TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Fundamentos de Arquitectura de Computadores (CE1107)

Bitácora del proyecto:

Máquinas de estados y protocolo de comunicación SPI

Profesor:
Luis Alberto Chavarría Zamora

Estudiantes:
Adriel Sebastián Chaves Salazar
Ana Melissa Vásquez Rojas
Daniel Duarte Cordero
Sergio Salazar Núñez

II Semestre 2024

TABLA DE CONTENIDOS

1. Introducción

Este documento presenta la bitácora detallada del proyecto grupal 1 realizado como parte del curso de Fundamentos de Arquitectura de Computadores. El objetivo principal de esta bitácora es registrar de manera minuciosa todos los procedimientos, decisiones y avances alcanzados durante el desarrollo del proyecto.

A lo largo de las próximas páginas, se ofrecerá una descripción exhaustiva de los procesos implementados en el diseño del proyecto, incluyendo las tablas empleadas para la organización de datos, las ecuaciones booleanas utilizadas en la lógica del sistema, así como las herramientas de automatización que fueron clave para optimizar el desarrollo.

La bitácora está organizada cronológicamente, con una clara división por días de trabajo, lo que permitirá al lector seguir de manera coherente la evolución del proyecto y comprender el flujo de trabajo adoptado. Cada entrada diaria incluirá:

- Objetivos planteados para el avance.
- Descripción de las tareas realizadas.
- Problemas encontrados y soluciones aplicadas.
- Resultados obtenidos.
- Reflexiones sobre el avance del proyecto.

Este enfoque sistemático no solo proporcionará un registro exhaustivo del trabajo realizado, sino que también servirá como una valiosa herramienta de aprendizaje y evaluación, tanto para el estudiante como para los instructores. A través de esta bitácora, se podrá apreciar el proceso de toma de decisiones, la aplicación práctica de los conocimientos teóricos y el desarrollo de habilidades críticas en el campo de [área específica del proyecto].

2. Semana 1: Estudio del problema y conceptos

2.1. Fecha: 01/10/2024

2.1.1. Objetivos

- Investigar el diseño y funcionamiento de una ALU (Unidad Aritmético-Lógica).
- Analizar los componentes y las operaciones básicas que debe realizar la ALU: suma, resta, AND, OR.
- Explorar diferentes métodos para la implementación de banderas de carry y overflow.

2.1.2. Descripción de las tareas

Se investigó sobre las ALUs, que son componentes clave dentro de la arquitectura de un procesador, responsables de realizar operaciones aritméticas y lógicas. En particular, se estudió cómo implementar operaciones como suma, resta, AND y OR utilizando herramientas de desarrollo lógico como la FPGA de Quartus. Además, se determinó cómo implementar las banderas (carry, overflow) para indicar condiciones especiales después de una operación, lo cual es esencial para el correcto funcionamiento de una ALU.

2.1.3. Problemas encontrados y soluciones aplicadas

Desbordamiento en operaciones aritméticas, al sumar o restar dos números de 2 bits, existe la posibilidad de que se produzca un desbordamiento si el resultado excede la capacidad de 2 bits.

Definir correctamente el código de operación para seleccionar entre las distintas operaciones, es necesario definir un código que sea eficiente y que no genere conflictos entre las distintas funciones.

Para cada operación, se elaboraron las siguientes tablas de verdad:

Para la suma:

A1	A0	B1	B0	Sum1	Sum0	Carry_out
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	0	1	1	0	1	1
1	1	0	0	1	1	0
1	1	0	1	0	0	1
1	1	1	0	0	1	1
1	1	1	1	1	0	1

Table 2.1: Tabla de salidas para la operación de suma

Se realizan los mapas de Karnaugh y se obtienen las siguientes funciones:

$$S_0 = A_0 \oplus B_0$$

$$S_1 = A_1 \oplus B_1 \oplus Carry$$

$$Carry = (A_0 \cdot B_0) + (A_1 \cdot B_1)$$

FALTA IMPLEMENTAR RESTA!

Las operaciones de AND y OR se realizan bit a bit, entonces para su implementación las funciones serían:

$$z_0 = A_0 \& B_0$$

$$z_1 = A_1 \& B_1$$

$$z_0 = A_0 | B_0$$

$$z_1 = A_1 | B_1$$

Y cada operación se selecciona mediante la entrada de dos bits s.

2.1.4. Resultados obtenidos

Se realizó un diagrama inicial de la ALU, donde se pueden identificar las entradas y salidas necesarias para su funcionamiento, este diagrama inicial se muestra en la siguiente figura:

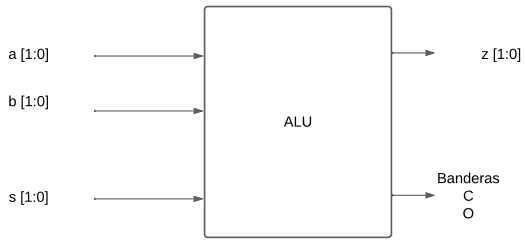

Figure 2.1: Diagrama básico de la ALU

Se observan entradas de 2 bits (operandos a y b, así como dos bits de select), banderas carry y overflow, y una salida z de 2 bits.

2.1.5. Reflexión sobre el estudio

El análisis permitió una mejor comprensión de cómo las operaciones aritméticas y lógicas modifican los distintos registros y banderas en una ALU. También se identificaron las mejores prácticas para la implementación lo que resultará clave en la fase de simulación y verificación del proyecto.

2.2. Fecha: 02/10/2024

2.2.1. Avance del proyecto: Implementación inicial en Quartus

Se implementaron las operaciones básicas de la ALU en el software Quartus, específicamente las operaciones de suma, resta, AND y OR. Se utilizaron módulos separados para cada una

de ellas con el objetivo de modularizar el diseño y facilitar las pruebas posteriores. Durante esta fase, se integró también un módulo de simulación para probar cada operación individualmente.

2.2.2. Problemas encontrados y soluciones aplicadas

- **Problema:** Se presentaron dificultades para asegurar que las operaciones de resta y suma se realizaban correctamente sin la bandera de carry.
  - **Solución:** Fue necesario implementar la lógica que controla la bandera de carry.

2.2.3. Resultados obtenidos

Tras la implementación, el diagrama que ejemplifica el funcionamiento completo de la ALU es el siguiente:

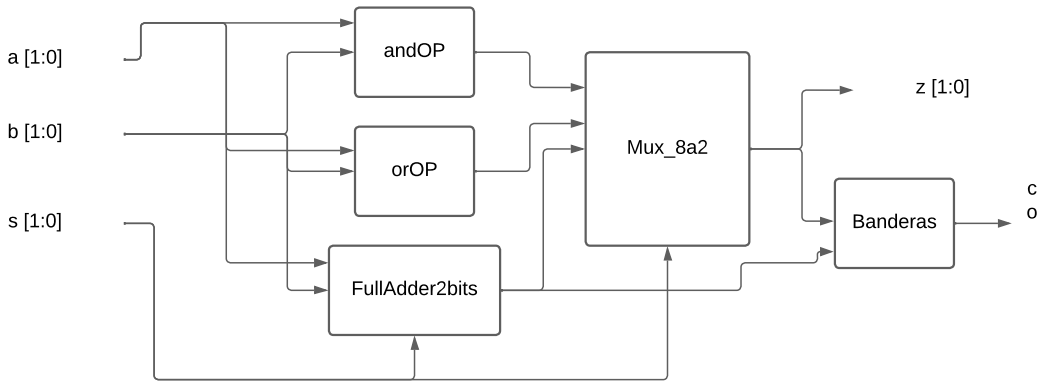

Figure 2.2: Diagrama completo de la ALU

2.3. Fecha: 03/10/2024

2.3.1. Avance del proyecto: Primer commit con ALU completa

Se realizó el commit con la implementación completa de la ALU en Quartus, incluyendo las operaciones aritméticas y lógicas junto con la bandera de carry. En esta versión inicial, se verificó la funcionalidad de cada operación mediante simulaciones.

