchez - B72710
Steven Mora Barboza - B95109
Kenny Wu Wen - C08592

Sede Rodrigo Facio
II Ciclo-2021

Primera parte

1. Investigue cuáles son las diferencias entre escribir los diseños en verilog de manera estructural y de manera conductual

Según Clavijo, P,R. (2012), menciona que en la descripción conductual se permitirá el uso de estructuras de control, su descripción será algorítmica, igual que el software, facilitará la creación de funciones complejas y se basará en la sentencia always. Entonces se puede entender que la descripción conductual se ocupa de la lógica o el comportamiento de un sistema y maneja la implementación de lógica compleja. (p.10)
Por otro lado, la descripción estructural, será donde se conectarán módulos que se definieron con anterioridad, en donde sus puertas lógicas ya se encuentran predefinidas en Verilog. Esto quiere decir que, este tipo utiliza elementos de la biblioteca o elementos creados previamente, y están conectados entre sí. Sería similar a un esquema de captura en el que la función del diseñador es inicializar los bloques y vincularlos.(Chávez, J., 1999, p.3)
Con la información anterior, se entiende que el diseño conductual se utiliza para describir la función de manera algorítmica, y es una descripción de diseño que usa bloques always, en cambio el diseño estructural, describe un diseño que utiliza componentes básicos (particularmente para bloques de construcción de nivel inferior, como puertas AND).

2. Realice la implementación de un Flip Flop tipo D con entrada de enable activa en bajo, de manera conductual.

Flip Flop tipo D enable activa en bajo:

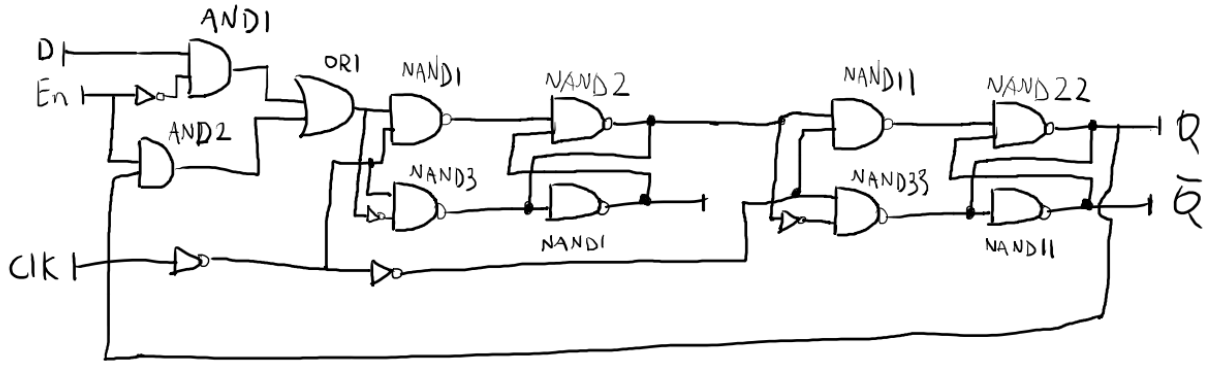
```
module BasicDFF_enable(
    input wire clk, en,
    input d,
    output reg q //No se ocupa q negada pero si se requiere solo lo agregamos y sera ~q
);
always @(posedge clk) //Cada flanco creciente activarse
    begin
        if (en == 0) q <= d;//Si en esta en bajo, se activa el flip flop
        else q=q; //De lo contrario no cambios
    end
endmodule
```

Cómo sabemos, los wires son un tipo de net, los cuales cablean el diseño cómo su nombre lo diría en español, (cables), luego, los integer al igual que los reg son de tipo variable. Es importante recordar que de manera predeterminada, los datos de

tipo reg y wire son números que no poseen signo, caso contrario del integer que si es un número con signo.
Seguidamente tenemos el bloque always, que se encarga de ejecutar todas las declaraciones de manera secuencial y dentro ponemos la condición if donde va a ejecutar la línea q=d solo cuando enable sea cero y ocurra un flanco creciente en el reloj.

3. Realice la implementación de un Flip Flop tipo D con entrada de enable activa en bajo, de manera estructural y de flanco creciente.

