Instituto Politécnico Nacional
Escuela Superior de Cómputo
“ESCOM”

Unidad de Aprendizaje:
Diseño de sistemas digitales

Entregable No. 7
SUMADOR 3 BITS

Integrantes:
Álvarez Hernández Gabriel Alexander
García Quiroz Gustavo Iván
Huesca Laureano Josué Alejandro
Muñoz Valdivia Irving Omar
Pedroza Villagómez Emir

Nombre del profesor: Flores Escobar José Antonio

Índice de contenido

Introducción

El desarrollo de programas que emplean hilos y su comunicación eficiente es un aspecto fundamental en la programación concurrente y paralela. El uso de hilos permite dividir tareas en segmentos más pequeños y ejecutarlos de manera simultánea, optimizando el rendimiento y reduciendo el tiempo de ejecución. En este informe, exploraremos la implementación de un programa en lenguaje C que utiliza hilos para la comunicación entre ellos y con el proceso padre, abordando detalles sobre la estructura del código, la implementación de funciones clave y las pruebas necesarias para garantizar su correcto funcionamiento.

Los hilos son flujos de ejecución independientes dentro de un mismo proceso, compartiendo recursos como el espacio de memoria y permitiendo una comunicación eficiente entre ellos. Esta característica los convierte en una solución ideal para abordar problemas que requieren el procesamiento paralelo de datos o la realización de múltiples tareas simultáneas. Según Silberschatz et al. (2018), "los hilos brindan una forma de mejorar el rendimiento a través del paralelismo" (p. 234).

El objetivo general de esta práctica es desarrollar programas en lenguaje C que permitan la creación de hilos y la comunicación entre ellos. Con esta práctica, se busca entender y aplicar conceptos de programación concurrente para solucionar problemas de manera eficiente. Los hilos pueden ser útiles en diversas aplicaciones, desde sistemas embebidos hasta aplicaciones de alta concurrencia, y es crucial aprender a usarlos de manera efectiva.

En las secciones siguientes, describiremos cómo se diseñó e implementó un programa que crea múltiples hilos para procesar una matriz de datos. Se explicará cómo cada hilo se comunica con el proceso padre utilizando apuntadores, permitiendo una transferencia de información entre ellos.

La sección de implementación detallará los pasos requeridos para compilar y ejecutar el programa en un entorno GNU/Linux Ubuntu, destacando el uso del compilador **gcc** y la librería de hilos **pthread.h**. Además, se analizará la estructura del código, centrándose en la creación y gestión de hilos, la comunicación entre ellos y el proceso padre, y la función principal que ejecuta las operaciones necesarias.

Para validar el correcto funcionamiento del programa, se llevaron a cabo pruebas específicas. La sección de pruebas detallará estos ensayos y sus resultados, confirmando que los hilos se comunican eficazmente y realizan las tareas asignadas sin errores ni comportamientos inesperados. Además, se verificará que los resultados obtenidos coincidan con las expectativas previstas.

Este informe proporciona una visión general de cómo los hilos pueden utilizarse en programas C para lograr comunicación entre ellos y con el proceso padre. Se destacarán los desafíos encontrados durante la implementación y cómo se resolvieron para garantizar un programa robusto y funcional. Con este conocimiento, los lectores obtendrán una comprensión profunda de la importancia de la programación concurrente y cómo aplicarla a problemas reales.

A través de este proyecto, se explorará la creación de hilos y la comunicación eficiente entre ellos utilizando el lenguaje C y la biblioteca pthreads. Esto permitirá desarrollar habilidades en programación concurrente, una habilidad esencial en el mundo actual, donde la demanda de

procesamiento paralelo y el aprovechamiento óptimo de los recursos son fundamentales para el desarrollo de aplicaciones de alto rendimiento.

Marco teórico

Contador

Un **contador** es un circuito secuencial construido a partir de biestables y puertas lógicas capaces de almacenar y contar los impulsos (a menudo relacionados con una señal de reloj), que recibe en la entrada destinada a tal efecto, así mismo también actúa como divisor de frecuencia. Normalmente, el cómputo se realiza en código binario, que con frecuencia será el binario natural o el BCD natural (contador de decenas). Ejemplo, un contador de módulo 4 pasa por 4 estados, y contaría del 0 al 3. Si necesitamos un contador con un módulo distinto de 2^n, lo que haremos es añadir un circuito combinacional.

Clasificación de los contadores de circuito secuencial

- Según la forma en que conmutan los números, podemos hablar de contadores **numeradores** (todos los números conmutan a la vez, con una señal de reloj común) o **asíncronos** (el reloj no es común y los números conmutan uno tras otro).
- Según el sentido de la cuenta, se distinguen en ascendentes, descendentes y UP-DOWN o numéricos (alterna en ascendentes o descendentes según la señal de control).
- Según la cantidad de números que pueden contar, se puede hablar de *contadores binarios de n bits* (cuentan todos los números posibles de n bits, desde 0 hasta), *contadores BCD* (cuentan del 0 al 9).

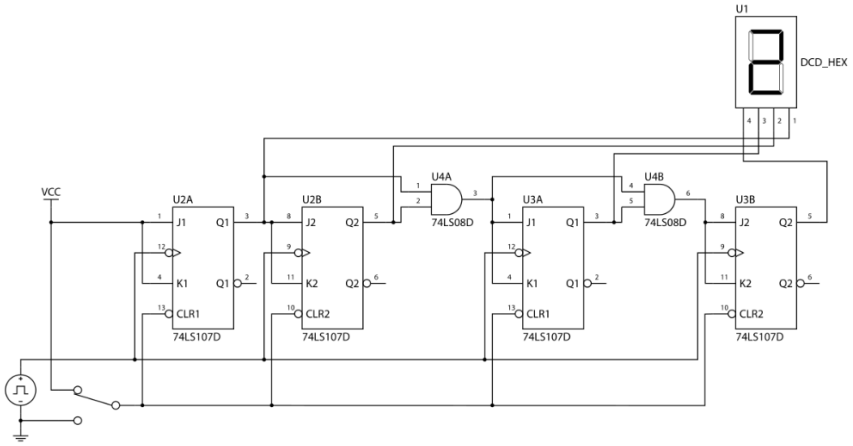

Figura 1 Contador síncrono de 4 bits.

El número máximo de estados por los que pasa un contador se denomina módulo del contador(*Número MOD*). Este número viene determinado por la expresión 2^n donde *n* indica el número de bits del contador. Ejemplo, un contador de módulo 4 pasa por 4 estados, y contaría del 0 al 3. Si necesitamos un contador con un módulo distinto de 2^n, lo que haremos es añadir un circuito combinacional.

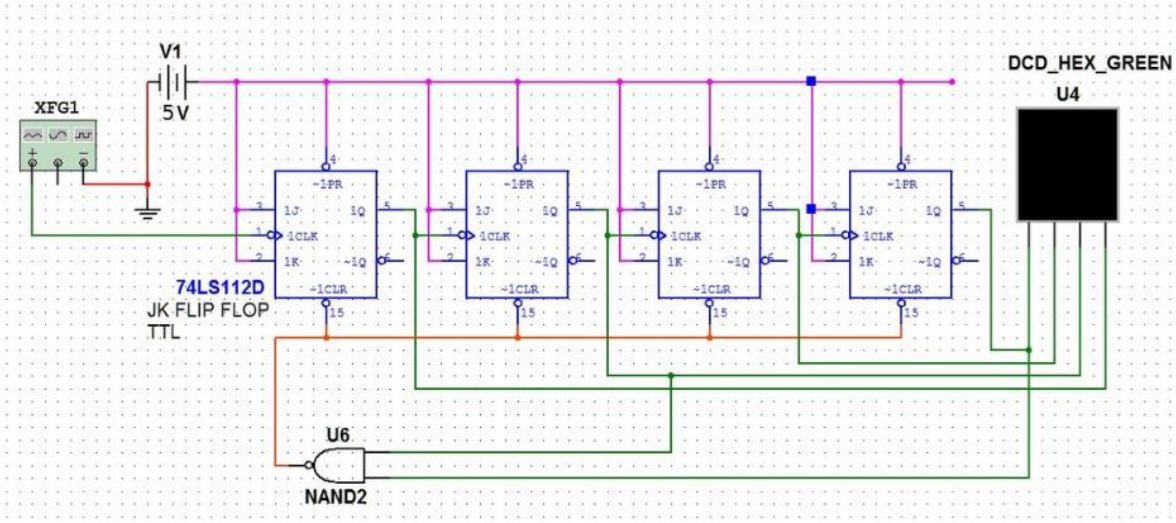

*Figura 2***Contador base 10.**

Desarrollo

El programa implementado es un contador binario ascendente/descendente de 3 bits desarrollado en Verilog HDL. El diseño consta de dos módulos principales: sumador3bits y DivFreq.

Implementación del código Verilog

El diseño del contador ascendente/descendente de 3 bits se implementó utilizando el lenguaje de descripción de hardware Verilog. A continuación, se muestra el código Verilog utilizado para cada uno de los módulos principales.

Módulo sumador3bits

```
/*
   Proyecto: sumador3bits
   Archivo: sumador3bits.v
   Descripcion: contador ascendente/descendente de 3 bits

   Asignatura: DSD
   Profesor: Flores Escobar Jose Antonio
   Equipo: Coloque a los integrantes...
            Álvarez Hernández Gabriel Alexander
            Bueno Aguilar Alexis Haziel
            Garcia Quiroz Gustavo Ivan
            Huesca Laureano Josue Alejandro
            Muñoz Valdivia Irving Omar
*/
module sumador3bits(
    input clk_i,
    input rst_ni,
    input dir_i,
    output [2:0] count
);

wire clk_div; // Reloj dividido por DivFreq
```

```
DivFreq #(
    .freqdev(10000000), // 10 MHz
    .freqfinal(10000000 / 4) // Dividido en 4
) div_freq (
    .clk_i(clk_i),
    .rst_ni(rst_ni),
    .clk_o(clk_div)
);

reg [2:0] count_r;

// Contador de 3 bits ascendente
always @(posedge clk_div or negedge rst_ni) begin
    if (!rst_ni) begin
        count_r <= 3'b000; // Reinicia el contador
    end else if (!dir_i) begin
        count_r <= count_r + 1'b1; // Incrementa el contador
           end else begin
            count_r <= count_r - 1'b1;
    end
end

assign count = count_r; // Asigna la salida
endmodule
```

Figura 3 Módulo sumador3bits

Este módulo implementa el contador ascendente/descendente de 3 bits. Utiliza un registro `count_r` para almacenar el valor actual del contador. En cada flanco positivo del reloj `clk_div` (generado por el módulo `DivFreq`), el valor del contador se incrementa o decrementa en función de la señal de dirección `dir_i`. Si la señal de reinicio `rst_ni` está activa (en bajo), el contador se reinicia a 000

Módulo DivFreq

```
module DivFreq #(
    parameter freqdev = 10000000, // 10MHz
    parameter freqfinal = freqdev / 4 // Dividido en 4
)(
    input       clk_i,
    input       rst_ni,
    output reg  clk_o
);
reg [31:0] counter_r;
always @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
        counter_r <= 0;
        clk_o <= 1'b0; // Comenzamos con el reloj en bajo
    end else if (counter_r >= (freqfinal - 1)) begin
        counter_r <= 0;
        clk_o <= ~clk_o;
    end else begin
        counter_r <= counter_r + 1;
    end
end
endmodule
```

Figura 4 Módulo DivFreq

Este módulo implementa un divisor de frecuencia parametrizable. Toma una señal de reloj de entrada `clk_i` y genera una señal de reloj de salida `clk_o` con una frecuencia más baja, determinada por los parámetros `freqdev` (frecuencia de entrada) y `freqfinal` (frecuencia de salida deseada).

Internamente, el módulo utiliza un contador `counter_r` que se incrementa en cada flanco positivo del reloj `clk_i`. Cuando el contador alcanza el valor `freqfinal - 1`, se reinicia a 0 y la señal `clk_o` cambia de estado (de 0 a 1 o de 1 a 0). De esta manera, la frecuencia de `clk_o` será `freqdev / freqfinal`.

El módulo `count` utiliza la señal de reloj dividida `clk_div` generada por `DivFreq` para actualizar el contador a una frecuencia más baja, lo que permite una visualización más cómoda del funcionamiento del contador.

Simulación

Para verificar el correcto funcionamiento del diseño, se realizó una simulación utilizando el simulador ModelSim de Quartus Prime. Se creó un banco de pruebas (testbench) en Verilog que instanciaba los módulos count y DivFreq, y se configuraron las formas de onda de entrada clk_i, rst_ni y dir_i.

Durante la simulación, se monitorearon las señales de salida `count y clk_div`. Se verificó que el contador se incrementara correctamente de 000 a 111 cuando `dir_i` estaba en 0, y que se decrementara de 111 a 000 cuando `dir_i` estaba en 1. Además, se comprobó que el contador se reiniciara a 000 cuando la señal `rst_ni` se activaba (en bajo).

La simulación también permitió verificar el correcto funcionamiento del divisor de frecuencia DivFreq, observando que la señal `clk_div` tenía una frecuencia cuatro veces menor que la señal `clk_i`.

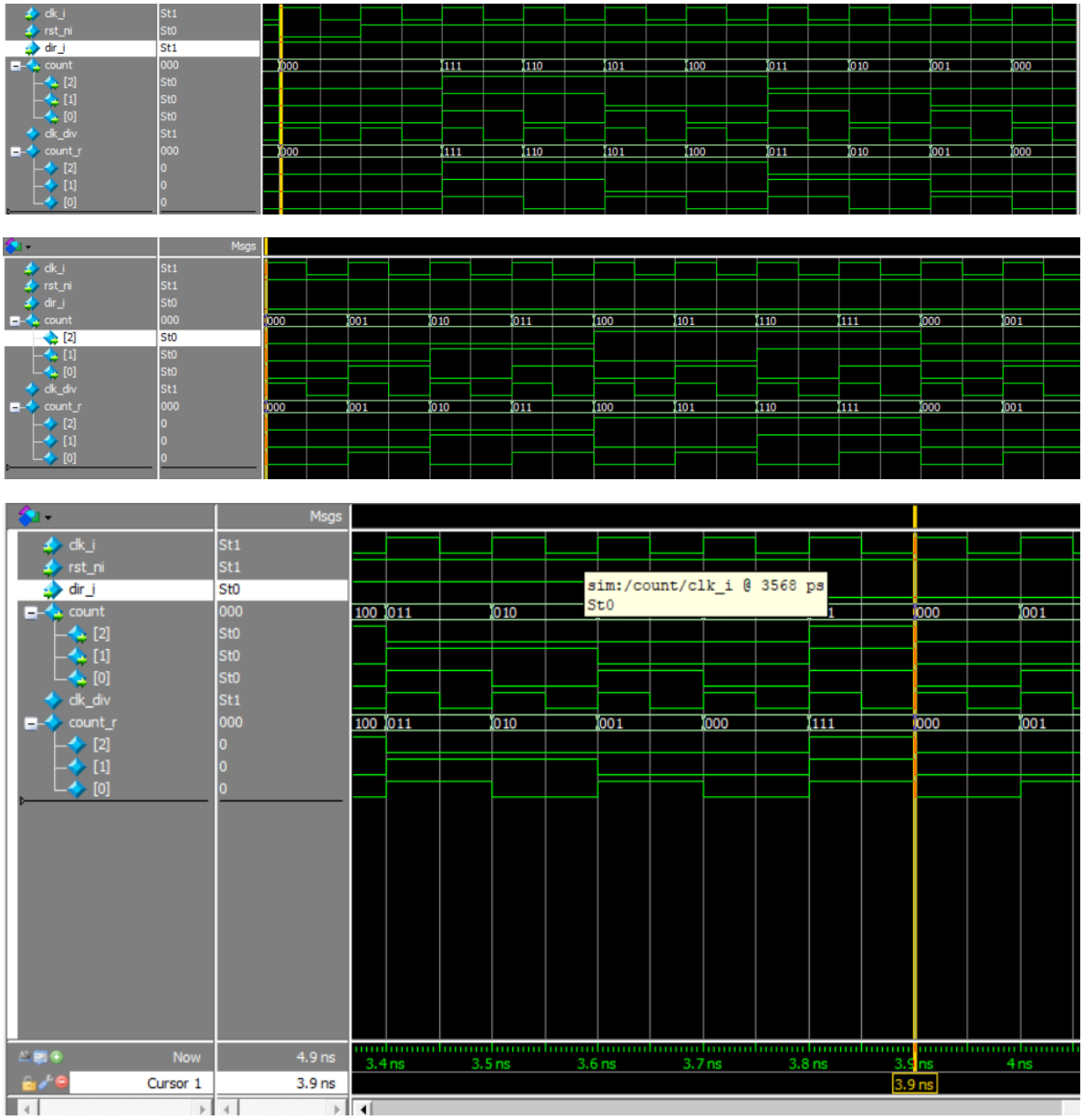

Figura 5

Análisis del circuito RTL

Después de la síntesis del diseño en Quartus Prime, se utilizó el Visor RTL (RTL Viewer) para analizar el circuito generado. Este visor muestra una representación gráfica del circuito a nivel de transferencia de registros (RTL), lo que permite visualizar la estructura y las conexiones entre los componentes del diseño.

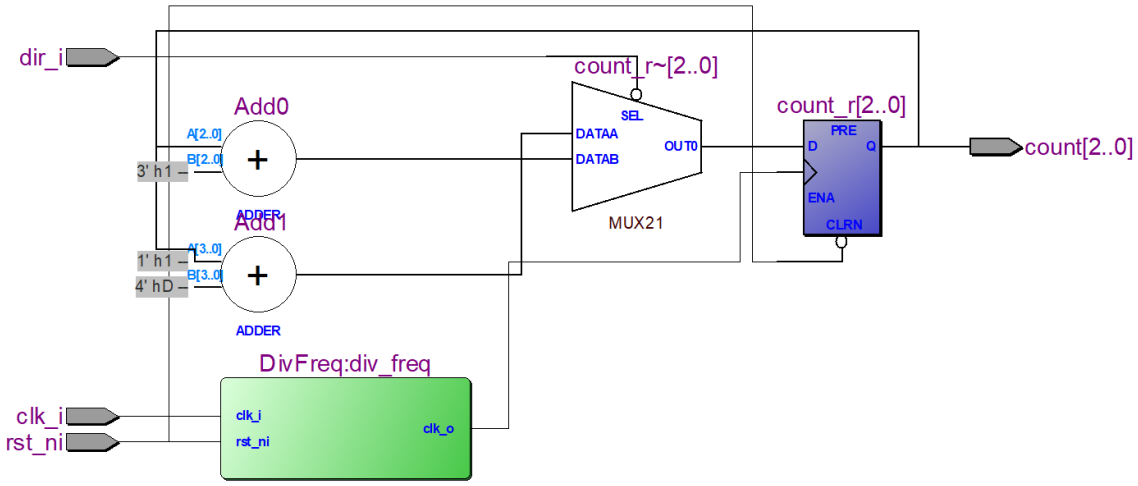

Figura 6 Análisis del circuito RTL

En el Visor RTL, se pudo observar cómo los módulos count y DivFreq se instanciaban y se conectaban entre sí. Se identificaron claramente los registros utilizados para almacenar el valor del contador (count_r) y el contador del divisor de frecuencia (counter_r), así como la lógica combinacional utilizada para actualizar estos valores.

El análisis del circuito RTL permitió verificar que la implementación del diseño era correcta y coincidía con la descripción en el código Verilog.

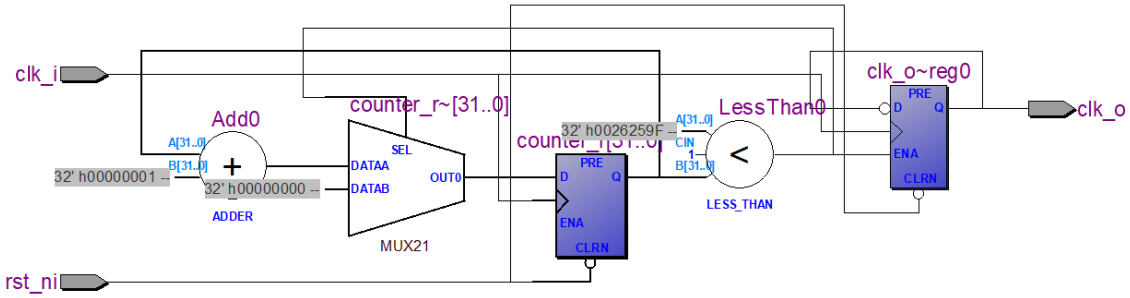

Figura 7 Circuito RTL

Implementación en hardware

Después de la simulación y el análisis del circuito RTL, se procedió a la implementación física del diseño en una tarjeta de desarrollo MAX V 5M240Z. Primero, se configuró el proyecto en Quartus Prime para utilizar esta tarjeta como dispositivo objetivo.