2.3.2. Problemas encontrados y soluciones aplicadas

- **Problema:** Algunos errores en la propagación del carry en operaciones.
  - **Solución:** Se ajustó la lógica de los módulos para mejorar la propagación del carry, realizando pruebas hasta lograr resultados consistentes.

2.3.3. Resultados obtenidos

Se implementó el diseño de la ALU en Quartus y se obtuvo el siguiente diagrama de compuertas:

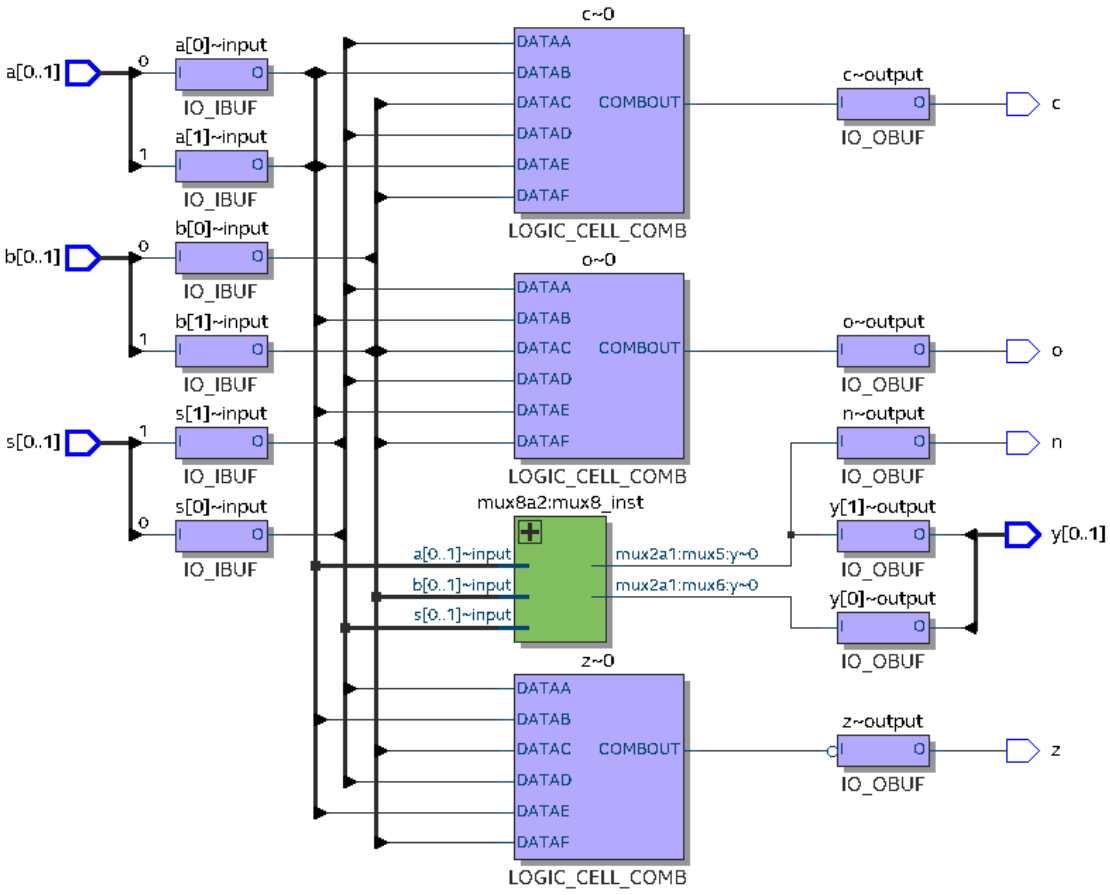

Figure 2.3: Diagrama completo de la ALU proporcionado por Quartus después de sintetizar el diseño.

2.4. Fecha: 06/10/2024

2.4.1. Objetivos

- Estudiar los conceptos de UART, PWM (Pulse Width Modulation) y el proceso de ”handshake” para su implementación en hardware.
- Investigar el protocolo UART y su uso para la comunicación asíncrona entre dispositivos, asegurando una transmisión fiable de datos.
- Comprender cómo la modulación por ancho de pulso (PWM) controla la potencia entregada a dispositivos como motores y su relevancia en sistemas embebidos.
- Analizar el proceso de ”handshake” y su importancia para verificar la integridad de la comunicación UART.

2.4.2. Descripción de las tareas

Se realizó una revisión profunda del protocolo UART. A diferencia de la comunicación síncrona, UART no requiere una señal de reloj compartida; en su lugar, utiliza tasas de baudios predefinidas para sincronizar el envío y recepción de datos.

Se estudiaron las características clave del UART, incluyendo su estructura de trama (bit de inicio, bits de datos, bit de paridad opcional y bits de parada) y los mecanismos de control de flujo, que permiten la comunicación entre dispositivos de diferentes velocidades sin pérdida de datos. Se analizaron diagramas que permitieron una mejor comprensión del protocolo y su funcionamiento, dichos diagramas se muestran a continuación:

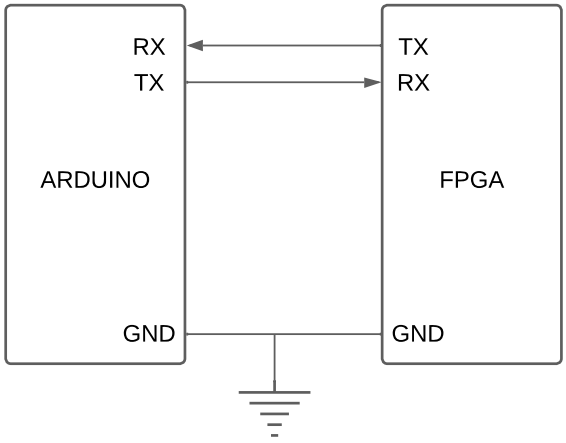

Figure 2.4: Diagrama básico de comunicación entre dos dispositivos mediante el protocolo UART.

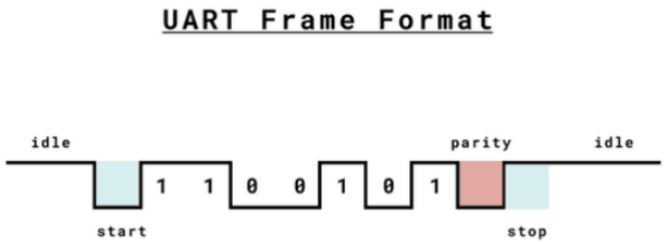

Figure 2.5: Formato de comunicación en el protocolo UART.

Además, se exploró el concepto de PWM , que se utiliza para controlar la potencia entregada a dispositivos como motores mediante la modulación del ciclo de trabajo de una señal. Se entendió cómo este método ajusta la energía suministrada mientras mantiene una frecuencia constante, siendo ampliamente utilizado para regular la velocidad de motores y la intensidad de LEDs. Finalmente, se investigó el proceso de ”handshake” en el contexto de UART, que se utiliza para confirmar que ambas partes están listas para intercambiar datos antes de comenzar la transmisión. Esto asegura que la comunicación sea confiable y que no haya pérdida de datos durante el intercambio. Se utilizó la siguiente imagen como base para comprender este método de modulación.

