Instituto Politécnico Nacional

Escuela Superior de Cómputo

Arquitectura de computadoras

Practica 2

Sumador con acarreo en serie y acarreo anticipado

Profesor:

María Elena Aguilar Jauregui

Alumnos:

Lechuga Canales Héctor Jair

García Quiroz Gustavo Ivan

Eduardo Alfonso Rivera Morelos

Quintero Maldonado Fabián

Grupo:

5CV1

Fecha de entrega: 13/04/2025

Índice

1 Introducción

En arquitectura de computadoras, los circuitos aritméticos representan la base fundamental para el procesamiento de datos. Esta práctica se enfocó en el diseño, implementación y comparación de dos arquitecturas clave de sumadores: el sumador con acarreo en serie (Ripple Carry Adder, RCA) y el sumador con acarreo anticipado (Carry Lookahead Adder, CLA), utilizando el lenguaje VHDL y la plataforma de desarrollo Lattice Diamond para la FPGA MachXO2.

El sumador con acarreo en serie (RCA) es la implementación más básica y directa de un sumador binario. Su estructura en cascada, donde el acarreo de salida (Cout) de un bit se convierte en el acarreo de entrada (Cin) del siguiente, lo hace sencillo de diseñar y requiere menos recursos de hardware. Sin embargo, su principal desventaja radica en el retardo acumulativo, ya que el acarreo debe propagarse secuencialmente desde el bit menos significativo (LSB) hasta el más significativo (MSB). Esto limita su velocidad en operaciones de múltiples bits, especialmente en aplicaciones de alto rendimiento como procesadores modernos o unidades aritmético-lógicas (ALU).

Por otro lado, el sumador con acarreo anticipado (CLA) aborda este problema mediante un enfoque paralelo. En lugar de esperar a que el acarreo se propague bit a bit, el CLA calcula todos los acarreos simultáneamente utilizando señales de generación (G = A AND B) y propagación (P = A XOR B), lo que permite una ejecución más rápida. Aunque esta técnica mejora significativamente el rendimiento, requiere una mayor cantidad de compuertas lógicas y, por ende, un mayor consumo de área en el silicio. Esta compensación entre velocidad y recursos hardware es un dilema clásico en el diseño de circuitos integrados, donde el ingeniero debe decidir qué priorizar según la aplicación.

La implementación de estos sumadores en VHDL para la FPGA MachXO2 permitió no solo comprender sus diferencias teóricas, sino también experimentar con su síntesis y comportamiento real en hardware programable. El uso de Lattice Diamond

como entorno de desarrollo facilitó la simulación y verificación de los diseños, asegurando su correcto funcionamiento antes de la programación física. Además, esta práctica reforzó conceptos clave como el manejo de señales en VHDL, la optimización de circuitos combinacionales y la importancia de las restricciones de tiempo (timing constraints) en sistemas digitales.

Finalmente, esta actividad no solo cumplió con el objetivo académico de comparar dos arquitecturas de sumadores, sino que también sirvió como puente entre la teoría y la práctica. Los resultados obtenidos proporcionaron insights valiosos sobre cómo las decisiones de diseño impactan en el rendimiento de un sistema digital, preparando el terreno para futuras aplicaciones en proyectos más complejos, como unidades de procesamiento gráfico (GPU), procesadores de señales digitales (DSP) o sistemas embebidos de alta eficiencia.

2 Objetivos

Implementar y comparar el funcionamiento de un sumador completo con acarreo en serie y un sumador con acarreo anticipado en VHDL, sintetizándolos en la FPGA MachXO2 para evaluar sus diferencias en velocidad y uso de recursos.

2.1 Objetivos Particulares

- Diseñar un sumador completo de 4 bits con acarreo en serie (RCA) en VHDL, analizando su propagación secuencial.
- Implementar un sumador con acarreo anticipado (CLA) de 4 bits, aprovechando señales de generación y propagación para acelerar el cálculo.
- Sintetizar ambos diseños en Lattice Diamond para la tarjeta MachXO2, verificando su correcta operación.
- Comparar los recursos lógicos utilizados y los tiempos de propagación de ambos sumadores.
- Documentar las ventajas y desventajas de cada arquitectura en un reporte técnico.

