INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Ingeniería en Sistemas Computacionales

Diseño de Sistemas Digitales

Practica 2. Máquina de Moore

Presenta:

Álvarez Hernández Gabriel Alexander
García Quiroz Gustavo Iván
Huesca Laureano Josué Alejandro
Muñoz Valdivia Irving Omar
Pedroza Villagómez Emir

FECHA DE ENTREGA: 27/06/24

Índice

Introducción

Una máquina de Moore se refiere a un concepto propuesto por Gordon Moore, cofundador de Intel, en 1965. Gordon Moore observó que la cantidad de transistores en un circuito integrado se duplicaba aproximadamente cada dos años, lo que implicaba un crecimiento exponencial en la capacidad de procesamiento de los microprocesadores y, por extensión, de las computadoras en general.

En el contexto de la Ley de Moore, se hace referencia a dos aspectos principales:

1. Crecimiento exponencial de la capacidad: Esta idea sugiere que la cantidad de transistores (y por lo tanto la capacidad de procesamiento) en los microprocesadores aumenta exponencialmente con el tiempo.

2. Impacto en la tecnología: Esta observación ha sido fundamental para la industria de la tecnología, ya que ha impulsado el desarrollo de dispositivos más potentes y eficientes con el paso de los años. La Ley de Moore ha sido un catalizador para la innovación tecnológica en la fabricación de chips y en la industria de semiconductores en general.

De forma general, una "máquina de Moore" no se refiere a un dispositivo específico, sino más bien al principio general del crecimiento exponencial en la capacidad de procesamiento de las computadoras debido al aumento constante en el número de transistores en los microprocesadores.

Desarrollo

Implementación en el Código

En el código, primero implementamos un módulo que define una máquina de estados de Moore con cuatro estados (E0, E1, E2 y E3). La máquina tiene una entrada de reloj (`clk_i`), una señal de reinicio (`rst_ni`), una entrada (`x_i`) y una salida (`y_o`). La salida `y_o` depende únicamente del estado actual de la máquina, no de las entradas.

Parámetros y Entradas/Salidas

1. Parámetros:

   `E0`, `E1`, `E2`, `E3`: Definen los cuatro estados posibles de la máquina, codificados como `2'b00`, `2'b01`, `2'b10` y `2'b11` respectivamente.

2. Entradas:

- `clk_i`: Señal de reloj.
- `rst_ni`: Señal de reinicio activa en bajo.
- `x_i`: Entrada de la máquina de estados.

3. Salidas:

   `y_o`: Salida de la máquina de estados.

Registro del Estado Actual

`actual_state_w`: Registro de 2 bits que almacena el estado actual de la máquina.

Proceso Secuencial de Transición de Estados

El bloque `always` sensitivo al flanco positivo del reloj (`posedge clk_i`) y al flanco negativo del reinicio (`negedge rst_ni`) define la lógica para la transición de estados:

- Reinicio: Si `rst_ni` es bajo (`0`), la máquina se reinicia al estado `E0`.

- Transiciones de Estado:

  a) En `E0`, si `x_i` es alto (`1`), permanece en `E0`; si `x_i` es bajo (`0`), transita a `E1`.

  b) En `E1`, si `x_i` es alto (`1`), transita a `E2`; si `x_i` es bajo (`0`), transita a `E3`.

  c) En `E2`, si `x_i` es alto (`1`), permanece en `E2`; si `x_i` es bajo (`0`), transita a `E3`.

  d) En `E3`, si `x_i` es alto (`1`), permanece en `E3`; si `x_i` es bajo (`0`), transita a `E0`.

Proceso Combinacional para Definir la Salida

El segundo bloque `always` es un bloque combinacional que define la salida `y_o` en función del estado actual:

- Si el estado es `E0` o `E2`, la salida `y_o` es alta (`1`).
- Si el estado es `E1` o `E3`, la salida `y_o` es baja (`0`).

De forma más concreta, podemos decir que:

- Estado `E0`: Salida `y_o = 1`. Permanece en `E0` si `x_i` es `1`, cambia a `E1` si `x_i` es `0`.
- Estado `E1`: Salida `y_o = 0`. Cambia a `E2` si `x_i` es `1`, cambia a `E3` si `x_i` es `0`.
- Estado `E2`: Salida `y_o = 1`. Permanece en `E2` si `x_i` es `1`, cambia a `E3` si `x_i` es `0`.
- Estado `E3`: Salida `y_o = 0`. Permanece en `E3` si `x_i` es `1`, cambia a `E0` si `x_i` es `0`.

```
2     Proyecto: MaquinaMoore
3     Archivo: MaquinaMoore.v
4     Descripción:Implementación de una máquina de estados de Moore en Verilog.
5     Esta máquina de estados tiene cuatro estados (E0, E1, E2, E3).
6     Asignatura: DSD
7     Profesor: Flores Escobar Jose Antonio
8     Equipo: Alvarez Hernandez Gabriel Alexander
9     Garcia Quiroz Gustavo Ivan
10    Huesca Laureano Josue Alejandro
11    Muños Valdivia Irving Omar
12    Pedroza Villagomez Emir
13 */
14 module MaquinaMoore  #(
15    parameter   E0 = 2'b00,
16    parameter   E1 = 2'b01,
17    parameter   E2 = 2'b10,
18    parameter   E3 = 2'b11
```

```
)
(
    input       clk_i,
    input       rst_ni,
    input       x_i,
    output reg  y_o
);
    reg   [1:0] actual_state_w;
    // Proceso secuencial para calcular los estados
    always @(posedge clk_i, negedge rst_ni)
    begin
       if(!rst_ni)
          actual_state_w <= E0;
       else
          case (actual_state_w)
             E0:
                if(x i)
```

```
                   actual_state_w <= E0;
                else
                   actual_state_w <= E1;
             E1:
                if(x_i)
                   actual_state_w <= E2;
                else
                   actual_state_w <= E3;
             E2:
                if(x_i)
                   actual_state_w <= E2;
                else
                   actual_state_w <= E3;
             E2:
                if(x_i)
                   actual_state_w <= E2;
                else
```

