Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Cuestionario Previo 1

Taller de diseño digital

Integrantes:

Arias Ortiz Kendy Raquel

Pérez Jiménez Jorge

Fabián M. Villegas Bonilla

Adrián Parajeles Alvarado

Profesor:

Dr.-Ing. Roberto Molina Robles

27 de febrero de 2025

Lab. 1: Introducción al diseño digital con HDL y herramientas EDA de síntesis

Preguntas

1. **Explique qué es el modelado de comportamiento y de estructura en diseño digital. Brinde un ejemplo de cada uno empleando SystemVerilog.**

   El modelado de comportamiento y de estructura en diseño digital son los medios por los que se puede crear e interactuar con sistemas computacionales. Por medio de un lenguaje computacional se pueden desarrollar configuraciones que asemejan un hardware que sea sencilla para los usuarios. **HDL** (Hardware Description Language) es este lenguaje que se utiliza para codificar circuitos digitales, según su estructura y comportamiento. **SystemVerilog** (La versión mejorada de Verilog) se utiliza como un sintetizador que controla un procesador del Hardware que por medio de registros y compuertas lógicas se fabricará una serie de circuitos con él propósito de ver su desarrollo en una FPGA. (Brown, & Vranesic, pp.53-59).

   Con el lenguaje de descripción se puede modelar el comportamiento de sistema, estos módulos tienen un nombre y forman piezas de hardware. Cada módulo debe tener nombre, entradas y salidas que faciliten el esquema de trabajo.

```
Module registro(
input logic       ent1,
input logic       ent2,
input logic       ent3,
output logic      salida
);
endmodule
```

   Una vez que definimos la estructura de un módulo, lo que sigue es especificar la funcionalidad del módulo. Para ello, utilizamos lo que se conoce como bloques procesales (procedural blocks).

   - `initial`
   - `always`
   - `always_comb`
   - `always_ff`
   - `always_latch`

   Estos bloques se dividen en dos partes:

   1. **Modelado combinacional**: Sigue un patrón lógico conforme llegan las instrucciones.
   2. **Modelado secuencial**: Requiere un reloj que controle el tiempo de ejecución. Si no está en el flanco positivo o negativo según se indique, la tarea no se ejecutará.

Si se requiere un sistema controlado por un reloj, se deben agregar ciertas instrucciones que mantengan control del tiempo, como se muestra a continuación.

```
Reset síncronico:
always_ff(posedge clk) begin
    if(reset) salida =>0;
    salida<=(...)
end
Reset asíncronico:
always_ff(posedge clk) begin
    if(reset) salida <=0;
    salida<=(...)
end
```

Estos resets permiten que la entrada de un módulo secuencial siempre comience con un valor conocido. De lo contrario, no se sabría con qué inicia el proceso, lo que podría generar problemas.

El uso de estos bloques procesales depende de, si el módulo implementado es combinacional, secuencia o testbench. El siguiente módulo muestra una estructura muy común con el nombre de Alu.

```
module Alu
#(
 parameter N = 7
 )
 (
 input logic [N:0] A,
 input logic [N:0] B,
 input logic [3:0]opcode,

 output logic [N:0] result,
 output logic flag
    );

 always_comb begin
    case(opcode)
        4'b0000: result = A + B;
        4'b0001: result = A - B;
        4'b0010: result = A & B;
        4'b0110: result = A | B;
    endcase
end
 assign flag = (result==0) ? 1 : 0;
endmodule
```

Una forma más estructural sería la Alu, esta es una pieza que manda instrucciones dependiendo de lo que se llegue a los puertos de entrada, como se puede observar en la el módulo Alu esta tiene 3 entradas, 2 que indica un valor y 1 que por medio del case(opcode) indica que se harán con los valores de A y B. Cuando se decide que será en la salida, se muestra el resultado según los bits de entrada del opcode. Se van combinando piezas o interconectando componentes que van formando una estructura de varias partes con sus propias indicaciones. (Brown, & Vranesic, pp.53-59).

Modelado de Comportamiento

Busca describir cómo funciona un sistema digital, utilizando un nivel alto de abstracción. Define operaciones, funcionalidades y relaciones de entrada y salida sin especificar su implementación interna.

```
module mux4x1 (
 input logic [3:0] data_in,
 input logic [1:0] sel,
 output logic data_out
);
always_comb begin
 case (sel)
  2'b00: data_out = data_in[0];
  2'b01: data_out = data_in[1];
 2'b10: data_out = data_in[2];
 2'b11: data_out = data_in[3];
  endcase
 end
endmodule
```

Este multiplexor tiene 4 entradas de datos, 2 entradas de selección y 1 salida. Según el valor de `sel`, la salida será la señal de entrada correspondiente.

Modelado de estructura

Describe la composición interna del sistema digital, desde sus componentes hasta la jerarquía y conexiones entre ellos.