3 Desarrollo

En esta práctica se implementaron dos tipos de sumadores en VHDL para la tarjeta MachXO2 utilizando Lattice Diamond:

1. Sumador con Acarreo Anticipado (Carry Lookahead Adder - CLA)
2. Sumador Completo con Acarreo en Serie (Ripple Carry Adder - RCA)

A continuación, se detalla el procedimiento seguido para cada uno:

3.1 Sumador con Acarreo Anticipado (Carry Lookahead Adder - CLA)

Se implementó un sumador de 4 bits utilizando la técnica de acarreo anticipado, que calcula los bits de acarreo en paralelo para reducir el tiempo de propagación.
Se definieron señales de Generación (G) y Propagación (P) para cada bit:

- G = A AND B (indica si se genera un acarreo).
- P = A XOR B (indica si el acarreo se propaga).

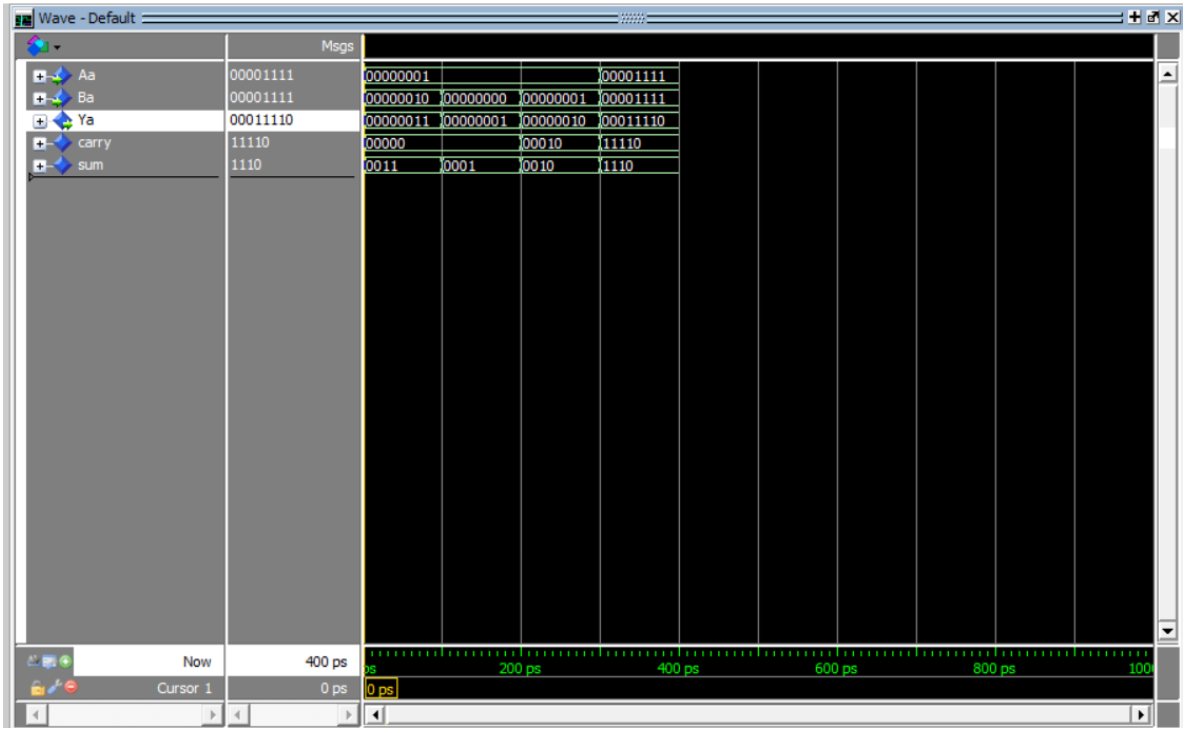
La lógica de acarreo se calculó de manera anticipada para evitar la propagación secuencial.