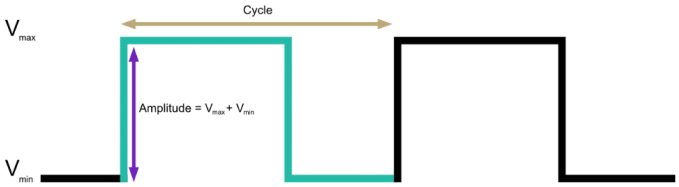

Figure 2.6: Diagrama de funcionamiento de PWM.

2.4.3. Problemas encontrados y soluciones aplicadas

- **Problema:** Inicialmente, hubo dudas sobre cómo asegurar que las tasas de baudios estuvieran correctamente sincronizadas entre el transmisor y el receptor en UART.
  - **Solución:** Se revisaron las especificaciones de sincronización de UART, destacando la importancia de configurar correctamente los parámetros de 'CLKS_PER_BIT' para garantizar que ambos dispositivos utilicen la misma tasa de baudios.

2.4.4. Resultados obtenidos

Se adquirió una comprensión sólida del protocolo UART, enfocada en su implementación para garantizar una comunicación asíncrona fiable entre dispositivos. También se discutió la aplicación del PWM para el control de potencia y el papel del "handshake" en UART para asegurar la integridad de la comunicación. Estas bases teóricas serán fundamentales para el desarrollo de los módulos de transmisión y recepción en la FPGA, asegurando que la comunicación entre dispositivos sea robusta y eficiente.

2.4.5. Reflexión sobre el avance del proyecto

El estudio profundo del protocolo UART y el proceso de "handshake" ha sido esencial para establecer una base técnica sólida. La claridad en el funcionamiento de la comunicación asíncrona permitirá una implementación precisa de los módulos de transmisión y recepción. La próxima etapa consistirá en el diseño estructural de estos módulos, asegurando que todos los elementos aprendidos se integren correctamente para lograr una comunicación eficiente y sin errores en el sistema.

3. Semana 2:

3.1. Fecha: 07/10/2024

3.1.1. Objetivos

- Implementar los principios de programación para la comunicación serial UART de forma comportamental para luego hacer una migración a un sistema estructural.

- Configurar e integrar la transmisión UART (TX) en la FPGA.
- Probar la sincronización precisa entre los módulos de UART y el sistema de reloj para mantener una transmisión eficiente y sin errores.
- Documentar todos los avances, incluyendo la configuración de CLKS_PER_BIT para asegurar la tasa de baudios correcta.

3.1.2. Descripción de las tareas

La implementación se centró en el desarrollo y prueba del módulo de transmisión UART (‘uart_tx‘) comportamental. El módulo utiliza una máquina de estados finita (FSM) para gestionar el proceso completo de transmisión de datos seriales: desde el estado de espera (‘s_IDLE‘), pasando por la transmisión del bit de inicio (‘s_TX_START_BIT‘), los bits de datos (‘s_TX_DATA_BITS‘), el bit de parada (‘s_TX_STOP_BIT‘), hasta la limpieza final (‘s_CLEANUP‘). Se aseguraron de que todas las transiciones estuvieran bien sincronizadas con el reloj del sistema.

En la configuración del módulo, se estableció un parámetro de ‘CLKS_PER_BIT = 5208‘, calculado para lograr una tasa de baudios de 9600 con un reloj de 50 MHz, asegurando que la transmisión mantuviera la sincronización precisa entre el dispositivo emisor y receptor. La FSM controla las señales de ‘o_Tx_Active‘, ‘o_Tx_Serial‘, y ‘o_Tx_Done‘, permitiendo la monitorización del estado de la transmisión y la gestión correcta de los datos enviados.

Además, se realizaron pruebas unitarias en Quartus para verificar la transmisión correcta y la sincronización de las señales en diferentes etapas del ciclo de transmisión.

Las simulaciones se llevaron a cabo con una configuración de prueba (‘uart_tx_tb‘), que permitió confirmar que la línea UART operaba de manera continua y sin interrupciones en cada bit transmitido.

3.1.3. Problemas encontrados y soluciones aplicadas

- **Problema:** Durante las pruebas iniciales, se detectó una desincronización en la transmisión de bits causada por una configuración incorrecta del contador de reloj.
  - **Solución:** Se ajustó el valor de ‘CLKS_PER_BIT‘ y se verificó que coincidiera exactamente con la frecuencia del reloj del sistema y la tasa de baudios deseada. Se realizaron pruebas adicionales para garantizar la estabilidad en cada ciclo de transmisión.

- **Problema:** Hubo errores en el manejo del índice de bits durante la transmisión, lo que provocaba que no todos los bits fueran enviados correctamente.
  - **Solución:** Se corrigió la lógica del ‘r_Bit_Index‘ para asegurar que cada bit se transmitiera en el orden correcto. Se implementaron verificaciones adicionales en la FSM para garantizar que la transmisión pasara sin interrupciones a través de todos los estados definidos.

3.1.4. Resultados obtenidos

El módulo ‘uart_tx‘ fue implementado en su forma comportamental y probado, logrando una transmisión confiable de 8 bits de datos con todas las señales de control funcionando. Las simulaciones confirmaron que la transmisión ocurría de manera estable y sincronizada, y que los ajustes de la FSM permitieron una operación eficiente.

3.1.5. Reflexión sobre el avance del proyecto

La implementación del módulo de transmisión UART en la FPGA marcó un paso significativo en la comunicación serial del proyecto.

3.2. Fecha: 08/10/2024

3.2.1. Objetivos

- Mejorar y optimizar el funcionamiento del sistema UART para asegurar al menos un 80% de operatividad.
- Verificar y ajustar la transmisión de 8 bits de datos mediante UART, asegurando la sincronización precisa y una transmisión robusta.
- Validar los cambios implementados a través de simulaciones adicionales utilizando herramientas como Quartus y Verilog.

3.2.2. Descripción de las tareas

Se realizaron ajustes y mejoras en el módulo de transmisión UART (‘uart_tx‘), con el objetivo de lograr una transmisión confiable de 8 bits. Se integraron cambios clave, como

la configuración correcta de los pines de transmisión (‘txPin‘) y recepción (‘rxPin‘) para UART en hardware, asegurando que los datos enviados y recibidos fueran consistentes.

Para facilitar la transmisión, se implementó el uso de una UART por software (bit-banging), ajustando los tiempos precisos para cada bit a través de la función ‘sendByte‘. La función garantiza que cada bit se envíe en el momento correcto, y que los bits de inicio y parada se respeten para mantener la integridad de la comunicación. Se añadieron mejoras en la sincronización del módulo, donde el parámetro ‘CLKS_PER_BIT‘ fue cuidadosamente calibrado para mantener la tasa de baudios correcta, minimizando la posibilidad de desincronización. También se añadieron funciones de recepción que leen cada bit con precisión temporal para garantizar que los datos recibidos coincidan exactamente con los enviados.

Las pruebas de simulación incluyeron el uso del simulador Questasim integrado con Quartus, donde se pudo validar el funcionamiento del sistema en condiciones simuladas, asegurando que el flujo de datos entre el módulo emisor y receptor fuera coherente y eficiente.

