

Diseño de Maquinas de Estados en Lenguaje Descriptivo de Hardware

Vargas Meza Aldo Alexandro, 213495653

Los parámetros ya definidos en un programa funcionan bien cuando la secuencia de los procesos ya está establecida, y previamente se ha diseñado un “camino” por el cual el flujo de los datos debe moverse. Es importante destacar la diferencia entre un programa que dependa de sus entradas para proporcionar una salida a un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores.

I. INTRODUCCIÓN

Una máquina de estados es una estructura de programa que nos sirve para determinar el comportamiento de algo en base al estado en el que se encuentre. Para cada estado por tanto se tendrá un comportamiento.

Las máquinas de estados se pueden utilizar en muchos aspectos y niveles. Podemos utilizarlas para controlar el estado de la aplicación que estemos realizando.

En este proyecto, se diseñarán dos máquinas de estados, una maquina Mealy, cuya salida depende de sus entradas y del estado actual de programa, y una Moore que solo depende del estado actual para proporciona una salida.

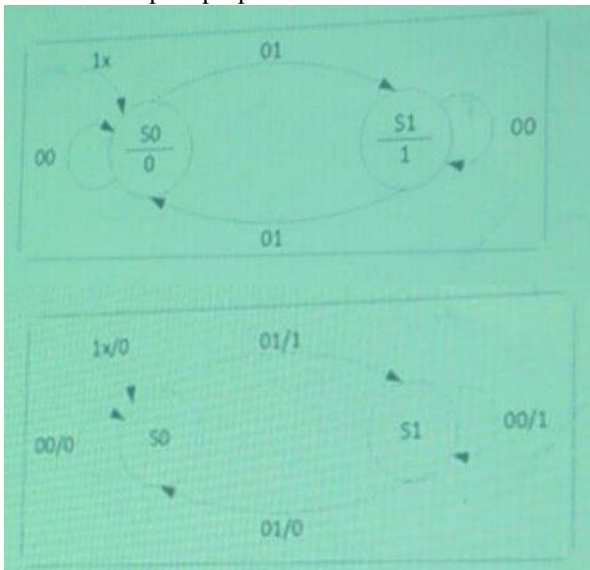


Imagen01 "Diagramas de estados"

II. COMPOSICIÓN DEL CÓDIGO

Ambos módulos tienen una configuración igual: una entrada “a” de 2 bits, un reloj, una entrada para reset, y una salida. De manera interna tiene también un registro para el estado, y uno para una bandera que indicará el momento en que el sistema puede empezar a trabajar, que es cuando la combinación en la entrada sea [1X].

```
module moore(  
    input wire [1:0] a,  
    input wire clk,  
    input wire reset,  
    output reg out  
);  
reg state;  
reg bandera;
```

Imagen 02 "Modulo Maquinas de estados"

Cuando $a \geq 2$, la variable bandera será igual a 1, así como el estado y la salida del sistema, serán igual a 0. Esto se cumplirá también, cuando la variable reset esté en estado alto, solo que la bandera permanecerá en 0 hasta que la combinación de la condicional se cumpla de nuevo.

```
always@(posedge clk or state)  
begin  
    if(a>=2'b10)begin  
        bandera=1;  
        state=0;  
        out=0;  
    end  
    else if(reset)begin  
        bandera=0;  
        state=0;  
        out=0;  
    end  
end
```

Imagen 03 "Proceso Always"

En el proceso case, están todas las condicionales separadas cuando la variable state, que es el estado del sistema esté en 1 y en 0. Las variables toman en cuenta que la variable bandera esté activada, y que la combinación de las entradas sea la correcta para proporcionar la salida y los cambios de estado. El único proceso que se diferencia entre los módulos, es la condicional case, que cambia levemente entre una y otra.

Estos cambios se deben a que los valores de las entradas tienen un cambio diferente entre ambos módulos.

```
case(state)
0:
    if(a==2'b00 && bandera==1)begin
        state=0;
        out=0;
    end

    else if(a==2'b01 && bandera==1)begin
        state=1;
        out=1;
    end

1:
    if(a==2'b00 && bandera==1)begin
        state=1;
        out=1;
    end
    else if(a==2'b01 && bandera==1)begin
        state=0;
        out=0;
    end

end
```

Imagen 04"Condicional CASE Moore"

```
case(state)
0:
    if(a==2'b00 && bandera==1)begin
        out=0;
    end

    else if(a==2'b01 && bandera==1)begin
        out=1;
        state=1;
    end

1:
    if(a==2'b00 && bandera==1)begin
        out=1;
    end
    else if(a==2'b01 && bandera==1)begin
        out=0;
        state=0;
    end

end
```

Imagen 05"Condicional CASE Mealy"

III. TEST BENCH

Para simular el funcionamiento del circuito, se optó por incorporar dos módulos test bench, para estimular las máquinas de estados y ver cómo funcionarían si ya fuera parte de un sistema.

El módulo se compone de las mismas variables del módulo contador, solo que estas son inversas, lo que significa que las entradas del módulo contador, son las mismas pero en las salidas del módulo test bench.