3.1.1 Implementación en VHDL:

Se utilizaron las entradas Aa(3:0) y Ba(3:0) para los operandos de 4 bits.
La salida Ya(3:0) contiene el resultado de la suma.
Los bits superiores (Ya(7:4) se fijaron en 0 ya que solo se trabajó con 4 bits.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library lattice;
use lattice.all;

entity and00 is
    port(
        Aa: in std_logic_vector(7 downto 0);
        Ba: in std_logic_vector(7 downto 0);
        Ya: out std_logic_vector(7 downto 0));
end and00;

architecture and0 of and00 is
    -- Internal signals for the 4-bit carry lookahead adder
    signal G, P: std_logic_vector(3 downto 0);  -- Generate and Propagate
    signal C: std_logic_vector(4 downto 0);     -- Carry signals
begin
    -- Initialize carry in (could be set to '0' or use another input if needed)
    C(0) <= '0';

    -- Generate and Propagate signals for each bit
    G <= Aa(3 downto 0) and Ba(3 downto 0);
    P <= Aa(3 downto 0) xor Ba(3 downto 0);

    -- Carry lookahead logic
    C(1) <= G(0) or (P(0) and C(0));
    C(2) <= G(1) or (P(1) and G(0)) or (P(1) and P(0) and C(0));
    C(3) <= G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or
            (P(2) and P(1) and P(0) and C(0));
    C(4) <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or
            (P(3) and P(2) and P(1) and G(0)) or
            (P(3) and P(2) and P(1) and P(0) and C(0));

    -- Sum calculation
    Ya(3 downto 0) <= P xor C(3 downto 0);