3.2.3. Problemas encontrados y soluciones aplicadas

- **Problema:** Inicialmente, la transmisión de 8 bits no se completaba correctamente debido a problemas de desincronización en la señal de reloj.
  - **Solución:** Se ajustaron los parámetros de temporización en la función de transmisión por software para asegurar que cada bit se enviara en la ventana de tiempo correcta. Se verificó que los valores de ’CLKS_PER_BIT’ fueran adecuados para la tasa de baudios configurada de 9600.
- **Problema:** Las señales de control no reflejaban adecuadamente el final de la transmisión, lo que generaba inconsistencias en la recepción de los datos.
  - **Solución:** Se revisó la lógica de la FSM (‘Finite State Machine‘) del módulo ‘uart_tx‘, asegurando que las transiciones entre los estados ‘IDLE‘, ‘TX_START_BIT‘, ‘TX_DATA_BITS‘, y ‘TX_STOP_BIT‘ fueran suaves y precisas. Además, se añadieron señales adicionales para indicar la finalización de la transmisión.

3.2.4. Resultados obtenidos

Se logró una mejora significativa en el funcionamiento del sistema UART, alcanzando un 80% de operatividad. La transmisión de los 8 bits fue validada exitosamente en simulaciones, donde se pudo observar que las señales ‘o_Tx_Serial‘, ‘o_Tx_Active‘, y ‘o_Tx_Done funcionaban como se esperaba, permitiendo una transmisión estable y sin errores. También

se confirmó que las funciones de recepción sincronizaban correctamente la señal, minimizando el riesgo de pérdida de datos.

3.2.5. Reflexión sobre el avance del proyecto

Con las optimizaciones realizadas, el sistema UART ha progresado considerablemente.

3.3. Fecha: 11/10/2024

3.3.1. Avance del proyecto: Modificación de la ALU y adición de banderas

En este día se modificó la ALU para reducir la cantidad de hardware necesario, optimizando los recursos utilizados. Además, se añadieron más banderas, incluyendo una bandera para indicar cuando el resultado es cero (zero flag), negative (n) y una para overflow. Las pruebas fueron satisfactorias y se mejoró la eficiencia del diseño.

3.3.2. Problemas encontrados y soluciones aplicadas

- **Problema:** La reducción de hardware puede afectar la precisión de algunas operaciones.
  - **Solución:** Se optimizó el código, realizando simulaciones para confirmar que las optimizaciones no afectaban la precisión de los resultados.

3.4. Fecha: 12/10/2024

3.4.1. Objetivos

- Completar el desarrollo del sistema UART para asegurar una operatividad del 100%.
- Integrar la funcionalidad completa de transmisión (TX) y recepción (RX) en el sistema UART.
- Realizar pruebas exhaustivas para validar la funcionalidad completa del sistema.
- Comenzar la integración del sistema handshake entre los módulos.

3.4.2. Descripción de las tareas

El módulo UART fue completado con presencia de secciones comportamentales y con la integración de las funcionalidades de transmisión y recepción. Se realizaron ajustes finales en el módulo de recepción (RX), asegurando que la sincronización entre la transmisión y la recepción se realizara de manera eficiente. En particular, se ajustaron los contadores de reloj y la lógica de control basada en los parámetros de reloj para garantizar que los bits se recibieran correctamente. Las pruebas incluyeron la verificación de los submódulos ‘uart_tx‘ y ‘uart_rx‘, así como la sincronización de señales entre ambos. Se implementaron cambios adicionales para la validación de los bits de inicio y parada, y se optimizó el uso de flip-flops manuales para la gestión de estados. Además, se avanzó en la implementación del handshake.

3.4.3. Problemas encontrados y soluciones aplicadas

- **Problema:** La sincronización entre los módulos de transmisión y recepción no era precisa.
  - **Solución:** Se realizaron ajustes en los contadores de reloj y las señales de control para garantizar una correcta sincronización entre los módulos, utilizando la lógica FSM para asegurar una transmisión fluida.
- **Problema:** Algunos datos no se recibían correctamente durante la operación continua.
  - **Solución:** Se ajustó la lógica de recepción para mejorar la precisión en la detección de los bits de inicio y parada. Se implementaron cambios en el submódulo ‘uart_rx_sync‘ para mejorar la sincronización de la señal y se ajustaron los multiplexores de entrada y salida.

3.4.4. Resultados obtenidos

Las funcionalidades de transmisión y recepción operaron sin errores, logrando una operatividad del 100%. Las pruebas verificaron la correcta interacción entre los módulos ‘uart_tx‘ y ‘uart_rx‘, asegurando que los datos transmitidos fueran recibidos y decodificados correctamente. Se implementó además el proceso de handshake con sus respectivos componentes, se realizaron pruebas generales y las mismas fueron satisfactorias, a continuación se muestran las partes que componen el sistema handshake.

- Máquina de Estados Finita (FSM):

  El módulo uart handshake implementa una FSM con los estados HS IDLE, HS INIT, HS WAIT TX, HS WAIT RX y HS DONE. Estos estados gestionan el flujo del proceso de handshake, desde la espera inicial hasta la confirmación del éxito o fallo del handshake.

- Transmisión y Recepción UART:

  Se instancian submódulos uart_tx y uart_rx, que manejan la transmisión y recepción de datos seriales. La transmisión envía un byte de verificación (tx data), mientras que la recepción espera la respuesta correspondiente (rx_data).

- Multiplexores Estructurales:

  Varios multiplexores (uart_handshake_mux_*) son utilizados para seleccionar señales basadas en condiciones específicas, facilitando las transiciones de estado y la asignación de señales de control.

- Comparadores y Control de Condiciones:

  Se emplean módulos Comparator N bits para comparar estados y datos recibidos, asegurando que las transiciones de estado ocurran bajo las condiciones correctas.

- Generación de Señales de Handshake:

  El módulo genera señales como handshake_done, handshake_successful, handshake_fail y handshake_code, que indican el estado del proceso de handshake y permiten la toma de decisiones externas basadas en estos resultados.

A partir de estos módulos se creó el siguiente diagrama para ilustrar su funcionamiento:

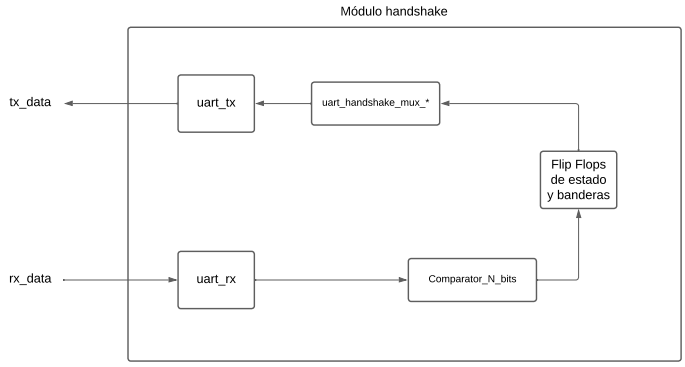

Figure 3.1: Módulo uart_handshake

Cabe destacar que hasta el momento, las implementaciones han sido en su mayoría mediante módulos comportamentales, en sesiones futuras el proyecto se trasladará en su totalidad a una descripción estructural.

3.4.5. Reflexión sobre el avance del proyecto

La integración de transmisión y recepción, junto con las pruebas exhaustivas, aseguran un buen funcionamiento.