Primero se realiza el esquemático respectivo, para de esta manera poder escribir el código de manera más sencilla.




```
//FF tipo D con enable activo en bajo
module DFF_enable_negado(
    input wire clk, en,
    input d,
    output wire q,q_n //Q y Q NEGADA
);
//Se baso en el diagrama para crear las conexiones entre cada compuerta

wire not1_out, not2_out, nand1_out, nand2_out, nand3_out, nand4_out;
wire and1_out, and2_out, or1_out;

wire not3_out, not4_out, nand11_out, nand33_out;

not NOT3(not3_out, clk); //clock negado
//Enable
not NOT1(not1_out, en);
and AND1(and1_out, d, not1_out);
and AND2(and2_out, en, q);
or OR1(or1_out, and1_out, and2_out);
//Primer latch
not NOT2(not2_out, or1_out);
nand NAND1(nand1_out, or1_out, not3_out);
nand NAND3(nand3_out, not2_out, not3_out);
nand NAND2(nand2_out, nand1_out, nand4_out);
nand NAND4(nand4_out, nand3_out, nand2_out);
//Segundo latch
not NOT44(not4_out, nand2_out);
nand NAND11(nand11_out, nand2_out, clk);
nand NAND33(nand33_out, not4_out, clk);
nand NAND22(q, nand11_out, q_n);
nand NAND44(q_n, nand33_out, q);
```

```
//Fin
endmodule
```

Se hace la definición del módulo, en este caso se le denomina DFF_enable_negado. Seguidamente, se especifican los nombres de los cables que se utilizaron para transmitir los valores lógicos. Luego de la conexión de cada cable que se establecieron en el esquemático, es decir, ir describiendo las conexiones de las compuertas.

4. Implemente un banco de pruebas capaz de verificar que las salidas de los diseños realizados en los dos ítems anteriores son correctas e idénticas.
TEST BENCH para el FFD enable activo en bajo.

```
`include "FFD_enable.v"
`include "FFD_enable_estruc.v"
`timescale 1ns/1ns

module FFD_enable_tb;
    reg clk, en;
    reg d;
    wire q,q_n;

always begin
        clk = 1'b1; #5; // 5ns en alto
        clk = 1'b0; #2; // 2ns en bajo
end
always begin
        en = 1'b1; #20; // 20ns en alto
        en = 1'b0; #20; // 20ns en bajo
end

//Creamos el Flip Flop y lo conectamos con el cascaron
BasicDFF_enable FFD_conduc (.clk(clk),
    .en(en), .d(d), .q(q));

initial begin
    $dumpfile("prueba1.vcd");//Guardamos los datos simulados en prueba1.vcd
    $dumpvars(1, FFD_enable_tb);
    d=0; #9 d=1; #1 d=0; #1 d=1; #2 d=0; #1 d=1; #12 d=0;
        #1 d=1; #2 d=0; #1 d=1; #1 d=0; #1 d=1; #1 d=0; # 7 d=1;
        #8
    $stop;
    $finish;
end

endmodule
```

Inicialmente se implementó el cascarón del test del flip flop, con las mismas patillas del mismo, los cuales corresponden a CLK, d y q, luego se hace un clock de 5 ns arriba y 2ns abajo para después hacer un enable de señal cuadrada con periodo de 3 s; entonces se crea el flip flop y se conecta al cascarón, luego se inicia en begin, se utiliza el comando $dumpfile para guardar los datos obtenidos simulados, en $dumpvars se ingresa el nombre del cascarón, y luego se simuló con los parámetros escritos en el código.

