Diseño de Sistema Usando HDL

José Bernardo Barquero Bonilla
Carné: 2023150476
Correo: jos.barquero@estudiantec.cr
Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

Jose Eduardo Campos Salazar
Carné: 2023135620
Correo: j.campos@estudiantec.cr
Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

Jimmy Feng Feng
Carné: 2023060347
Correo: jifeng@estudiantec.cr
Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

Alexander Montero Vargas
Carné: 20233166058
Correo: ale_montero@estudiantec.cr
Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

I. Introducción

En el presente documento, se muestra el desarrollo e implementación de un sistema digital utilizando la FPGA DE10-Standard y el lenguaje de descripción de hardware SystemVerilog. El objetivo principal fue diseñar un circuito combinacional-secuencial que integre diferentes módulos como registros, decodificadores y controladores de display, siguiendo un diseño estructural. La implementación contempló el uso de entradas binarias a través de interruptores, el almacenamiento temporal de datos mediante registros tipo D, la interpretación de combinaciones binarias con decodificadores, y la visualización de resultados en un display de siete segmentos.

II. Marco Teórico

A. ¿Qué es un HDL? ¿Por qué un HDL como SystemVerilog tiene funciones que no se pueden sintetizar?

Un lenguaje de descripción de hardware (HDL) es un lenguaje especializado que permite describir el comportamiento, la estructura y la organización de circuitos digitales. Los HDL permiten modelar sistemas digitales de forma textual, facilitando su simulación y síntesis para ser implementados físicamente en dispositivos como FPGAs o ASICs [1].

SystemVerilog es una extensión del lenguaje Verilog que incorpora características de modelado de alto nivel, tales como programación orientada a objetos y estructuras de control complejas. No todas estas funciones son sintetizables, es decir, no todas pueden traducirse directamente a lógica digital en hardware. Por ejemplo, construcciones como $assert$, ciclos infinitos o tareas específicas para simulación ($display$, $monitor$) están pensadas para pruebas y verificación, pero no tienen una representación física en puertas lógicas [2].

B. ¿Cuáles son algunos ejemplos de lenguajes HDL? ¿Cuáles industrias los usan más habitualmente?

Los siguientes lenguajes HDL son los más utilizados en la industria:

- Verilog es comúnmente utilizado en la industria estadounidense y se caracteriza por su sintaxis similar a C.
- VHDL, desarrollado por el Departamento de Defensa de los Estados Unidos, es más detallado y se utiliza frecuentemente en aplicaciones críticas, como aeroespacial y sistemas embebidos seguros [3].
- SystemVerilog es una evolución de Verilog, diseñado para la verificación y diseño de circuitos complejos, y ampliamente adoptado en el diseño de SoCs (System on Chip) por empresas como Intel, AMD o NVIDIA [4].

La industria de los semiconductores, automatización industrial, telecomunicaciones y defensa son los principales usuarios de estos lenguajes debido a su necesidad de diseñar y verificar circuitos digitales complejos.

C. ¿Qué es un modelo estructural y un modelo comportamental? ¿Por qué se le dice caja transparente al diseño estructural?

Un modelo estructural en HDL describe explícitamente cómo se interconectan los distintos componentes del circuito, como si se tratara de un diagrama esquemático. Es una representación jerárquica donde cada subcomponente está instanciado y conectado manualmente. Por esta razón se le llama "caja transparente", ya que permite observar internamente cómo está construido el sistema [5].

Por otro lado, un modelo comportamental describe qué hace el sistema, sin especificar cómo está construido internamente. Se utiliza para describir la funcionalidad con alto nivel de abstracción, lo cual es útil en fases tempranas del diseño o para módulos simples [2].

III. Diseño en Hardware

La implementación física del sistema se realizó utilizando componentes digitales discretos montados sobre una protoboard, alimentados mediante una fuente externa regulada de 5V. En la Figura 1 se presenta el esquemático general del circuito construido.

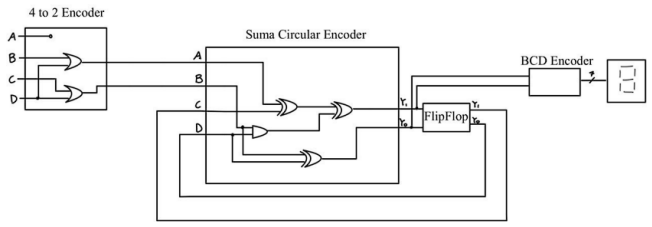

Fig. 1. Esquemático del diseño implementado físicamente

Las entradas del sistema corresponden a cuatro señales binarias (denotadas en el circuito como A, B, C y D), las cuales fueron generadas mediante sensores fotoresistivos con resistencias de pull-down. Estas señales se conectaron al módulo **4 a 2 Encoder**, implementado utilizando compuertas OR para generar dos bits de salida codificada binaria (denotados como A y B en la siguiente etapa). Cabe destacar que, debido a una diferencia de nomenclatura en el diagrama, la señal etiquetada como D representa en realidad la entrada lógica A, y viceversa.