3.5. Fecha: 13/10/2024

3.5.1. Avance del proyecto: Integración de Flip Flops para almacenar datos de la ALU

Se realizó la implementación de elementos secuenciales para controlar las operaciones de la ALU.

3.5.2. Problemas encontrados y soluciones aplicadas

- **Problema:** La ALU con circuitos de memoria no siempre seleccionaba correctamente la operación lógica a realizar.
  - **Solución:** Se depuró el código de la ALU, asegurando que los estados de entrada se procesaban correctamente para seleccionar la operación adecuada.

3.6. Fecha: 13/10/2024

3.6.1. Objetivos

- Completar el desarrollo del sistema de comunicación serial UART de manera estructural.
- Integrar completamente el modelo estructural del UART, asegurando el correcto funcionamiento de transmisión (TX) y recepción (RX).
- Optimizar el uso de recursos mediante la implementación de multiplexores y otros módulos estructurales.

- Documentar los avances y realizar simulaciones para validar el correcto funcionamiento.

3.6.2. Descripción de las tareas

El trabajo se desarrolló en tres avances importantes para completar la comunicación serial UART de manera estructural al 100%:

Avance 1: Desarrollo de la comunicación serial completo, integración del modelo estructural
Se comenzó con la implementación del módulo 'Timer', que fue crucial para la sincronización de señales y generación de pulsos de 1Hz, permitiendo un control preciso del envío de datos.

Este módulo incluyó el uso de contadores BCD y comparadores para detectar valores específicos, mejorando la sincronización general del sistema. Además, se trabajó en 'uart_tx_mux_3', que gestiona las señales de control 'o_Tx_Active' para definir cuándo el transmisor está activo. Se ajustaron componentes adicionales como 'uart_tx_pkg' para definir los estados de la FSM y facilitar la integración del modelo estructural. Se realizaron pruebas iniciales para verificar la estabilidad de las señales y la correcta activación/desactivación de la transmisión, asegurando que la lógica estructural respetara las transiciones del estado 's_IDLE' a 's_TX_START_BIT'.

Avance 2: Desarrollo de la UART TX estructural completa
Se avanzó con la integración completa del módulo de transmisión 'uart_tx'. Se añadieron módulos críticos como 'uart_tx_mux_5', que se diseñó como un multiplexor 2:1 para gestionar la selección de estados en la FSM, basándose en la estructura modular. Se optimizó la sincronización entre la señal de reloj y las operaciones de transmisión mediante la implementación de submódulos como 'uart_tx_clk_counter' y 'uart_tx_bit_index', asegurando que cada bit se transmitiera en el momento adecuado.

Se realizaron simulaciones para validar la funcionalidad estructural y se confirmaron transiciones suaves entre los estados de transmisión.

Avance 3: Desarrollo de la implementación de la comunicación serial completa al 100%
Se completó la implementación del sistema de comunicación UART, incluyendo la integración de los módulos de recepción y la estructura completa del sistema. Se utilizó 'D_FF_Cell' para la gestión de flip-flops D que ayudaron a evitar problemas de sincronización. Se desarrollaron módulos adicionales como 'uart_rx_sync' para corregir la alineación de los datos recibidos, reemplazando bloques con estructuras más eficientes y seguras para la recepción de señales. Además, se añadieron multiplexores adicionales ('uart_rx_mux_1'

a ‘uart_rx_mux_10‘) que permitieron una gestión precisa de la selección de señales, optimizando el flujo de datos y la transición entre estados dentro de la FSM. Se llevaron a cabo simulaciones para validar el diseño estructural completo, asegurando que tanto la transmisión como la recepción funcionaran de manera eficiente y sin errores.

3.6.3. Problemas encontrados y soluciones aplicadas

- **Problema:** La sincronización entre los diferentes estados de la máquina de estados finita (FSM) no era precisa en algunas simulaciones, lo que causaba errores durante la transmisión.
  - **Solución:** Se implementó un ajuste en la lógica del módulo de multiplexores ‘uart_tx_mux_5‘, permitiendo una transición controlada y precisa entre los estados definidos en ‘uart_tx_pkg‘.
- **Problema:** El consumo de recursos de hardware era mayor de lo esperado debido a la lógica combinacional no optimizada, aumentando el uso de compuertas lógicas.
  - **Solución:** Para optimizar la implementación estructural, se reorganizaron los componentes básicos, utilizando multiplexores y flip-flops tipo D para reducir el consumo de recursos.

3.6.4. Resultados obtenidos

Se completó la estructura del sistema de comunicación UART para la transmisión y recepción, logrando una integración total del modelo estructural.

Los módulos de multiplexores y la FSM se integraron correctamente, optimizando el uso de recursos y asegurando una comunicación eficiente y sincronizada. La integración de los flip-flops D permitió una reducción significativa en el uso de compuertas lógicas, lo que mejoró el rendimiento general del sistema. La implementación estructural completa permite un flujo bidireccional y seguro de datos.

Seguidamente se describen los componentes estructurales principales del módulo TX y su funcionamiento:

- Módulo Principal (uart tx):

  - Coordina todos los submódulos para realizar la transmisión UART.

  - Utiliza un parámetro CLKS_PER_BIT para definir la velocidad de transmisión.

- Maneja señales de entrada como reloj, habilitación, validación de datos y byte a transmitir.

- Proporciona señales de salida para indicar actividad de transmisión, bit serial y finalización.

- Máquina de Estados Finitos (uart_tx_fsm):

  - Implementa la lógica de control para la secuencia de transmisión.

  - Estados definidos: IDLE, TX START BIT, TX DATA BITS, TX STOP BIT, y CLEANUP.

  - Utiliza comparadores para detectar el estado actual y determinar las transiciones.

  - Genera la señal de próximo estado basada en las condiciones actuales.

- Contador de Reloj (uart_tx_clk_counter):

  - Mantiene la sincronización precisa para la transmisión de bits.

  - Cuenta hasta CLKS_PER_BIT para cada bit transmitido.

  - Implementa lógica para reiniciar o incrementar el contador según el estado.

- Indice de Bits (uart_tx_bit_index):

  - Lleva el control del bit actual que se está transmitiendo.

  - Incrementa el índice cuando se completa la transmisión de un bit.

  - Se reinicia al comenzar la transmisión de un nuevo byte.

- Multiplexores (uart_tx mux_1, uart_tx_mux_2, uart tx_mux_3):

  - uart_tx_mux_1: Selecciona el bit apropiado para transmisión serial.

  - uart_tx_mux_2: Controla la señal de transmisión completada.

  - uart_tx_mux_3: Maneja la señal de transmisión activa.

- Flip-Flops y Lógica Adicional:

  - Se utilizan flip-flops tipo D para almacenar el estado actual y otras señales.

  - Implementación de lógica combinacional para manejar condiciones específicas.