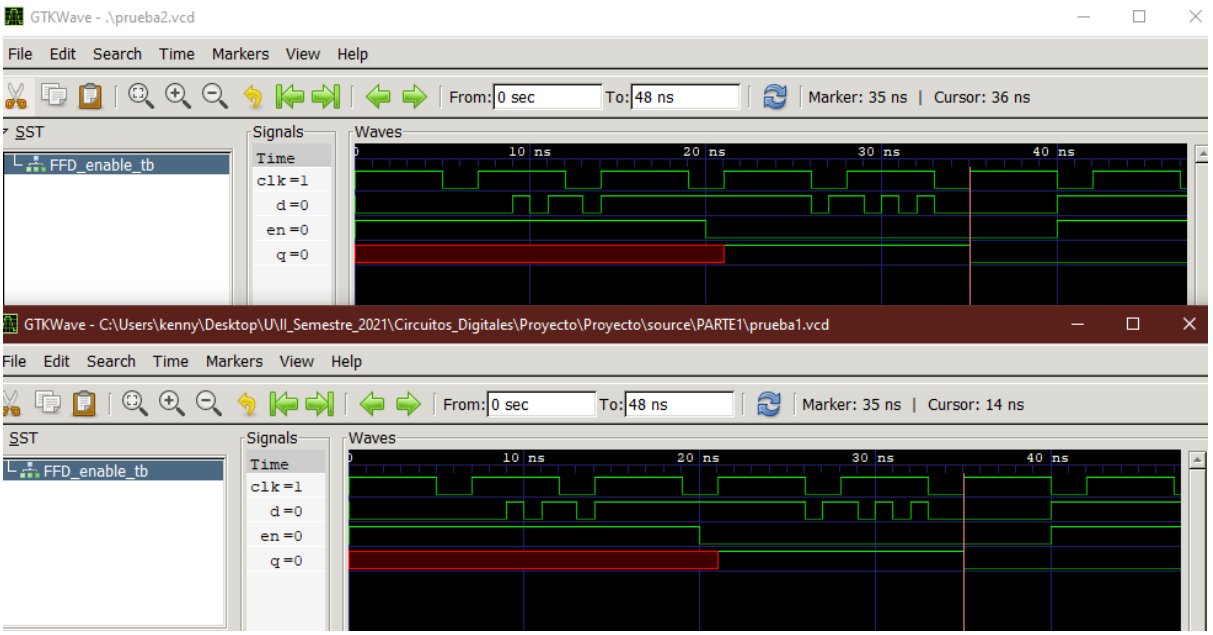
A continuación se muestra el FFP enable activo en bajo de manera estructural:

```
`include "FFD_enable.v"
`include "FFD_enable_estruc.v"
`timescale 1ns/1ns

module FFD_enable_tb;
    reg clk, en;
    reg d;
    wire q,q_n;

always begin
        clk = 1'b1; #5; // 5s en alto
        clk = 1'b0; #2; // 2s en bajo
end
always begin
        en = 1'b1; #20; // 20s en alto
        en = 1'b0; #20; // 20s en bajo
end


DFF_enable_negado FFD (.clk(clk),
    .en(en), .d(d), .q(q));


initial begin
    $dumpfile("prueba2.vcd");//Guardamos los datos simulados en test2.vcd
    $dumpvars(1, FFD_enable_tb);
    d=0; #9 d=1; #1 d=0; #1 d=1; #2 d=0; #1 d=1; #12 d=0;
        #1 d=1; #2 d=0; #1 d=1; #1 d=0; #1 d=1; #1 d=0; # 7 d=1;
        #8
    $stop;
    $finish;
end

endmodule
```

Una vez simulado, se puede observar que la simulación del TEST BENCH para el FFD enable activo en bajo arroja los resultados esperados satisfactoriamente y ambas simulaciones concuerdan en el comportamiento.

```
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\PARTE1> iverilog -o .\prueba1.o .\FFD_enable_tb.v
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\PARTE1> iverilog -o .\prueba2.o .\FFD_enable_estruc_tb.v
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\PARTE1> vvp .\prueba1.o
VCD info: dumpfile prueba1.vcd opened for output.
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 48 ticks.
> cont
** Continue **
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\PARTE1> vvp .\prueba2.o
VCD info: dumpfile prueba2.vcd opened for output.
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 48 ticks.
> cont
** Continue **
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\PARTE1> gtkwave .\prueba1.vcd

GTKWave Analyzer v3.3.71 (w)1999-2016 BSI

[0] start time.
[48] end time.

GTKWave Analyzer v3.3.71 (w)1999-2016 BSI

[0] start time.
[48] end time.
```

Ejecución del código:



Se puede observar que durante los primeros 20ns no hay salida de q, al ser enable en alto empieza a tener respuesta cuando enable está en bajo y cambia en cada flanco creciente dependiendo del valor de d.

