INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Carrera:

Ingeniería en sistemas computacionales

Unidad de Aprendizaje:

Sistemas en chip

Entregable 1:

Práctica de segmentación - Verilog

Integrantes:

Cazares Cruz Jeremy Sajid

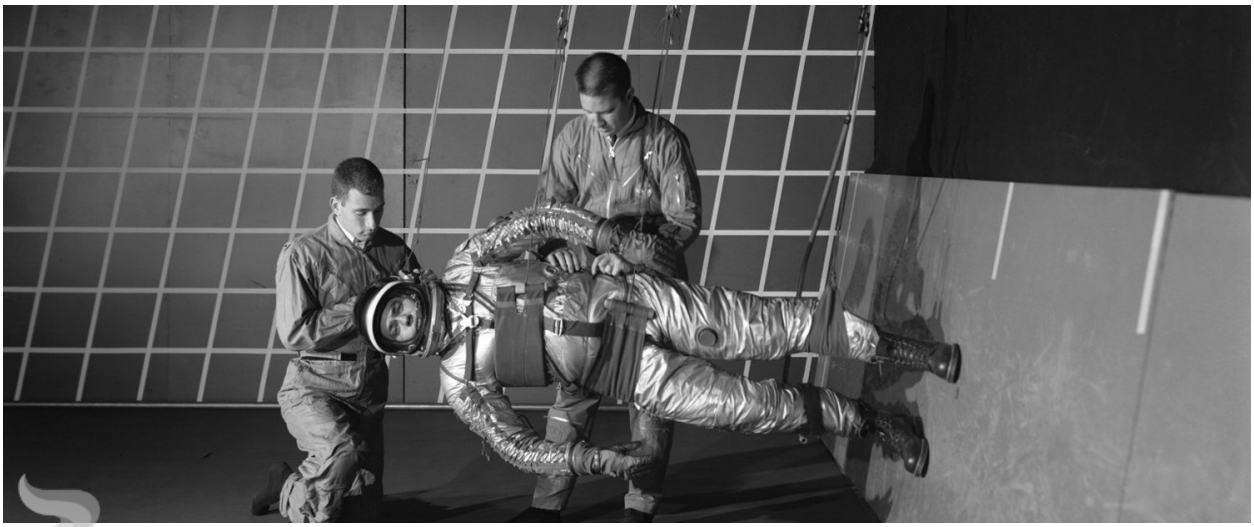
Bucio Barrera Oscar Daniel

Perez Ortiz Saúl

Acosta Cortes Gerardo

Fecha de entrega

18 de septiembre de 2023



Reporte Práctica 1

Introducción

Quartus es una herramienta de diseño para circuitos electrónicos, la cual se utiliza ampliamente para la programación y simulación de dispositivos FPGA. Al iniciar un proyecto en Quartus, se comienza seleccionando el tipo de dispositivo FPGA a utilizar, configurando la familia de dispositivos, y especificando el modelo exacto. Posteriormente, se crea un esquema o se escribe el código en un lenguaje de descripción de hardware como Verilog o VHDL. Una vez escrito el código, se procede a la compilación, simulación y finalmente a la programación del dispositivo.

En esta práctica, se ha desarrollado un sumador segmentado utilizando el lenguaje de descripción de hardware Verilog. A continuación, se desglosa y explica detalladamente cada parte del código presentado.

Análisis del Código

```
module sumador (
  input     clk,
  input [3:0] A,
  input [3:0] B,
  input     Cin,
  output reg [3:0] S1,
```

```
  output reg     cout1
);
```

En esta sección, se declara el módulo principal llamado "sumador". Este módulo cuenta con las siguientes entradas y salidas:

- Entradas:
  - `clk`: Es la señal de reloj que sincroniza las operaciones del circuito.
  - `A[3:0]`: Es un bus de 4 bits que representa el primer número a sumar.
  - `B[3:0]`: Es un bus de 4 bits que representa el segundo número a sumar.
  - `Cin`: Es el acarreo de entrada.
- Salidas:
  - `S1[3:0]`: Es un bus de 4 bits que representa el resultado de la suma.
  - `cout1`: Es el acarreo de salida resultante de la operación de suma.