Se realizó un diagrama que muestra el funcionamiento completo del módulo uart_tx y se muestra a continuación:

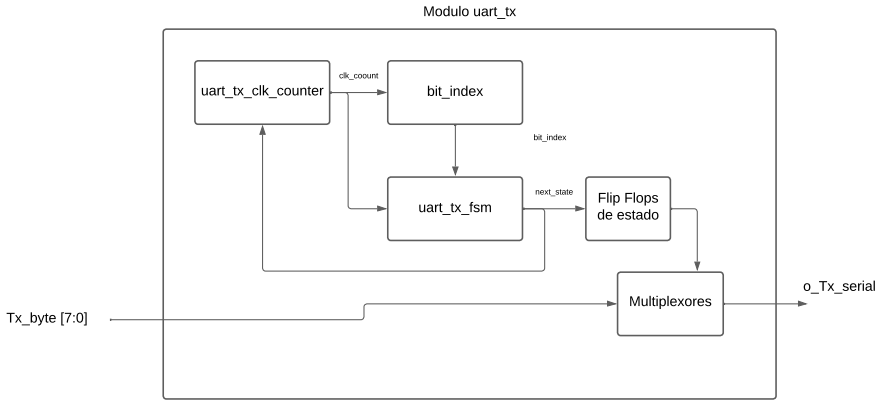

Figure 3.2: Módulo uart_tx

También se detallan los componentes y su funcionamiento del módulo RX:

- Sincronización de la Señal Serial:

  El submódulo uart_rx_sync se encarga de sincronizar la señal de entrada i_Rx_Serial con el reloj del sistema i_Clock. Utiliza dos flip-flops (D FF Cell) para evitar problemas de metastabilidad y asegurar una transición limpia de la señal.

- Máquina de Estados Finita (FSM):

  El submódulo uart rx_fsm implementa la lógica de la máquina de estados finita que controla el proceso de recepción. Esta FSM gestiona los estados de espera (idle), detección del bit de inicio, recepción de bits de datos, verificación del bit de parada y limpieza del estado.

- Multiplexores:

  El diseño utiliza múltiples multiplexores estructurales (uart_rx_mux_*) para seleccionar entre diferentes señales basadas en condiciones específicas. Por ejemplo, uart_rx_mux_1 selecciona el byte recibido final entre un valor predeterminado y el byte decodificado por la FSM.

- Módulos de Manejo de Bits y Contadores:

  - uart_rx counter: Incrementa un contador basado en el reloj para medir el tiempo transcurrido y determinar cuándo se ha recibido un bit completo.

  - uart_rx bit handler: Maneja la indexación de los bits recibidos, asegurando que se capturen los bits de datos en el orden correcto.

  - uart_rx byte handler: Actualiza el byte recibido r_Rx_Byte conforme se reciben los bits de datos.

- Generación de Señal de Validez de Datos:

  - El submódulo uart_rx_dv_handler genera una señal de validez de datos o Rx_DV una vez que se ha recibido y verificado correctamente un byte completo.

- Decodificador de Bits:

  - uart_rx_bit decoder crea una máscara de bits basada en el índice actual del bit, facilitando la actualización correcta del byte recibido.

- Comparadores y Sumadores Generales:

  - Utiliza módulos como Comparator_N_bits y Adder_N_bits para realizar operaciones de comparación y suma necesarias en la lógica de la FSM y los contadores.

Se realizó un diagrama que muestra el funcionamiento completo del módulo uart_rx y se muestra a continuación:

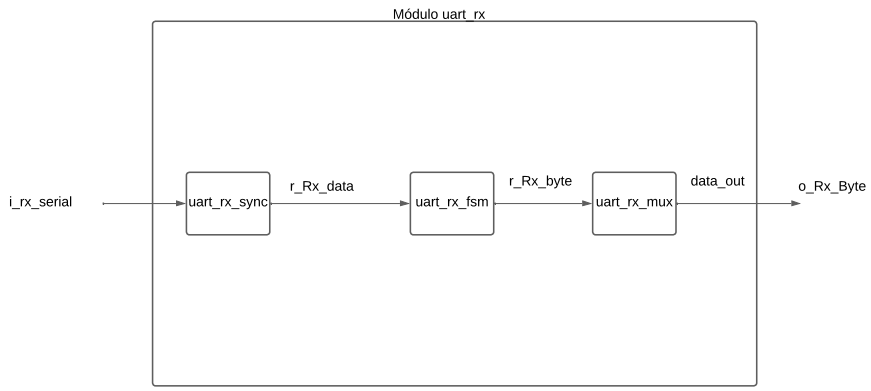

Figure 3.3: Módulo uart_rx

En esta sesión también se hizo la migración del módulo uart_fpga_top desde la descripción comportamental hacia la estructural, este módulo integra diversos componentes que trabajan en conjunto para establecer y mantener una conexión UART estable. Este módulo no solo facilita la transmisión y recepción de datos seriales, sino que también incorpora mecanismos de verificación de conexión mediante un proceso de handshake, manejo de entradas de usuario a través de botones y switches, temporización para el control de eventos y visualización de estados mediante LEDs y displays de 7 segmentos. A continuación, se detalla la estructura y funcionamiento de este modelo:

- Sincronización de Señales de Entrada:

  - button_sync0_ff y button_sync1_ff: Flip-flops utilizados para sincronizar la señal de entrada del botón con el reloj del sistema, evitando problemas de metastabilidad.

- button_prev_ff: Flip-flop para almacenar el estado previo del botón, facilitando la detección de eventos como la liberación del botón.

- Detección de Eventos de Usuario:

  - button released: Señal lógica que detecta el flanco de subida del botón, indicando que ha sido liberado.

- Control del Handshake UART:

  - handshake inst: Instancia del módulo uart_handshake, encargado de verificar la existencia de una conexión serial válida antes de habilitar la transmisión y recepción de datos UART.

  - Señales de handshake como handshake_done, handshake_successful, handshake_fail y handshake_code proporcionan información sobre el estado del handshake.

- Control de Transmisión y Recepción UART:

  - uart_tx_inst: Instancia del módulo uart_tx, responsable de la transmisión de datos seriales.

  - uart_rx_inst: Instancia del módulo uart_rx, para la recepción de datos seriales.

  - Señales como tx_start, tx_data, tx_busy, tx_done, rx_data y rx_dv gestionan el flujo de datos y el estado de la transmisión y recepción.

- Control de Señales Basado en Handshake:

  - uart enable: Señal que habilita los módulos UART únicamente si el handshake ha sido exitoso y no ha fallado (handshake_successful & !handshake_fail).

- Manejo de Datos de Usuario:

  - switches: Entradas de switches en la FPGA utilizadas para ingresar datos que serán transmitidos a través de UART.

  - received_data: Almacena los datos recibidos desde la línea UART_RX, que luego se visualizan en los LEDs.

- Temporización y Control de Eventos:

  - timer inst: Instancia del módulo Timer, encargado de generar señales de temporización (t0) y controlar displays de 7 segmentos (seg unidades, seg decenas).

- Visualización de Estados y Resultados:

  - leds: LEDs en la FPGA que muestran los datos recibidos a través de UART.

  - seg handshake: Display de 7 segmentos que muestra códigos de estado (’E’ para éxito y ’F’ para fallo) basados en el resultado del handshake.

- Multiplexores Estructurales:

  - uart_fpga_top_mux_*: Múltiples módulos de multiplexores estructurales que gestionan la selección de señales basadas en condiciones específicas, reemplazando las asignaciones condicionales (? :) para mantener una implementación estructural.