Segunda parte

1.La memoria RAM, que sus siglas significa Random Access Memory o en español, memoria de acceso aleatorio. Estos tienen el trabajo de ser la memoria dinámica de la computadora en donde se almacena información de programas que se están ejecutando en tiempo real. Esta memoria RAM se refiere al tipo de almacenamiento

de datos que nos permite leer o escribir los datos almacenados en cualquier orden, o sea, de manera aleatoria y no en secuencia (pila o fila). (Dell, C. R. (s.f.))

2. Creando un banco de pruebas para testear la memoria de 8 bits con los casos de escribir en una línea, leer una línea y leer una línea vacía. Se va a tener un parámetro local llamado period que se usará para la duración de las señales y un ciclo de reloj de 20s en alto y 20s en bajo.

Como observamos en la figura, obtenemos los resultados esperados, cada flanco positivo o creciente del reloj, nuestra memoria lee o escribe dependiendo del valor de wrEnable, en nuestro caso escribe C0FFEE00 en la línea 0 y luego en el siguiente flanco creciente nos lee la línea que se guarda en dataOut de 32 bits. Después, se sigue leyendo las líneas 01 y 120 (78 en hexadecimal), donde la línea 01 contiene la palabra BEBEBEBE y la otra está vacía como se esperaba. Se adjunta el archivo ram_tb.v usado para el test:

```
`include "ram.v"
`timescale 1 ns/1 ns

module ram_tb;
    reg [7:0] addr;
    reg [31:0] dataIn;
    reg wrEnable;
    wire [31:0] dataOut;
    localparam period = 20; //periodo de 20s


    reg clk;
    //clock de 20s alto y bajo
    always begin
        clk = 1'b1; #20;
        clk = 1'b0; #20;
    end
    //conectamos con el cascaron los wires
    ram UUT (
        .clk(clk),
        .addr(addr),
        .dataIn(dataIn),
        .wrEnable(wrEnable),
        .dataOut(dataOut)
    );
```

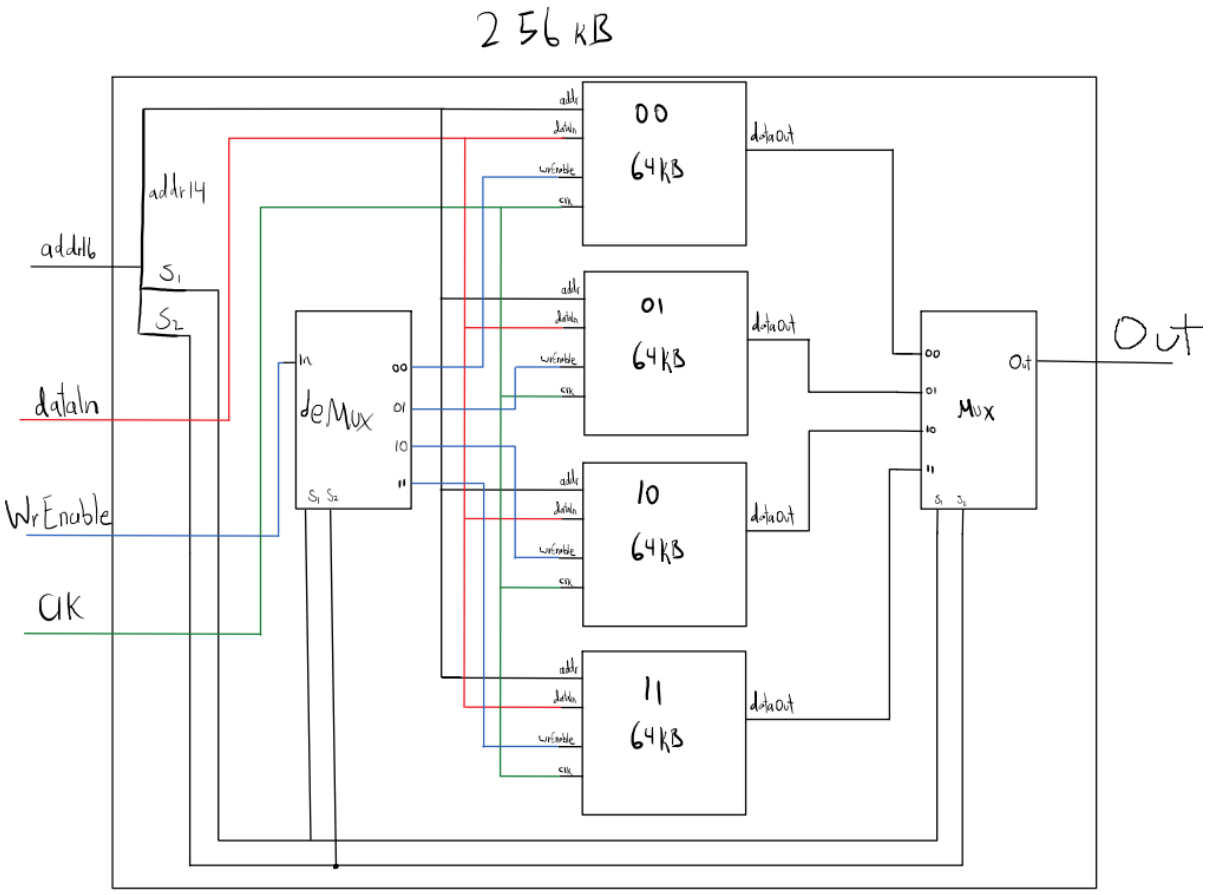
```
initial begin
    $dumpfile("test.vcd");
    $dumpvars(1, ram_tb);
    #period; #period; #period;
    wrEnable = 1;
    dataIn = 32'hC0FFEE00;
    addr = 7'h0;
    #period; #period; #period;
    wrEnable = 0;
    addr = 7'h0;
    #period; #period; #period;
    addr= 7'h1;
    #period; #period; #period;
    addr= 7'h78;
    #period; #period; #period;
    $stop;
    $finish;

end

endmodule
```