A continuación, se realizó el proceso de compilación del diseño, que incluye la síntesis, el mapeo de tecnología y el ensamblaje del archivo de programación (.sof). Una vez generado este archivo, se utilizó el programador USB Blaster para programar la FPGA de la tarjeta de desarrollo con el diseño del contador ascendente/descendente.

Una vez programada la FPGA, se conectaron los pines de entrada clk_i, rst_ni y dir_i a los interruptores y botones disponibles en la tarjeta de desarrollo. Los pines de salida count se conectaron a los LEDs de la tarjeta para visualizar el valor actual del contador.

Al aplicar diferentes combinaciones de entradas en los interruptores y botones, se pudo observar el correcto funcionamiento del contador ascendente/descendente en los LEDs de la tarjeta.

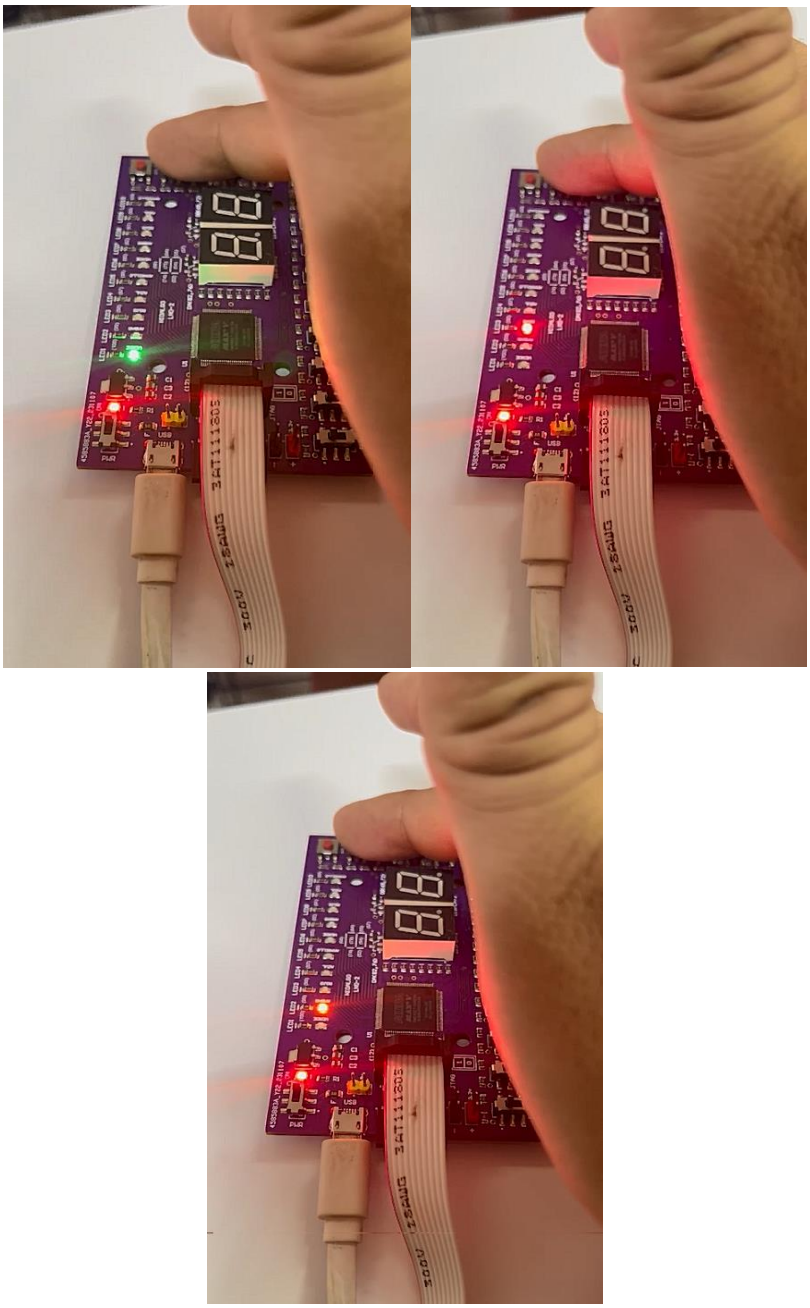

Figura 8 Implementación en hardware MAX V 5M240Z

Conclusiones

En este proyecto, se implementó un contador binario ascendente/descendente de 3 bits utilizando el lenguaje de descripción de hardware Verilog. El diseño se compone de dos módulos principales: el módulo count, que implementa la lógica del contador, y el módulo DivFreq, que actúa como un divisor de frecuencia para generar un reloj más lento a partir del reloj de entrada.