Ejemplo: Multiplexor 4 a 1 con compuertas lógicas

```
module mux4x1_structural (
```

```
    input logic d0, d1, d2, d3,
    input logic s0, s1,
    output logic y
);
logic not_s0, not_s1;
logic and0, and1, and2, and3;
// Inversores
assign not_s0 = ~s0;
assign not_s1 = ~s1;
// Compuertas AND
assign and0 = d0 & not_s1 & not_s0;
assign and1 = d1 & not_s1 & s0;
assign and2 = d2 & s1 & not_s0;
assign and3 = d3 & s1 & s0;
// Compuerta OR para la salida
assign y = and0 | and1 | and2 | and3;
endmodule
```

En este modelo:

1. Se usan compuertas NOT para invertir las señales de selección.
2. Se utilizan compuertas AND para generar señales de selección.
3. Una compuerta OR selecciona la entrada de datos correcta y la envía a la salida.

Este diseño estructural describe cómo está compuesto internamente el multiplexor en términos de compuertas lógicas y sus interconexiones.

Conclusión

El modelado de comportamiento y estructural en SystemVerilog permiten desarrollar sistemas digitales de manera eficiente:

- El modelado de comportamiento define qué hace el sistema sin preocuparse por su implementación interna.

- El modelado de estructura detalla cómo está construido el sistema a partir de componentes básicos.

Ambos enfoques son fundamentales en el diseño digital, y su elección depende del nivel de abstracción requerido en cada etapa del desarrollo

2. **Explique el proceso de síntesis lógica en el diseño de circuitos digitales, tanto para el desarrollo de un ASIC como para una FPGA.**

   La síntesis lógica es el proceso de generar un circuito lógico a partir de una especificación inicial, que puede estar dada en formato de diagrama esquemático o como código escrito en un lenguaje de descripción de hardware. El proceso de transcripción o compilación del código **VHDL** a una red de puertas lógicas es parte de la síntesis. El resultado de este proceso es un conjunto de expresiones lógicas que describen las funciones necesarias para realizar el circuito.

   Ahora, **¿Qué es un ASIC?**

   En situaciones en las que el diseñador del chip no necesita flexibilidad completa para el diseño de cada transistor individual, se puede reducir el esfuerzo de diseño utilizando una tecnología llamada celdas estándar. Los chips creados con esta tecnología se conocen comúnmente como **Application-Specific Integrated Circuits** (ASICs).

   Por otro lado, **¿Qué es una FPGA?**

   Una FPGA **(Field-Programmable Gate Array)** es uno de los tipos más sofisticados de PLD (Programmable Logic Device). Contiene cientos de millones de transistores y permite la creación de circuitos digitales complejos. Su arquitectura está formada por un número variable de elementos lógicos que se organizan en una estructura regular bidimensional. Estos elementos lógicos pueden ser conectados y configurados por switches programables, lo que permite al diseñador personalizar la funcionalidad del circuito de acuerdo a sus necesidades.

   La **síntesis lógica en una FPGA** y en un **ASIC** comparten varios aspectos fundamentales, como la conversión de una descripción de alto nivel (VHDL/Verilog) en una red de puertas lógicas. Sin embargo, existen diferencias clave debido a la naturaleza de cada tecnología:

Diferencias clave:

1. **Reconfigurabilidad vs. Personalización**

   FPGA: La FPGA es un dispositivo reconfigurable. Esto significa que una vez que se ha programado con un archivo bitstream, puede ser reprogramado en el futuro para realizar otros diseños. Las celdas lógicas y la conectividad entre ellas son programables.

   ASIC: En contraste, un **ASIC** es un circuito integrado específico para una aplicación. Una vez que se ha fabricado, no se puede modificar. El diseño es "fijo" y se realiza para un uso específico, lo que requiere más esfuerzo y costo en el proceso de diseño y fabricación.

2. **Estructura de Recursos Lógicos**

FPGA: Las FPGAs tienen una **estructura de celdas lógicas programables** (como LUTs, flip-flops) que el diseñador conecta y configura utilizando herramientas de síntesis. El mapeo de estos recursos es flexible, y el hardware se configura dinámicamente.

ASIC: En un ASIC, el diseño es completamente personalizado. El proceso de síntesis lógica y el mapeo de celdas lógicas en los transistores y conexiones es más controlado y específico a las necesidades del diseño. No hay reconfiguración una vez fabricado el chip.

3. **Optimización**

   FPGA: En las FPGAs, las optimizaciones tienden a estar más limitadas debido a las restricciones de las celdas lógicas programables. El proceso de optimización se enfoca principalmente en la eficiencia del uso de recursos lógicos y el rendimiento dentro de las limitaciones de la FPGA.