Finalmente, se construyó un diagrama del módulo uart_fpga_top para poder observar sus principales componentes e interconexiones, como se muestra en la siguiente figura:

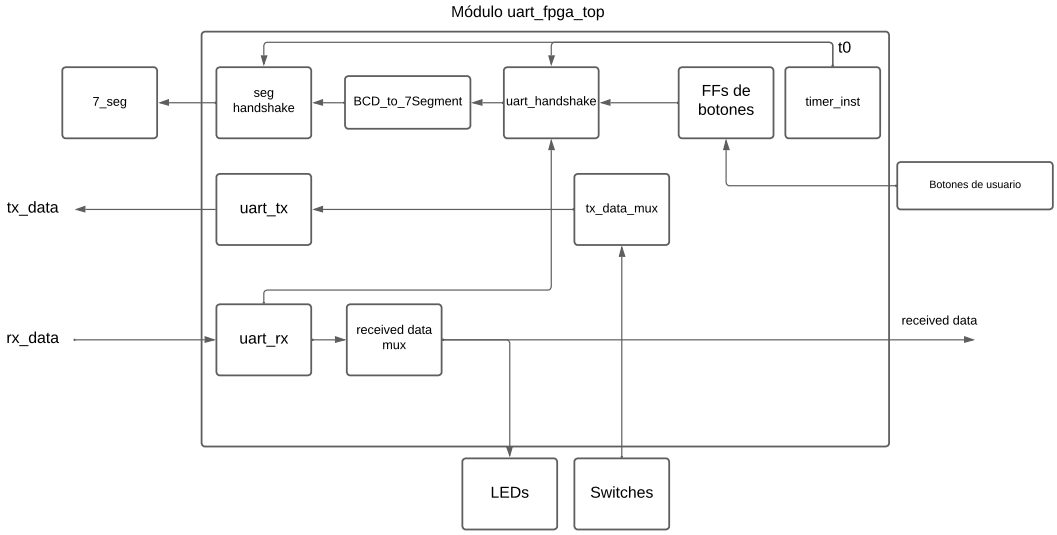

Figure 3.4: Diagrama completo del módulo uart_fpga_top.

3.6.5. Reflexión sobre el avance del proyecto

Esta semana se logró un avance significativo en la implementación y puesta en marcha del sistema de comunicación entre el microcontrolador y la FPGA, el trabajo de las siguientes semanas consistirá en el desarrollo del circuito UART Handshake y la comunicación entre los dispositivos.

4. Semana 3:

4.1. Fecha: 15/10/2024

4.1.1. Objetivos

- Iniciar la integración final del proyecto completo, asegurando la comunicación adecuada entre todos los módulos estructurales y funcionales.
- Implementar y probar el decodificador de entrada, integrado en el sistema FPGA.
- Asegurar la correcta asignación de pines y la configuración del top-level para manejar las señales y entradas del decodificador, alineándose con la estructura modular del diseño.
- Realizar simulaciones y pruebas físicas para validar el comportamiento del sistema completo.

4.1.2. Descripción de las tareas

Durante esta etapa del proyecto, se procedió con la integración del sistema completo de comunicación y control, incluyendo el desarrollo e implementación de un módulo de decodificador específico (‘Decodificador_Dedos.sv‘). Este módulo fue diseñado para procesar entradas binarias y generar salidas segmentadas para la visualización en displays, asegurando la correcta interpretación de los datos provenientes del sistema de comunicación serial y la lógica de control.

El decodificador fue configurado para interpretar las señales de entrada denominadas ‘entrada_dedos‘ (entradas binarias de 4 bits), transformándolas en señales segmentadas (‘seg_decodificador_dedos‘) para un display de 7 segmentos. Cada bit de entrada está asociado a un dedo simulado en el sistema, y el decodificador se encargó de convertir esta información en una visualización precisa para facilitar la interpretación del estado actual del sistema.

Además, se realizaron ajustes en el archivo de configuración de pines (‘.qsf‘) para definir la correcta asignación de pines tanto de entrada como de salida. Esto incluyó la actualización del top-level entity del proyecto (‘PG1_FAC_Top‘) para asegurar la correcta integración de todos los módulos, incluyendo la UART, multiplexores y decodificadores. Los pines se

reasignaron para asegurar que las señales lleguen a los componentes correctos, y se añadió un control para la señal de memoria (‘currentMemori‘) que monitorea el estado actual del sistema en operaciones de memoria intermedia.

4.1.3. Problemas encontrados y soluciones aplicadas

- **Problema:** La integración de los módulos adicionales (multiplexores y UART) dentro del top-level causó un mayor uso de recursos que inicialmente previsto.
  - **Solución:** Se optimizó la asignación de pines y la estructura de control, aprovechando la modularidad del diseño para redistribuir los recursos de manera más eficiente. Se implementaron multiplexores adicionales (‘uart_fpga_top_mux_‘) para simplificar la gestión de señales, logrando reducir la complejidad y mejorar la eficiencia del hardware.

4.1.4. Resultados obtenidos

Se logró la implementación completa del módulo de decodificación de entradas, integrado exitosamente con el sistema general de la FPGA. Las señales de entrada fueron correctamente decodificadas y mostradas en el display de 7 segmentos, cumpliendo con los requerimientos del proyecto. Además, se confirmó que la asignación de pines y el control del top-level (‘PG1_FAC_Top‘) permitieron una operación sincronizada y sin errores entre los módulos de transmisión UART, recepción, y el decodificador.

4.2. Fecha: 19/10/2024

4.2.1. Objetivos

- Implementar el generador de PWM en el proyecto, asegurando la correcta modulación de ancho de pulso para diversas aplicaciones de control.
- Desarrollar y probar un banco de pruebas (‘testbench‘) para validar el funcionamiento del módulo PWM bajo diferentes configuraciones de ciclo de trabajo (‘duty cycle‘).
- Integrar el módulo PWM dentro del sistema existente, asegurando compatibilidad y sincronización con otros componentes del proyecto.
- Optimizar el código y las asignaciones en el archivo ‘.qsf‘ para gestionar de forma eficiente los recursos y conexiones físicas.

4.2.2. Descripción de las tareas

Se implementó el módulo ‘PWM_Generator‘ para generar señales de PWM con ciclos de trabajo ajustables. La estructura del módulo permite configurar el ciclo de trabajo mediante un valor de entrada de 2 bits (‘duty_cycle‘), que define el porcentaje de tiempo que la señal se mantiene alta durante cada ciclo. El módulo de prueba (‘PWM_Generator_tb.sv‘) fue desarrollado para verificar el funcionamiento bajo diferentes valores de ‘duty_cycle‘.

En este banco de pruebas, se simularon múltiples escenarios cambiando el valor de ‘duty_cycle‘ y verificando que la señal ‘pwm_out‘ correspondiera correctamente al ciclo esperado. Se emplearon aserciones (‘asserts‘) para confirmar que el comportamiento del PWM coincidiera con el ciclo de trabajo establecido, asegurando que el módulo funcionara de manera estable y precisa en condiciones reales de operación.

Además, se realizaron cambios en la configuración del archivo ‘.qsf‘, ajustando la entidad principal del proyecto a ‘PWM_Generator‘. Se aseguraron las asignaciones de pines necesarias para la correcta conexión física y operación del módulo en la FPGA. Esto incluyó la definición de pines de entrada para el control de ‘duty_cycle‘ y la salida para la señal modulada (‘pwm_out‘).

4.2.3. Problemas encontrados y soluciones aplicadas

- **Problema:** Durante las pruebas iniciales, se observó que el ‘pwm_out‘ no seguía correctamente el ciclo de trabajo para ciertos valores de ‘duty_cycle‘.
  - **Solución:** Se ajustó la lógica interna del generador PWM, asegurando que los contadores sincronizados con la señal de reloj (‘clk‘) procesaran correctamente las transiciones de alto a bajo. Se realizaron simulaciones adicionales para comprobar el correcto ajuste de tiempos y evitar discrepancias en el comportamiento de la señal.
- **Problema:** La configuración de pines en el archivo ‘.qsf‘ no reflejaba adecuadamente las necesidades de conexión física, lo que generaba conflictos durante la programación de la FPGA.
  - **Solución:** Se redefinieron las asignaciones de pines, separando las entradas de control y las salidas de la señal PWM. Además, se actualizó la entidad principal a ‘PWM_Generator‘, asegurando que el proyecto reconociera correctamente los módulos y archivos relacionados con el PWM.