    -- Set upper bits to zero since we're only implementing a 4-bit adder
    Ya(7 downto 4) <= "0000";
end and0;
```

3.1.2 Simulación



3.2 Sumador Completo con Acarreo en Serie (Ripple Carry Adder - RCA)

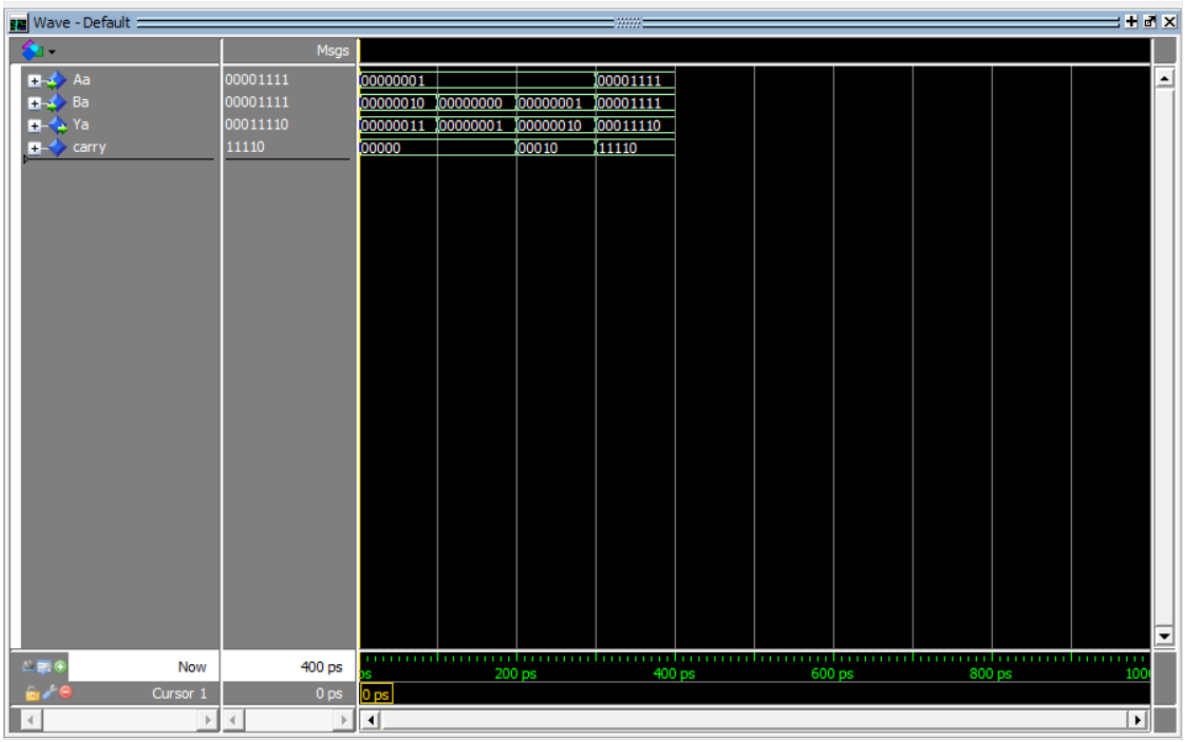
Se implementó un sumador de 4 bits utilizando la estructura de acarreo en serie, donde el acarreo se propaga de un bit al siguiente. Cada Full Adder toma A, B y el acarreo anterior (Cin) para producir la suma (S) y el acarreo de salida (Cout).

3.2.1 Implementación en VHDL

Se utilizó un bucle para instanciar 4 sumadores en cascada. La señal carry(4:0) maneja los acarreos entre etapas. El resultado de la suma se almacena en Ya(3:0), mientras que el acarreo final se asigna a Ya(4). Los bits no utilizados (Ya(7:5)) se fijaron en 0.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library lattice;
use lattice.all;

entity and00 is
    port(
        Aa: in std_logic_vector(7 downto 0);
        Ba: in std_logic_vector(7 downto 0);
        Ya: out std_logic_vector(7 downto 0));
end and00;

architecture and0 of and00 is
    signal carry: std_logic_vector(4 downto 0);
    signal sum: std_logic_vector(3 downto 0);
begin
    -- 4-bit full adder implementation
    carry(0) <= '0'; -- Initial carry in is 0

    -- Generate the 4 full adders
    gen_adders: for i in 0 to 3 generate
        sum(i) <= Aa(i) xor Ba(i) xor carry(i);
        carry(i+1) <= (Aa(i) and Ba(i)) or (Aa(i) and carry(i)) or (Ba(i) and carry(i));
    end generate gen_adders;

    -- Output assignment
    Ya(3 downto 0) <= sum;          -- Sum result in lower 4 bits
    Ya(4) <= carry(4);              -- Carry out in bit 4
    Ya(7 downto 5) <= "000";        -- Unused bits set to 0
end and0;
```

3.2.2 Simulación



4 Conclusiones

- **Lechuga Canales Héctor Jair**

Esta práctica fue una oportunidad invaluable para entender la importancia de la optimización en el diseño de circuitos digitales. Al implementar el CLA, pude apreciar cómo el cálculo paralelo de acarreos reduce drásticamente los retardos, aunque a costa de un mayor uso de recursos. Esto me hizo reflexionar sobre cómo en la industria, especialmente en aplicaciones como procesadores de alto rendimiento, se prioriza la velocidad sobre el área. También aprendí a manejar herramientas como Lattice Diamond, lo que me permitió verificar el comportamiento real de los sumadores en una FPGA. Como área de mejora, identifico la necesidad de profundizar en técnicas de síntesis para optimizar aún más el uso de LUTs y flip-flops en diseños futuros.

- **García Quiroz Gustavo Ivan**

El desarrollo de ambos sumadores me permitió ver que el RCA fue intuitivo y fácil de implementar, su limitación en velocidad lo hace inviable para sistemas que requieren operaciones rápidas. Por otro lado, el CLA, aunque más complejo, demostró ser una solución elegante al problema de propagación de acarreos. Un aprendizaje clave fue el uso de señales de generación y propagación, que simplifican la lógica del circuito. Sin embargo, enfrenté desafíos al depurar el código VHDL, lo que me enseñó la importancia de las simulaciones previas a la implementación física.