Las salidas del encoder (A, B) fueron conectadas al bloque denominado **Suma Circular Encoder**. Este módulo fue implementado utilizando compuertas XOR y AND para realizar una operación de suma circular sobre las señales actuales y su retroalimentación. La operación implementada fue:

- $Y_1 = A \oplus B$
- $Y_0 = (A_{\text{prev}} \oplus B_{\text{prev}}) \oplus (A \wedge B)$

Los resultados Y_1 y Y_0 fueron enviados a un **registro tipo D**, el cual almacena estos valores utilizando un botón físico como señal de reloj (CLK), conectado con resistencia pull-down, que actualiza los valores en el flanco positivo. También se utilizó un segundo botón como señal de borrado asíncrono (CLR), permitiendo reiniciar las salidas del registro a cero.

Las salidas del registro (Y_{r1}, Y_{r0}) cumplen dos funciones: por un lado, se retroalimentan al bloque de suma circular como entradas anteriores, lo cual confiere al sistema un comportamiento secuencial acumulativo; por otro lado, son dirigidas al módulo **codificador BCD**. Las salidas de este módulo (segmentos a–g) fueron conectadas directamente a un display de siete segmentos de cátodo común, el cual fue configurado como activo bajo.

Finalmente, se incluyó una etapa adicional de **desacople**, consistente en una compuerta XOR que opera sobre las señales codificadas A y B (provenientes del encoder 4 a 2). La salida de esta XOR controla la base de un transistor BJT utilizado como interruptor para activar un actuador externo, como un pequeño motor. Esta configuración permite generar una señal física de salida basada en el resultado de la codificación inicial.

IV. Diseño en HDL

En esta sección se describen los diferentes módulos desarrollados en SystemVerilog que conforman el diseño completo. Para cada módulo se presenta una captura de pantalla de su simulación RTL obtenida mediante Quartus, junto con una explicación de su funcionalidad principal.

A. Codificador 4 a 2

En la Figura 2 se muestra la simulación RTL del módulo `Enco_Four_Two`, encargado de convertir cuatro señales de entrada en una representación binaria de dos bits.

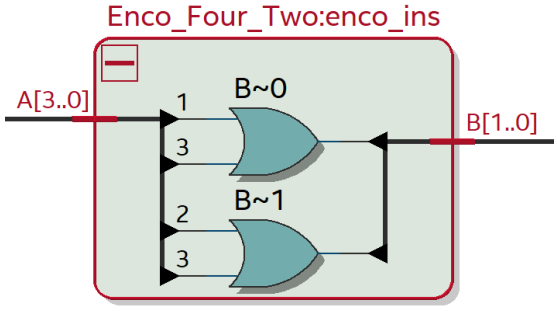

Fig. 2. Simulación RTL del módulo `Enco_Four_Two`

Este módulo implementa un codificador 4 a 2, el cual recibe como entrada cuatro líneas activas ($A[3:0]$) y produce una salida codificada de dos bits. Las expresiones booleanas utilizadas son:

- $B[0] = A[3] \vee A[1]$
- $B[1] = A[3] \vee A[2]$

Este módulo permite transformar múltiples entradas provenientes de sensores (fotoresistencias) en una representación binaria más compacta.

B. Flip-Flop tipo D

A continuación, en la Figura 3 se muestra la simulación RTL del módulo `Flip_Flop_D`, utilizado como almacenamiento temporal en el sistema.

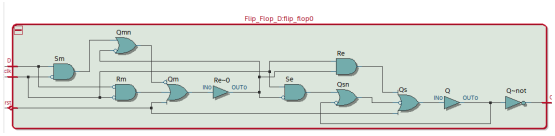

Fig. 3. Simulación RTL del módulo `Flip_Flop_D`

Este módulo implementa un flip-flop tipo D a nivel estructural, compuesto por dos bloques *latch* maestro-esclavo. El dato de entrada es almacenado en la salida Q en el flanco positivo del reloj, y puede ser reiniciado de forma asíncrona mediante la señal de *reset* activo en alto. Su propósito dentro del sistema es retener el estado de una de las salidas parciales de la operación de suma circular, permitiendo la retroalimentación del valor previamente almacenado para ser reutilizado en el siguiente ciclo de procesamiento.

C. Suma Circular

En la Figura 4 se presenta la simulación RTL del módulo `Enco_Sum`, encargado de realizar la operación de suma circular entre dos operandos binarios.