El módulo count utiliza un registro para almacenar el valor actual del contador y una lógica de actualización que incrementa o decrementa el contador en función de la señal de dirección dir_i. El módulo DivFreq permite configurar la frecuencia de salida deseada mediante los parámetros freqdev y freqfinal, lo que permite ajustar la velocidad de conteo.

El diseño se implementó y verificó en varias etapas, incluyendo la simulación funcional, el análisis del circuito RTL y la programación en una FPGA de la tarjeta de desarrollo MAX V 5M240Z. Las pruebas y verificaciones realizadas demostraron el correcto funcionamiento del contador en todas las condiciones de entrada.

Durante el desarrollo del proyecto, se adquirieron habilidades valiosas en el diseño de circuitos digitales utilizando Verilog, análisis de circuitos RTL, simulación y programación de FPGAs. Además, se profundizó en el entendimiento de los conceptos de contadores y divisores de frecuencia.

Como trabajo futuro, se podría considerar la ampliación del contador a un mayor número de bits, la implementación de funcionalidades adicionales, como la detección de desbordamiento o la posibilidad de cargar un valor inicial. Además, se podría explorar la integración del contador en diseños más complejos o en aplicaciones específicas.

En resumen, este proyecto ha permitido aplicar los conocimientos adquiridos en el diseño de circuitos digitales y ha brindado una experiencia práctica en el desarrollo de un contador ascendente/descendente utilizando Verilog y FPGAs.

Referencias

Anexo. Códigos completos

```
// sumador3bits
/*
        Proyecto: sumador3bits
        Archivo: sumador3bits.v
        Descripcion: contador ascendente/descendente de 3 bits

        Asignatura: DSD
        Profesor: Flores Escobar Jose Antonio
        Equipo: Coloque a los integrantes...
                                Álvarez Hernández Gabriel Alexander
                                Bueno Aguilar Alexis Haziel
                                Garcia Quiroz Gustavo Ivan
                                Huesca Laureano Josue Alejandro
                                Muñoz Valdivia Irving Omar
*/
module sumador3bits (
   input clk_i,
   input rst_ni,
         input dir_i,
   output [2:0] count
);

wire clk_div; // Reloj dividido por DivFreq

// Instancia del divisor de frecuencia
DivFreq #(
   .freqdev(10000000), // 10 MHz
   .freqfinal(10000000 / 4) // Dividido en 4
) div_freq (
```

```
    .clk_i(clk_i),
    .rst_ni(rst_ni),
    .clk_o(clk_div)
);

reg [2:0] count_r;

// Contador de 3 bits ascendente
always @(posedge clk_div or negedge rst_ni) begin
    if (!rst_ni) begin
        count_r <= 3'b000; // Reinicia el contador
    end else if (!dir_i) begin
        count_r <= count_r + 1'b1; // Incrementa el contador
                                end else begin
                                        count_r <= count_r - 1'b1;
    end
end

assign count = count_r; // Asigna la salida

endmodule

// DivFreq
module DivFreq #(
            parameter freqdev = 10000000, // 10MHz
            parameter freqfinal = freqdev / 4 // Dividido en 4
)(
            input               clk_i,
            input               rst_ni,
            output reg          clk_o
```

```
);
reg [31:0] counter_r;
always @(posedge clk_i or negedge rst_ni) begin
	if (!rst_ni) begin
		counter_r <= 0;
		clk_o <= 1'b0; // Comenzamos con el reloj en bajo
	end else if (counter_r >= (freqfinal - 1)) begin
		counter_r <= 0;
		clk_o <= ~clk_o;
	end else begin
		counter_r <= counter_r + 1;
	end
end
endmodule
```