- **Eduardo Alfonso Rivera Morelos**

Trabajar con la FPGA MachXO2 fue un parteaguas en mi comprensión de los sistemas digitales. El CLA, en particular, me sorprendió por su eficiencia al eliminar cuellos de botella en la propagación de acarreos, algo crítico en aplicaciones como procesadores multinúcleo. Sin embargo, confirmé que no existe una solución universal: el RCA sigue siendo relevante en sistemas de bajo consumo donde la velocidad no es prioritaria. Además, esta práctica me ayudó a dominar el uso de bucles generate en VHDL, una herramienta poderosa para diseños escalables. Como reflexión, veo necesario estudiar más a fondo las métricas de timing en FPGAs para optimizar diseños complejos.

- **Quintero Maldonado Fabián**

Como principiante en VHDL, esta práctica fue un reto que transformó mi perspectiva sobre el diseño digital. Implementar el RCA me dio confianza en los conceptos básicos, pero el CLA me mostró el potencial de la optimización algorítmica. Aprendí que cada compuerta lógica cuenta y que decisiones aparentemente pequeñas (como usar XOR en lugar de AND-OR) pueden impactar el rendimiento. La síntesis en Lattice Diamond me reveló la importancia de las restricciones físicas, como el número de LUTs disponibles. A futuro, quiero investigar cómo técnicas como pipeline podrían acelerar aún más estos diseños. Esta experiencia no solo consolidó mis bases en electrónica digital, sino que también despertó mi interés por el co-diseño hardware/software.

5 Anexo

- **Sumador con acarreo en serie**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library lattice;
use lattice.all;

entity and00 is
    port(
        Aa: in std_logic_vector(7 downto 0);
        Ba: in std_logic_vector(7 downto 0);
        Ya: out std_logic_vector(7 downto 0));
end and00;

architecture and0 of and00 is
    signal carry: std_logic_vector(4 downto 0);
    signal sum: std_logic_vector(3 downto 0);
begin
    -- 4-bit full adder implementation
    carry(0) <= '0'; -- Initial carry in is 0

    -- Generate the 4 full adders
    gen_adders: for i in 0 to 3 generate
        sum(i) <= Aa(i) xor Ba(i) xor carry(i);
        carry(i+1) <= (Aa(i) and Ba(i)) or (Aa(i) and carry(i)) or (Ba(i) and carry(i));
    end generate gen_adders;

    -- Output assignment
    Ya(3 downto 0) <= sum;          -- Sum result in lower 4 bits
    Ya(4) <= carry(4);              -- Carry out in bit 4
```

```
    Ya(7 downto 5) <= "000";        -- Unused bits set to 0
end and0;
```

- **Sumador con acarreo anticipado**

```
library ieee;
use ieee.std_logic_1164.all;
library lattice;
use lattice.all;

entity and00 is
    port(
        Aa: in std_logic_vector(7 downto 0);
        Ba: in std_logic_vector(7 downto 0);
        Ya: out std_logic_vector(7 downto 0));
end and00;

architecture and0 of and00 is
    -- Internal signals for the 4-bit carry lookahead adder
    signal G, P: std_logic_vector(3 downto 0);  -- Generate and Propagate
    signal C: std_logic_vector(4 downto 0);     -- Carry signals
begin
    -- Initialize carry in (could be set to '0' or use another input if needed)
    C(0) <= '0';

    -- Generate and Propagate signals for each bit
    G <= Aa(3 downto 0) and Ba(3 downto 0);
    P <= Aa(3 downto 0) xor Ba(3 downto 0);

    -- Carry lookahead logic
    C(1) <= G(0) or (P(0) and C(0));
    C(2) <= G(1) or (P(1) and G(0)) or (P(1) and P(0) and C(0));
    C(3) <= G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or
```

```
        (P(2) and P(1) and P(0) and C(0));
    C(4) <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or
        (P(3) and P(2) and P(1) and G(0)) or
        (P(3) and P(2) and P(1) and P(0) and C(0));

    -- Sum calculation
    Ya(3 downto 0) <= P xor C(3 downto 0);

    -- Set upper bits to zero since we're only implementing a 4-bit adder
    Ya(7 downto 4) <= "0000";
end and0;
```