3. Para esta parte como la instancia más grande de ram que se podía crear eran de 64 kB, y cada línea era de 32 bits, además de que ocupamos crear la memoria de 256 kB, entonces:

1 línea = 32 bits = 4 bytes
64 kB = 16384 líneas
4*64 kB = 256 kB = 65536 líneas

Quiere decir que necesitamos 65536 líneas para tener un almacenamiento de 256 kB y eso conlleva a utilizar 4 instancias de ram de 64kB y eso son 16384 líneas que aplicando logaritmo de base 2 nos da 14 bits, quiere decir que con 14 bits podemos representar las 16384 líneas en base 2. Por eso, se utilizará un BANDWIDTH de 14 para las rams de 64kB. Luego viene siendo que las 65536 líneas se pueden representar con 16 bits, o sea 2 bits más que los de 64kB, estos 2 bits extra los utilizaremos para los demux y mux con el fin de decidir cuál ram estar leyendo y escribiendo, y, los restantes 14 bits serán enviados como la dirección a la ram.

Se va a seguir el siguiente esquemático para realizar la memoria de 256kB:



Como vemos en la figura, la memoria 256kB tendrá cuatro entradas igual a las rams de 64kB. La entrada addr16 se dividirá en addr14 (14 bits) y los dos bits más significativos para los Demux y Mux. Todos las rams tendrán conectada directamente la entrada de reloj CLK y dataIn, sin embargo, la entrada wrEnable de las rams de 64kB dependerá de la salida del Demux y este de s1 y s2.

Para evitar problemas de escritura, para eso está el Demux con sus salidas conectados a los wrEnable de las rams de 64kB y su entrada como wrEnable de la ram de 256kB. Esta funciona de modo si s=00, la salida 00 del Demux será igual al wrEnable de la ram de 256kB mientras que las otras salidas tendrán un valor de 0, con el fin de que nunca escriban sobre la dirección de memoria en el que está actualmente.

Más adelante, tenemos las salidas dataOut de las rams dentro, que llegan a un mux de 4x1, que funciona semejante de las Demux descritas anteriormente. Este se encarga de que dependiendo de las entradas s1 y s2, solo una salida se pone en la salida del ram de 256kB.