El módulo del test bench se compone de las entradas y la salida del módulo contador, solo que a la inversa para poder monitorear e inyectar valores.

```
module test_bench(
    output reg[1:0]a,
    output reg clk,
    output reg reset,
    input wire out
);

mealy test_mealy(
    .a(a),
    .clk(clk),
    .reset(reset),
    .out(out)
);
```

Imagen 06"Modulo e instanciación"

El CLK fue simulado con un proceso Always, y es una entrada al contador, es importante destacar que el CLK es aquel que marca cuando hay cambios en el sistema, en este caso con un flanco de subida.

```
always
begin
    clk = 0;
    #10;
    clk = 1;
    #10;
end
```

Imagen 07"Proceso CLK"

El comportamiento del circuito se define en el proceso Initial(), que define los tiempos y los cambios entre las variables.

```
initial begin
    reset=1;
    a=00;
    #10;
    reset=0;
    a=10;
    #10;
    a=01;
    #20;
    a=00;
    #20;
    a=01;
    #20;
    a=00;
    #20;
```

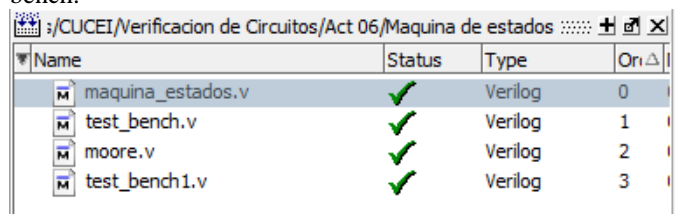
Imagen 08"Parte del Proceso INITIAL"

IV. SIMULACIÓN

Para simular, primero es necesario llevar a cabo la compilación del código y de cada uno de los módulos, así como una compilación completa de cada uno.

El software utiliza un compilador propio, que hace un chequeo de sintaxis y una vez compilados todos los módulos involucrados es posible llevar a cabo la simulación.

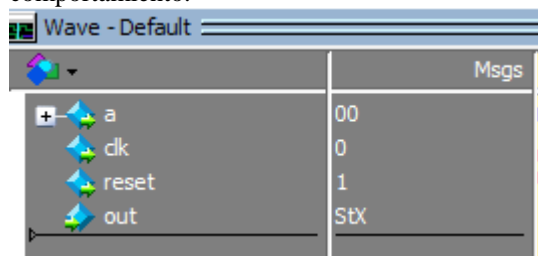
En este diseño, fue necesaria la implementación de dos módulos de las máquinas de estados, y dos módulos de test bench.



Name	Status	Type	On Δ
maquina_estados.v	✓	Verilog	0
test_bench.v	✓	Verilog	1
moore.v	✓	Verilog	2
test_bench1.v	✓	Verilog	3

Imagen 09 "Archivos compilados"

Una vez compilados los archivos, se procede a la simulación de las siguientes variables añadidas a la curva de comportamiento.



Variable	Value
a	00
clk	0
reset	1
out	StX

Imagen 10 "Variables en la curva"

V. COMPROBACIÓN DE DATOS

Una vez simulado, es posible checar que los datos que salen sean los esperados, a nivel bit por bit en ambas máquinas de estados.

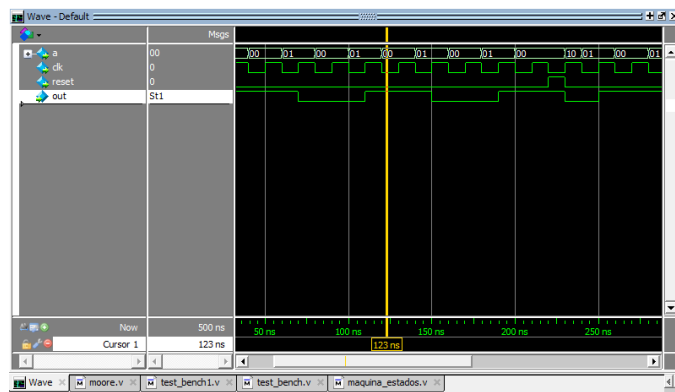
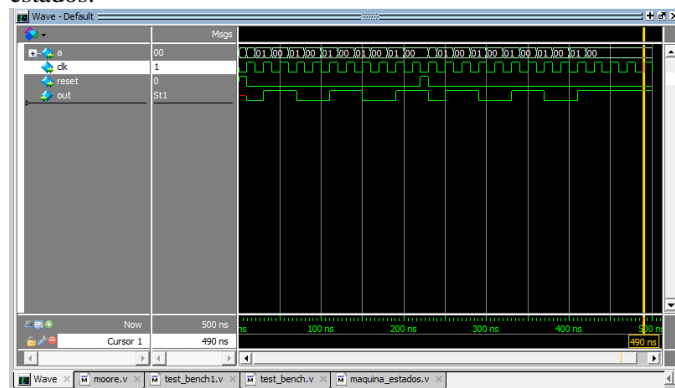


Imagen 11 "Simulación del Test bench"

VI. CONCLUSIÓN

No es de extrañar que muchos desarrolladores profesionales de sistemas embebidos de talla mundial, no solamente usen regularmente este estilo de programación, sino que recomienden su uso en cualquier proyecto. En mi opinión personal, es una buena manera de tener un orden en el proceso, que quizá es más sencillo de lograr con este tipo de lógica.

VII. BIBLIOGRAFÍA

<http://tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/13-introduccion-a-las-maquinas-de-estado-finito1>

WIKIBOOKS. Obtenido de

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog/M%C3%A1quinas_de_estado

UNCL (s.f.). Obtenido de Colorado Education:

http://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/maquinasef.pdf

Verilog Oficial Site. (s.f.). Obtenido de

<http://www.verilog.com/>

Wikibooks. (s.f.). Obtenido de

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog/