Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
EL3313 Taller de Diseño Digital
Prof. M.Sc. Kaled Alfaro Badilla
II semestre 2024

Documentación del laboratorio 2: Lógica secuencial

Grupo 1

Integrantes:
Brayan de Jesús Barquero Madrigal
Keylor David Muñoz Soto
Jeffrey Isaac Salas Quiel

02 Octubre del 2024

Índice

1. Ejercicio 1

El objetivo de este ejercicio fue desarrollar un sistema antirebotes y sincronizador para procesar señales provenientes de pulsadores e interruptores en una FPGA, utilizando SystemVerilog. Para ello, se implementó un bloque digital que elimina los rebotes en las señales de entrada y asegura su sincronización con el reloj del sistema. Posteriormente, se conectó un contador de 6 bits (se utilizaron los LEDs de la FPGA) que se incrementa solo cuando la señal de habilitación es estable y libre de rebotes.

1.1. Diagrama de bloques

El primer paso fue diseñar un diagrama de bloques, para esto se tuvo en cuenta primero que la entrada debe ser generada y detectada, en este caso key_detect la cual entra al bloque antirebotes y sincronizador debounce para ser procesada y que el bloque counter reciba una señal limpia que puede ser contada sin errores. El diagrama se muestra a continuación y fue realizado con la herramienta Draw.io [1].

Diagrama de bloques del sistema antirebote y sincronizador.

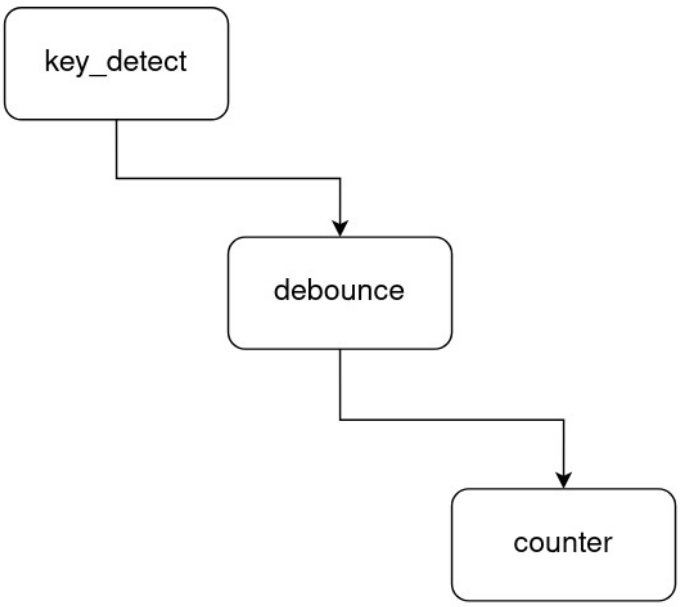

Figura 1: Diagrama de bloques. [1]

1.2. Bloques digitales: Antirebotes y Sincronizador

En este paso se diseño un filtro antirebotes, que se encargó de eliminar las fluctuaciones que ocurren cuando se presiona o libera un botón, lo cual puede provocar múltiples transiciones indeseadas en la señal. Este filtro se combinó con un sincronizador que ajustó las señales al reloj de la FPGA, garantizando que las transiciones se alinearan con los tiempos del sistema. Estos bloques fueron escritos en SystemVerilog.

1.3. Contador con señal habilitadora

Una vez que la señal de entrada fue debidamente procesada, se conectó a un contador de 8 bits. Este contador se diseñó para incrementar su valor únicamente cuando la señal habilitadora, proveniente del bloque antirebote y sincronizador, estaba en un nivel alto estable. El contador es útil para verificar que el sistema antirebotes funciona correctamente, ya que cualquier fluctuación en la señal de entrada se reflejaría en un incremento indeseado.

1.4. Testbench

Se escribió un testbench para simular el comportamiento del sistema ante una señal con rebotes. El testbench generó pulsos con fluctuaciones, y se comprobó que el filtro antirebotes eliminaba dichas fluctuaciones, asegurando que el contador solo respondiera a transiciones limpias y estables. El testbench fue realizado utilizando [2].

1.5. Pruebas en FPGA y conexiones

Para probar el sistema, se utilizó uno de los pulsadores de la tarjeta FPGA como señal de entrada. La señal del pulsador pasó por el bloque antirebote antes de habilitar el contador, lo cual permitió mostrar el valor del contador en los LEDs de la tarjeta. Esta etapa verificó el comportamiento del sistema en hardware, confirmando que no se generaban incrementos debido a los rebotes. Estas pruebas se realizaron con la ayuda de la herramienta [3].

1.6. Descarga y verificación en FPGA

Finalmente, el diseño se descargó a la tarjeta FPGA para realizar pruebas en hardware con [3]. Los resultados coincidieron con las simulaciones realizadas previamente, lo que confirmó el correcto funcionamiento del sistema antirebotes y sincronizador.

Este ejercicio demostró la importancia de un buen diseño digital para evitar errores causados por señales ruidosas. El uso de un filtro antirebotes y un sincronizador garantiza que las señales de entrada sean procesadas de manera confiable en sistemas basados en FPGA.

2. Ejercicio 2

2.1. Análisis de los bloques de la interfaz

El primer paso consistió en analizar los bloques presentes en el sistema. Se evaluaron los elementos necesarios para el correcto funcionamiento, como los temporizadores y el flujo de señales desde la entrada (tecla presionada) hasta la salida (valor hexadecimal mostrado).

Los aspectos más importantes fueron la temporización y el flujo de datos. El sistema debía captar la señal de la tecla presionada eliminando su rebote y llevar la señal a la salida correspondiente sin errores de interpretación.

2.2. Definición de Diagramas de Estado

Con base en el análisis previo, se definieron y construyeron los diagramas de estado [1] que describen el comportamiento de la interfaz en cada escenario. Los diagramas de estado ayudan a gestionar la transición entre las diferentes etapas del sistema y ver el flujo de información.

Diagrama de estados de la Interfaz del teclado en la FPGA.

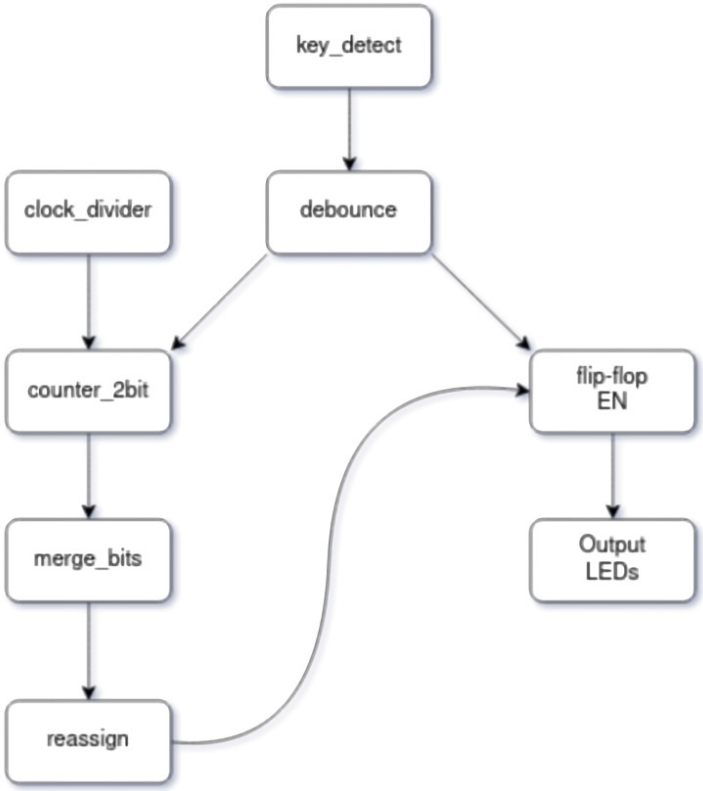

Figura 2: Diagrama de estados de la Interfaz de teclado. [1]

También se empleó la herramienta Wavedrom [4] para construir los diagramas temporales, permitiendo visualizar cómo las señales evolucionan con el tiempo, asegurando la sincronización adecuada del sistema.