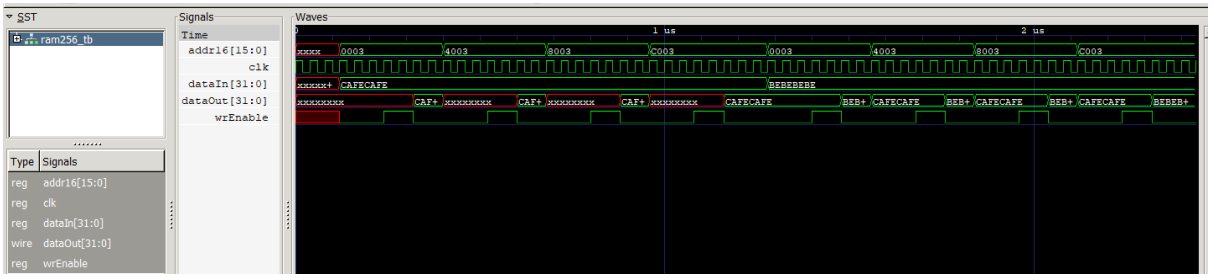
Finalmente se conecta todo el circuito como estaba en la figura y obtenemos el código en Verilog de la ram de 256kB.

Para los diseños de las rams de 64kB se utilizó el código entregado por el profesor, que se comprobó su funcionamiento en el punto 2, con la modificación de que serán de 14 bits.
Para el mux y demux se crearon con bloques always pero también se pudo de forma estructural, sin embargo, se prefirió la forma conductual al ser más fácil de entender.

4. Para el testbench, lo haremos en un intervalo de tiempo de 1ns/1ns, con un ciclo de reloj de 20s alto y 20s en bajo. Se verificará que se están utilizando las 4 rams de 64kB, de manera que sabiendo que cada uno contiene un máximo de 16384 líneas, entonces 14'h3 representa la cuarta final de la ram00, 14'h4003 representa la cuarta fila de la ram01, 14h'8003 la cuarta fila de la ram10 y 14'C003 la de ram11. Se procederá a no tener vacías las memorias de manera que inicialmente ninguna tiene guardado algo por lo que si se leen se espera que muestren valores no importa (XXX…).

```
PS C:\Users\kenny\Desktop\U\II Semestre 2021\Circuitos Digitales\Proyecto\256ram> iverilog -o salida2.o .\256ram_t
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\256ram> vvp .\salida2.o
VCD info: dumpfile test2.vcd opened for output.
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 3560 ticks.
> cont
** Continue **
PS C:\Users\kenny\Desktop\U\II_Semestre_2021\Circuitos_Digitales\Proyecto\256ram> gtkwave .\test2.vcd

GTKWave Analyzer v3.3.71 (w)1999-2016 BSI

[0] start time.
[3560] end time.
WM Destroy
```

Compilamos con iverilog el archivo .v para crear un objeto (.o) y lo corremos con vvp el archivo vvp para finalmente correr el gtkwave con el archivo .vcd creado con el nombre que le asignamos.



Como se puede observar en la figura, cada flanco creciente, se lee y escribe dependiendo del valor de wrEnable y se puede ver claramente los resultados esperados. Se puso que se leyera cada ram de 64kB su linea 4, inicialmente deberían no contener nada y eso es justamente lo que nos muestra la figura, luego el wrEnable se prende y escribe la palabra CAFECAFE en su linea 4 para que cuando vuelva a leerlo aparezca CAFECAFE. Más adelante, se vuelven a leer las

líneas 4 de las rams esperando que aparezca CAFECAFE cuando haya un flanco creciente de reloj y se escribe BEBEBEBE cuando wrEnable este prendido y haya un flanco creciente de reloj para luego leer que aparezca BEBEBEBE en la dirección de memoria.

Bibliografía:

Chávez, J. (1999). Manual de Verilog ver 0.4. Recuperado de: https://marceluda.github.io/rp_dummy/EEOF2018/verilog.pdf

Clavijo Vázqquez, P. R. de, (2012). Introducción a HDL Verilog. Departamento de Tecnología Electrónica. Recuperado de: https://www.dte.us.es/Members/paulino/Verilog-Intro.pdf

Dell, C. R. (s.f.). ¿Qué es la memoria (RAM)? Recuperado de: https://www.dell.com/support/kbdoc/es-cr/000148441/what-is-memory-ram