4.2.4. Resultados obtenidos

Las pruebas realizadas con el banco de pruebas confirmaron que el módulo generaba correctamente los ciclos de trabajo deseados, ajustándose según la configuración de ‘duty_cycle‘. El ajuste de la lógica interna permitió mejorar la precisión y estabilidad de la señal, evitando problemas de sincronización.

A continuación se muestra un diagrama que describe las partes que conforman el módulo PWM y se detalla su funcionamiento:

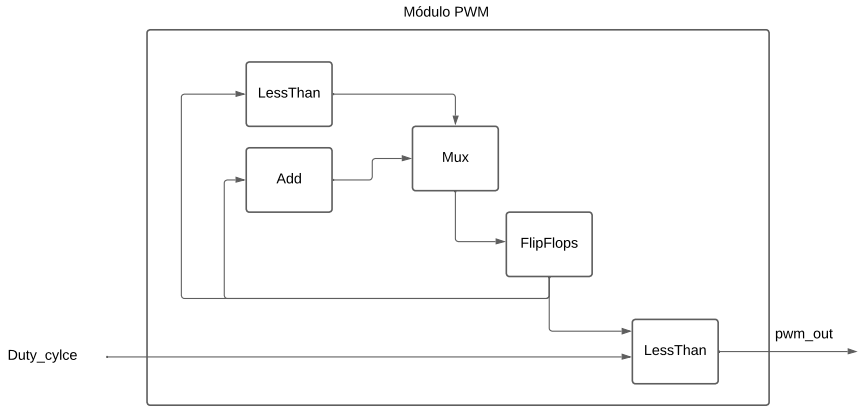

Figure 4.1: Módulo PWM

Componentes principales:

- Contador (counter):

  Es un registro que cuenta de 0 a 3, en cada flanco de subida del reloj.

  El valor de este contador se compara contra el valor del ciclo de trabajo (duty_cycle[1..0]) usando comparadores.

- Sumador (Add0):

  Este sumador recibe dos valores: 2’h1 y el valor actual del contador (counter[1..0]), generando una nueva salida que se alimenta de nuevo al contador, incrementando su valor. De esta forma, el contador se va incrementando de manera cíclica cada vez que pasa un ciclo del reloj.

- Comparador (LessThan0 y LessThan1):

  El LessThan0 compara el valor del contador con 2’h0, lo cual ayuda a controlar la cuenta y el reset del contador. El LessThan1 compara el valor del contador con el

ciclo de trabajo (duty_cycle[1..0]). Este comparador genera la salida pwm_out basada en la comparación.

Funcionamiento del circuito

El contador está constantemente incrementándose de 0 a 3 en sincronía con la señal de reloj. El comparador LessThan1 compara el valor del contador con el valor del ciclo de trabajo (duty_cycle). Si el valor del contador es menor al valor del ciclo de trabajo, la señal de salida pwm_out estará en alto (1), y si es mayor, estará en bajo (0).

Dependiendo del valor del duty_cycle[1..0], el ancho de la señal de salida (la proporción de tiempo que está en alto o bajo) cambiará. Por ejemplo, si duty_cycle = 3, la señal PWM estará en alto casi todo el tiempo, mientras que si duty_cycle = 0, la señal estará en bajo la mayor parte del tiempo.

4.3. Reflexión sobre el avance del proyecto

El módulo PWM logra generar de forma satisfactoria una señal adecuada para el manejo del motor. Se logra modular la señal de tal forma que un ciclo de trabajo mayor significa una señal en alto por más tiempo, y uno menor significa una señal en bajo durante la mayor parte del ciclo. El contador y los comparadores aseguran que esta modulación se sincronice con la señal de reloj para obtener una señal PWM precisa.

4.4. Fecha: 19/10/2024

4.4.1. Objetivos

- Integrar los módulos individuales desarrollados, asegurando que cada componente funcione en conjunto con los demás.
- Validar la sincronización y comunicación entre los módulos, incluyendo acoples manuales de la ALU, decodificadores, y el PWM.
- Realizar pruebas exhaustivas para confirmar que las señales y módulos interactúan correctamente, sin errores de sincronización.
- Optimizar la configuración del proyecto en el archivo ‘.qsf‘ para asegurar que los pines y asignaciones se gestionen de manera efectiva.

4.4.2. Descripción de las tareas

En esta etapa del proyecto, se centró la atención en acoplar los módulos desarrollados, con especial énfasis en componentes como 'PWM_Generator', 'Decodificador', 'ALU', y los módulos de operaciones lógicas ('andOP', 'orOP'). Cada uno de estos componentes se diseñó de forma modular para permitir una integración fluida.

El proceso comenzó con la conexión del módulo 'PWM_Generator', asegurando que su señal de salida 'pwm_out' pudiera ser controlada y monitoreada a través del módulo de la ALU, el cual gestiona operaciones básicas como AND, OR, y suma. Se realizaron ajustes en la lógica de control para que las señales del PWM se gestionaran adecuadamente en sincronía con las operaciones de la ALU.

Además, se integró el 'Decodificador'. La conexión entre el decodificador y otros componentes permitió que la salida del decodificador gestionara la activación y desactivación de operaciones dentro del sistema. Para asegurar la precisión de esta integración, se realizaron pruebas individuales en cada módulo antes de la integración general, asegurando que los datos fluyeran de manera consistente y sin pérdidas de sincronización.

Los cambios manuales realizados incluyeron la modificación de archivos de simulación para asegurar que se reflejaran las rutas correctas de cada módulo y la correcta asociación de pines en el archivo '.qsf'. La estructura del proyecto fue actualizada para que la entidad principal pudiera gestionar los módulos 'fulladderOP', 'PWM_Generator', y 'andOP' dentro del mismo entorno de simulación, eliminando conflictos previos que surgían de configuraciones incorrectas. Esto incluyó una asignación precisa de los pines de entrada y salida para módulos como 'dff', lo que aseguró la correcta propagación de señales entre componentes.

4.4.3. Problemas encontrados y soluciones aplicadas

- **Problema:** Hubo conflictos en las asignaciones de pines en el archivo '.qsf', causando errores durante la compilación.
  - **Solución:** Se revisó y modificó el archivo '.qsf' para eliminar asignaciones redundantes y asegurar que cada pin estuviera correctamente asignado al módulo correspondiente. Este ajuste permitió una correcta interacción física entre los módulos y eliminó errores de ruta que se detectaron durante la simulación.
- **Problema:** Las simulaciones iniciales mostraron inconsistencias en la operación conjunta de 'PWM_Generator' y la ALU.

- **Solución:** Se realizaron ajustes manuales en los scripts de simulación, asegurando que las configuraciones del ‘PWM_Generator‘ se alinearan con los tiempos de reloj de la ALU. Se agregó lógica adicional para resetear señales en caso de interrupciones, logrando una sincronización precisa y constante entre componentes.

4.4.4. Resultados obtenidos

La integración de los módulos fue buena, logrando una operación estable y sincronizada entre todos los componentes. Las pruebas realizadas mostraron que el sistema podía gestionar operaciones lógicas complejas y sincronizar señales PWM sin errores de sincronización.

References