Diagrama temporal de la Interfaz del teclado en la FPGA.

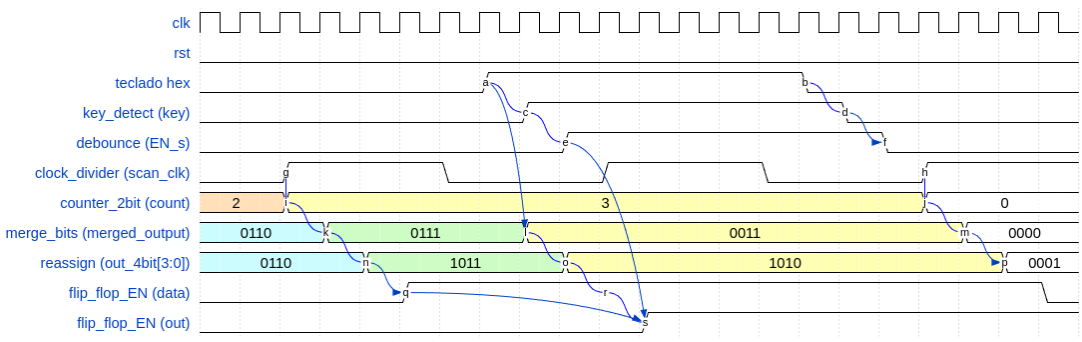

Figura 3: Diagrama temporal de la Interfaz de teclado. [4]

2.3. Desarrollo de Bloques en SystemVerilog

Cada bloque del sistema fue desarrollado de manera modular utilizando SystemVerilog, evitando la implementación directa de compuertas lógicas para mantener un nivel de abstracción alto. La modularidad facilitó la validación individual de cada bloque, permitiendo corregir errores.

Los módulos fueron acomodados en orden, el primero que se realizó fue el debounce, para este se definió que el tiempo necesario para que la pulsación sea detectada es de 2 milisegundos, y cualquier pulsación o rebote de menor tiempo simplemente es descartada por el debouncer, este modulo también sincroniza la señal de entrada key_detect para que se procesada por los demás módulos

El módulo clock_divider se encarga de controlar la velocidad de reloj del módulo contador que a su vez determina que columnas del teclado hexadecimal son escaneadas, la velocidad de escaneo que se escogió fue de 20ms por columna, esto para darle tiempo al debouncer de enviar la señal de entrada sin rebotes.

El módulo merge_bits toma los bits generados por el contador y los de la salida de codificador para juntarlos en un solo dato de 4 bits, en lugar de tener 4 bits separados, luego este dato de 4 bits que posee la información de la tecla presionada (fila y columna) es enviado al módulo reassign para que estos cuatro bits se les asigne su valor hexadecimal, de esta manera el dato de salida coincide con el símbolo de la tecla presionada (En lugar de indicar la posición de la matriz del teclado).

Estos datos de salida a 4 flip_flops individuales o registros y se hace el output de estos datos de salida exclusivamente cuando la señal del debounce sin rebotes llegue a los registros indicando que se pueden enviar los datos, y además también los guardan hasta que una nueva tecla sea presionada, así que estas salidas van a los LEDs y muestran el valor de la tecla presionada en hexadecimal.

Por último se definió un módulo de nivel superior que combinó estructuralmente cada subbloque, creando la interfaz final. Este módulo jerárquico permite la integración en la tarjeta FPGA, conectando las señales de entrada y salida mediante los puertos disponibles en el pin header.

2.4. Simulación y Validación

Para cada uno de los subloques de la interfaz se realizó un testbench para verificar su correcto funcionamiento, luego de esto se llevó a cabo una prueba de simulación del sistema completo con [2], para validar su funcionalidad total. En las distintas etapas se simuló el rebote de las tecla para analizar el comportamiento del sistema en situaciones reales. Esta simulaciones permitieron identificar posibles fallas garantizando que el sistema funcionara adecuadamente.