A continuación, se declaran algunos registros intermedios y señales:

```
reg [1:0] A1Low;
reg [1:0] B1Low;
reg [1:0] A1High;
reg [1:0] B1High;
reg      Cin1;
wire [3:0] S;
wire     Cout;
```

Estos registros y señales son utilizados para segmentar la suma en partes, permitiendo que la operación sea más modular. La idea es dividir los números de entrada en dos partes: la parte alta y la parte baja. Por ejemplo, `A1Low` y `A1High` representan las partes baja y alta del número A, respectivamente.

La siguiente sección del código se encarga de registrar las entradas con cada flanco ascendente del reloj:

```
always @(posedge clk)
begin
  A1Low <=   A[1:0];
  B1Low <=   B[1:0];
  A1High <=   A[3:2];
```

```
  B1High <=   B[3:2];
  Cin1 <= Cin;
end
```

Aquí, con cada flanco ascendente del reloj, se actualizan los valores de `A1Low`, `B1Low`, `A1High`, `B1High` y `Cin1` tomando los valores correspondientes de las entradas A, B y Cin.

Continuando con el análisis, se declaran algunas señales adicionales:

```
wire Cout0;
wire Cout1;
wire Cout2;
```

Estas señales representan acarreos intermedios que se generan durante la suma segmentada.

La siguiente parte del código se encarga de sumar la parte baja de los números:

```
sumador1 FA0 (A1Low[0], B1Low[0], Cin1, S[0], Cout0);
sumador1 FA1 (A1Low[1], B1Low[1], Cout0, S[1], Cout1);
```

Aquí, se utilizan dos instancias de un módulo llamado "sumador1" (que no está presente en el código proporcionado, pero se asume que es un sumador de un bit). La primera instancia, `FA0`, suma el bit menos significativo de `A1Low` y `B1Low` junto con el acarreo de entrada `Cin1`. La segunda instancia, `FA1`, suma el siguiente bit de `A1Low` y `B1Low` utilizando el acarreo de salida de la primera instancia (`Cout0`).

Luego, se utiliza un latch de interetapa para registrar los valores de la parte alta y los resultados parciales de la suma:

```
reg [1:0] A1High1;
reg [1:0] B1High1;
reg [1:0] Slow;
reg      Cout1_r;
always @(posedge clk)
begin
A1High1 <= A1High;
B1High1 <= B1High;
Slow    <= S[1:0];
```

```
  Cout1_r <= Cout1;
  end
```

Estos registros almacenan temporalmente los valores que serán utilizados en la siguiente etapa de la suma.

Posteriormente, se suman los bits de la parte alta:

```
  sumador1 FA2 (A1High1[0], B1High1[0], Cout1_r, S[2], Cout2);
  sumador1 FA3 (A1High1[1], B1High1[1], Cout2, S[3], Cout);
```

Al igual que en la suma de la parte baja, se utilizan dos instancias del módulo "sumador1". La primera instancia, `FA2`, suma el bit menos significativo de `A1High1` y `B1High1` utilizando el acarreo registrado `Cout1_r`. La segunda instancia, `FA3`, suma el siguiente bit utilizando el acarreo de salida de la primera instancia (`Cout2`).

Finalmente, se registran las salidas:

```
  always @(posedge clk)
  begin
  S1 <= {S[3:2],Slow};
  cout1 <= Cout;
  end
endmodule
```

El resultado de la suma, `S1`, se forma concatenando los bits de la parte alta de la suma (`S[3:2]`) con los bits registrados de la suma de la parte baja (`Slow`). El acarreo de salida, `cout1`, toma el valor de `Cout`.

Conclusión

El código presentado implementa un sumador segmentado de 4 bits. La segmentación permite modularizar la operación de suma, dividiendo el proceso en etapas que suman partes específicas de los números. Esta modularidad facilita el diseño, la optimización y la depuración del circuito. A través de este análisis detallado,