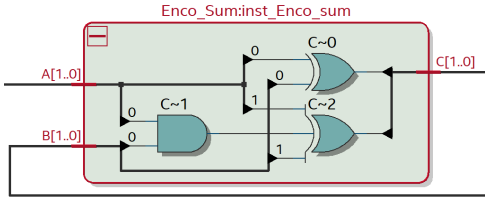

Fig. 4. Simulación RTL del módulo `Enco_Sum`

Este módulo realiza una operación de suma circular sobre dos operandos de 2 bits, utilizando una combinación de compuertas XOR y AND. La salida consiste en un valor binario de 2 bits, donde:

- $C[0] = A[0] \oplus B[0]$
- $C[1] = (A[1] \oplus B[1]) \oplus (A[0] \wedge B[0])$

Este tipo de operación permite simular una suma con acarreo circular, donde el resultado nunca sobrepasa los 2 bits. Los operandos de entrada provienen del codificador 4 a 2 y del valor previamente almacenado en registros tipo flip-flop. Todo el proceso es secuencial y responde a señales de reloj y reset controladas desde pines GPIO protegidos mediante el conversor bidireccional `TXS0108E`.

D. Codificador BCD a Display de 7 segmentos

En la Figura 5 se muestra la simulación RTL del módulo `BCD_seven_seg`, encargado de controlar la visualización en el display de siete segmentos.

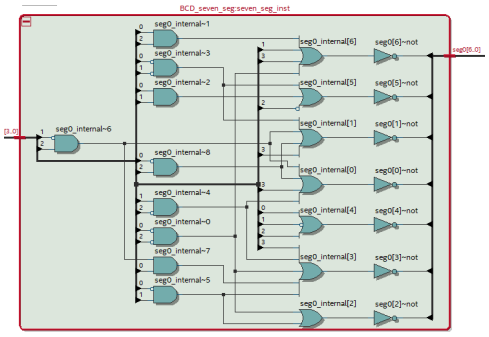

Fig. 5. Simulación RTL del módulo `BCD_seven_seg`

Este módulo toma como entrada un valor codificado en formato BCD de 4 bits, y genera la activación de los siete segmentos del display (además de un bit no utilizado). Cada segmento se calcula utilizando expresiones booleanas optimizadas para representar correctamente los dígitos del 0 al 3. Las salidas son activas en bajo, por lo que se realiza una negación final de los bits internos.

El valor de entrada se genera por los módulos internos del sistema y es representado visualmente en el display de la FPGA conectado a través de los pines HEX. Este display es de cátodo común y opera con lógica activa baja. Las salidas del módulo están preparadas para ser conectadas directamente al hardware de visualización.

E. Módulo de Desacople

En la Figura 6 se muestra la simulación RTL del módulo `desacople`, utilizado para activar un actuador físico externo.

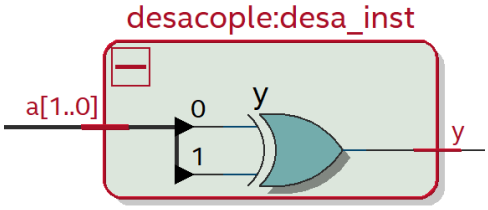

Fig. 6. Simulación RTL del módulo `desacople`

Este módulo implementa una compuerta XOR entre los dos bits de salida del registro (z1 y z0). El resultado se utiliza como señal lógica de control para activar un desacople (transistor BJT) que enciende un motor u otro actuador. Su salida y está conectada a un pin GPIO configurado como salida digital.

F. Módulo: Topmodule

En la Figura 7 se muestra la simulación RTL del módulo `Topmodule`, el cual integra todos los bloques lógicos que componen el sistema completo.

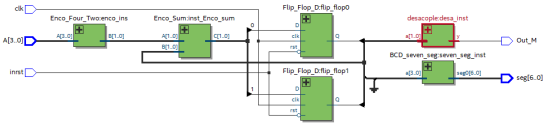

Fig. 7. Simulación RTL del módulo `Topmodule`

El módulo `Topmodule` coordina todo el funcionamiento del sistema, integrando los bloques de codificación, suma circular, almacenamiento con flip-flops, conversión BCD y salida de control para el actuador. Las entradas $A[3:0]$ provienen de sensores externos conectados a través de pines GPIO de entrada. La señal de reloj clk y la señal de reinicio $inrst$ también provienen de botones físicos conectados a pines GPIO con resistencias pull-down.

Todas estas señales externas, al provenir de un entorno de 5V, se adaptan a 3.3V utilizando el conversor de nivel lógico bidireccional `TXS0108E`, garantizando así la protección eléctrica de la FPGA DE10-Standard. La salida hacia el actuador se realiza mediante otro pin GPIO configurado como salida digital, también protegido mediante el conversor.

Este módulo define la lógica secuencial general del sistema, operando sobre los datos recibidos y manteniendo el estado mediante retroalimentación desde los registros internos.

Referencias

[1] D. A. Pucknell y K. Eshraghian, *Basic VLSI design*. PHI Learning Pvt. Ltd., 2003.

[2] J. Bhasker, *Verilog HDL: A Guide to Digital Design and Synthesis*. Prentice Hall, 2003.

[3] P. J. Ashenden, *The Designer's Guide to VHDL*. Morgan Kaufmann, 2008.
[4] S. H. Gerez, *A Course in Digital Design and Synthesis with VHDL*. Wiley, 2006.
[5] J. F. Wakerly, *Digital Design: Principles and Practices*. Pearson Education, 2006.