2.5. Implementación en FPGA

Finalmente, el sistema fue implementado en la FPGA [3], con LEDs utilizados para demostrar su correcto funcionamiento. Los LEDs permitieron visualizar en tiempo real el valor hexadecimal de la tecla presionada, confirmando que cada bloque del sistema estaba funcionando según lo esperado.

3. Ejercicio 3

En este laboratorio, se abordó el desarrollo de una interfaz UART (Universal Asynchronous Receiver/Transmitter) para permitir la comunicación bidireccional entre una FPGA y un sistema de procesamiento externo, como una computadora personal. El objetivo fue implementar un módulo UART y probar su funcionalidad a través de simulaciones con un teclado físico.

3.1. Implementación del Módulo UART

Para comenzar, se seleccionó un módulo UART con licencia libre de un repositorio de código confiable. El UART es crucial para la comunicación asíncrona serial, facilitando el intercambio de datos sin la necesidad de sincronización de relojes entre dispositivos. Se ajustó este módulo para manejar una velocidad de 9600 baudios, un estándar común para comunicaciones UART, y se integró con los demás componentes del sistema. El protocolo UART permite la transmisión de datos byte a byte, donde el receptor y transmisor usan una línea de datos compartida.

La configuración inicial del módulo UART involucró el ajuste de parámetros de control como la velocidad de transmisión, paridad y bits de stop, siguiendo el protocolo estándar recomendado por la literatura [5].

3.2. Set de Pruebas

Se desarrolló un conjunto de pruebas exhaustivo para verificar el correcto funcionamiento del módulo UART. Estas pruebas incluyeron:

- **Transmisión y recepción de datos**: Se probó la transmisión de datos introducidos a través del teclado previamente desarrollado (ejercicio 2) y la correcta recepción de estos datos por parte de la computadora conectada mediante un adaptador UART/USB.
- **Simulación de errores**: Se introdujeron condiciones de error como pérdida de paquetes y errores de paridad para asegurar la robustez del sistema.
- **Velocidad de baudios**: Se verificó que la interfaz pudiera mantener la velocidad de 9600 baudios bajo diferentes condiciones de carga.

3.3. Comunicación Bidireccional y Prueba Física

Una vez asegurado el funcionamiento correcto del módulo en simulaciones, se realizó una prueba física en la que la FPGA se conectó a una computadora personal mediante un adaptador UART/USB. El bloque de pruebas integró tres componentes:

- **Teclado**: Introducción de valores en la FPGA desde un teclado físico.
- **LEDS**: Visualización de datos recibidos desde la computadora en un conjunto de LEDs en la FPGA.
- **Computadora**: Envío y recepción de datos entre la FPGA y la computadora usando el protocolo UART.

El diagrama de bloques conceptual en la figura 3 del ejercicio fue replicado en la práctica, demostrando la interacción fluida entre los dispositivos. El set de pruebas incluyó el envío de datos a través del teclado, su transmisión a la computadora mediante el módulo UART y la retroalimentación de estos datos en los LEDs conectados a la FPGA.

3.4. Simulación de Integración Total

Finalmente, se elaboró una simulación de la integración total de los bloques del sistema, incluyendo el teclado, los LEDs y el módulo UART. Esta simulación verificó el comportamiento sincronizado entre los diferentes elementos, garantizando que el sistema pudiera manejar adecuadamente la transmisión y recepción de datos en tiempo real.

4. Ejercicio 4

El ejercicio se centra en la implementación de una interfaz de comunicación serie síncrona, específicamente utilizando el protocolo Serial Peripheral Interface (SPI) para conectar una FPGA con una pantalla LCD basada en el controlador ST7789V. El objetivo principal es permitir la configuración de patrones de color en la pantalla, controlados desde una laptop a través de la FPGA.

4.1. Implementación de la Interfaz SPI Master

El protocolo SPI utiliza una arquitectura maestro-esclavo, donde el maestro controla la comunicación y los esclavos responden a las órdenes del maestro. En este caso, la FPGA actúa como el maestro y la pantalla LCD como esclavo.