   ASIC: Los ASICs permiten optimizaciones más profundas, ya que el diseño es personalizado desde cero. Las optimizaciones pueden ser a nivel de transistor, interconexión y desempeño, lo que permite alcanzar mayores rendimientos y menor consumo de energía.

4. **Tiempo de Desarrollo y Costo**

   FPGA: El tiempo de desarrollo es generalmente **más corto** y el costo es **más bajo**, ya que no hay necesidad de pasar por un proceso de fabricación complejo. Además, se pueden realizar pruebas más rápidamente, y el dispositivo puede ser reutilizado o reprogramado para otros propósitos.

   ASIC: Los ASICs requieren un proceso de diseño **mucho más largo** y costoso. Esto incluye la fabricación de máscaras y la producción en masa, lo que hace que la inversión inicial sea considerablemente más alta.

Proceso de síntesis lógica en FPGA vs. ASIC (Diferencias)

1. **Mapeo de recursos**:

   FPGA: Los recursos lógicos son predefinidos y programables, y el proceso de síntesis mapea el diseño en estos bloques reutilizables (como LUTs, flip-flops).

   ASIC: El mapeo de recursos es más específico y depende de la tecnología de fabricación elegida. Las celdas lógicas son diseñadas específicamente para el circuito, y no son reconfigurables como en las FPGAs.

2. **Fabricación**:

   FPGA: Después de la síntesis, se genera un archivo de bitstream para programar la FPGA. El dispositivo ya está disponible y listo para ser configurado.

   ASIC: Una vez completado el diseño, se debe pasar por un proceso de fabricación que puede incluir la creación de máscaras para fotolitografía, lo que hace que la fabricación de un ASIC sea un proceso mucho más largo y costoso.

3. **Investigue sobre la tecnología de FPGAs. Describa el funcionamiento de la lógica programable en general, así como los componentes básicos de un chip FPGA.**

Un FPGA (Field Programmable Gate Array) es un tipo de circuito integrado que, a diferencia de otros chips con funciones predefinidas, se entrega sin configuración, permitiendo que cada usuario lo programe y adapte según sus necesidades específicas. Esto significa que una vez fabricado, el cliente puede personalizar su funcionamiento para ejecutar distintas tareas.

Estos dispositivos suelen incorporar diversos bloques de procesamiento y múltiples conectores que posibilitan la configuración de la lógica interna, permitiendo desde operaciones simples hasta cálculos complejos. Dependiendo del modelo, algunos FPGAs también incluyen memoria interna, especialmente en aquellos con estructuras lógicas más avanzadas.

Componentes de un FPGA

Un FPGA está compuesto por varios elementos esenciales que trabajan conjuntamente para facilitar la implementación de circuitos digitales personalizables. Entre sus principales componentes se encuentran:

Bloques lógicos configurables (CLB): Son las unidades fundamentales del FPGA, integrando tablas de consulta (LUT), registros (flip-flops) y, en algunos casos, unidades aritméticas específicas.

Puntos de interconexión programables (PIP): Son los recursos de conexión y enrutamiento programables que vinculan los CLB y otros elementos dentro del FPGA, permitiendo una comunicación flexible entre las diferentes partes del chip.

Bloques de entrada/salida (IOB): Actúan como interfaz entre el FPGA y dispositivos externos o componentes adicionales en la placa, permitiendo la compatibilidad con distintos estándares y protocolos de señalización.

Memoria RAM en bloque (BRAM): Es la memoria integrada en el FPGA utilizada para almacenar datos y ejecutar funciones de memoria con menor latencia en comparación con las memorias externas.

Bloques de procesamiento de señales digitales (DSP): Son módulos especializados en la ejecución rápida de operaciones matemáticas, como multiplicaciones y acumulaciones, optimizando el procesamiento de señales.

Recursos de gestión de reloj: Incluyen componentes como los bucles de sincronización de fase (PLL) y los administradores de reloj digital (DCM), encargados de generar y distribuir las señales de reloj dentro del FPGA.

Memoria de configuración: Se trata de la memoria no volátil donde se almacena el flujo de bits que define la configuración del FPGA para ejecutar un diseño digital determinado.

Referencias

1. Brown, S., & Vranesic, Z. (2009). *Fundamentals of Digital Logic with VHDL Design* (3rd ed.). McGraw-Hill.
2. Geeknetic, "Qué es una FPGA y para qué sirve," *Geeknetic*. [En línea]. Disponible: https://www.geeknetic.es/FPGA/que-es-y-para-que-sirve.
3. F. J. Reyes Gil, "Arquitecturas basadas en FPGAs," Universidad de Sevilla, 2008. [En línea]. Disponible: https://biblus.us.es/bibing/proyectos/abreproy/11375/fichero/MEMORIA%252FFPGAs.pdf.
4. Wevolver, "What is an FPGA? Field Programmable Gate Array Overview," *Wevolver*. [En línea]. Disponible: https://www.wevolver.com/article/fpga.