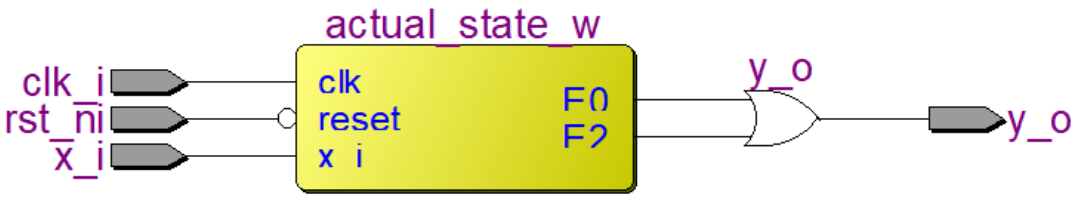
```
                   actual_state_w <= E3;
             E3:
                if(x_i)
                   actual_state_w <= E3;
                else
                   actual_state_w <= E0;
          endcase
    end

    //Proceso pata definir las salidas
    always @(*)
    begin
       case(actual_state_w)
          E0:
             y_o = 1'b1;
          E1:
             y_o = 1'b0;
```

```
67              y_o = 1'b1;
68          E1:
69              y_o = 1'b0;
70          E2:
71              y_o = 1'b1;
72          E3:
73              y_o = 1'b0;
74      endcase
75  end
76
77  endmodule
```

RTL Viewer

En el RTL podemos observar lo siguiente:



1. Entradas:

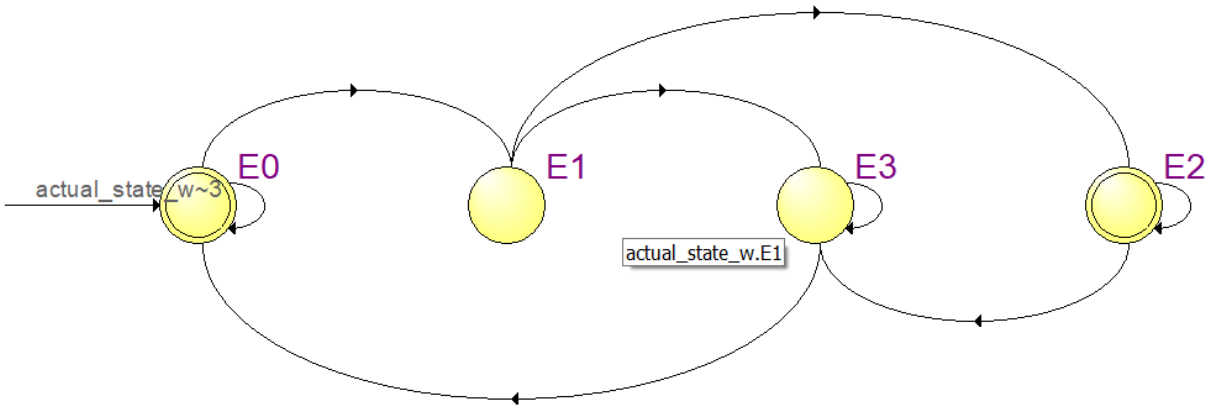
- clk_i: Señal de reloj.
- rst_ni: Señal de reset.
- x_i: Señal de entrada que afecta las transiciones de estado.

2. Salidas:

- y_o: Salida de la máquina de estados.

3. Componente Principal:

- actual_state_w: Registro que mantiene el estado actual



1. Estados:

- E0: Estado inicial.
- E1, E2, E3: Estados intermedios.

2. Transiciones:

- Las transiciones entre los estados claramente definidas con flechas que indican el movimiento de un estado a otro basado en las condiciones de entrada.

3. Condiciones de Transición:

- actual_state_w ~3: Condición de transición
- actual_state_w.E1: Esta es otra condición de transición.

Conclusiones

Durante esta práctica, nuestro equipo implementó una máquina de estados de Moore en Verilog. Esta experiencia nos permitió entender cómo diseñar y estructurar un circuito digital que pueda cambiar de estado en respuesta a entradas específicas y generar salidas basadas únicamente en su estado interno. Primero, definimos los parámetros y entradas necesarios para la máquina de estados, estableciendo los estados `E0`, `E1`, `E2`, y `E3`. Utilizamos una señal de reloj para sincronizar las transiciones de estado y una señal de reinicio activa baja para restablecer la máquina al estado inicial `E0`. La entrada `x_i` determinó las transiciones de estado según condiciones específicas para cada estado. Implementamos dos bloques `always`: uno para controlar las transiciones de estado y otro para definir la salida `y_o`. El primero aseguró que la máquina cambiara de estado de manera secuencial y conforme a las reglas establecidas, mientras que el segundo determinó la salida `y_o` según el estado actual de la máquina. A través de esta práctica, consolidamos nuestro entendimiento sobre la importancia de la sincronización de señales en diseño digital y la aplicación de máquinas de estados finitos para controlar secuencialmente un sistema basado en entradas discretas. Además, fortalecimos nuestra habilidad para escribir código en Verilog que sea claro, eficiente y capaz de cumplir con especificaciones funcionales.

Bibliografía

- https://delta.cs.cinvestav.mx/~gmorales/ta/node50.html