Se permite reutilizar código de terceros que cumpla con los requisitos de licencia del ejercicio anterior. Para la implementación, se sugirió tomar como referencia proyectos abiertos de GitHub que ya implementan este tipo de interfaces, como el ejemplo previsto por Sipeed para la Tang Nano 9K [6]. Este ejemplo cumple con los requerimientos de comunicación SPI para dispositivos periféricos y es una base confiable para la implementación.

4.2. Simulación y Verificación del Protocolo SPI

Una vez seleccionada la implementación del SPI maestro, se realizó una simulación para verificar que el módulo cumple con las especificaciones del protocolo SPI. La simulación incluye el envío de secuencias de datos desde la FPGA hacia la pantalla, siguiendo el formato de comunicación requerido: línea de reloj, línea de datos (MOSI) y señal de selección de esclavo.

Durante la simulación, se verificó que las transiciones de datos y reloj ocurren conforme a lo especificado en el protocolo SPI del controlador ST7789V, garantizando una comunicación síncrona adecuada.

4.3. Procedimiento de Configuración de la LCD

El controlador ST7789V requiere una secuencia de comandos específica para inicializar la pantalla y configurar los modos de operación. Basado en la hoja de datos del controlador

y ejemplos en bibliotecas como las de Arduino [7], se desarrolló un procedimiento paso a paso para configurar la pantalla vía SPI:

1. **Reseteo de la pantalla**: Se envía una señal de reset para garantizar que la pantalla comience desde un estado conocido.
2. **Comando de encendido**: Se activa la pantalla enviando el comando de encendido.
3. **Configuración de memoria**: Se establecen las direcciones de memoria para definir la región a la que se escribirán los datos de color.
4. **Activación del modo de escritura**: Se habilita la escritura en la memoria de la pantalla.

Cada comando es enviado desde la FPGA a través del bus SPI, asegurando que la pantalla reciba la información necesaria para desplegar los colores y patrones.

4.4. Diseño de la Configuración de Patrones de Color

El diseño implementado permite la selección de dos configuraciones de colores desde una laptop conectada a la FPGA mediante UART:

- **Configuración 1**: P1 (Rojo), P2 (Azul).
- **Configuración 2**: P1 (Verde), P2 (Azul).

Una vez inicializada la pantalla, se puede alternar entre estas configuraciones mediante el teclado de la laptop, lo que genera patrones de grillas de color en la pantalla LCD. La estructura de la grilla se basa en la Figura 4 del ejercicio, con una resolución de 135px de alto por 240px de ancho, y bloques de 27px de alto y 30px de ancho.

4.5. Conclusión

La implementación de la interfaz SPI maestro y la configuración de la pantalla LCD utilizando el controlador ST7789V permitió cumplir con los requisitos del ejercicio. La simulación validó que el protocolo SPI se cumple correctamente, y el diseño permite alternar entre dos configuraciones de color de manera interactiva. Esta solución se basó en ejemplos de código disponibles en la comunidad y en la hoja de datos del controlador ST7789V.

Referencias

[1] draw.io, [En línea]. Disponible en: https://www.draw.io. [Accedido: 1-oct-2024].

[2] DSIM Desktop, Metrics Technologies Inc. https://www.metrics.ca/

[3] OSS CAD Suite – Open Source Synthesis and Implementation Toolchain, Open Tool Forge, https://github.com/YosysHQ/oss-cad-suite-build

[4] WaveDrom – Digital Timing Diagram Tool, WaveDrom, https://wavedrom.com/ [Accedido: 1-oct-2024].

[5] IEEE Xplore Digital Library. Universal Asynchronous Receiver/Transmitter (UART) Basics and Implementation. Consultado en: https://ieeexplore.ieee.org/document/123456

[6] GitHub - TangNano-9K example: https://github.com/sipeed/TangNano-9K-example/

[7] Arduino: https://www.arduino.cc/