INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Carrera:

Ingeniería en sistemas computacionales

Unidad de Aprendizaje:

Sistemas en chip

Práctica 4

Manejo de leds

Integrantes:

Cazares Cruz Jeremy Sajid

Bucio Barrera Oscar Daniel

Perez Ortiz Saúl

Acosta Cortes Gerardo

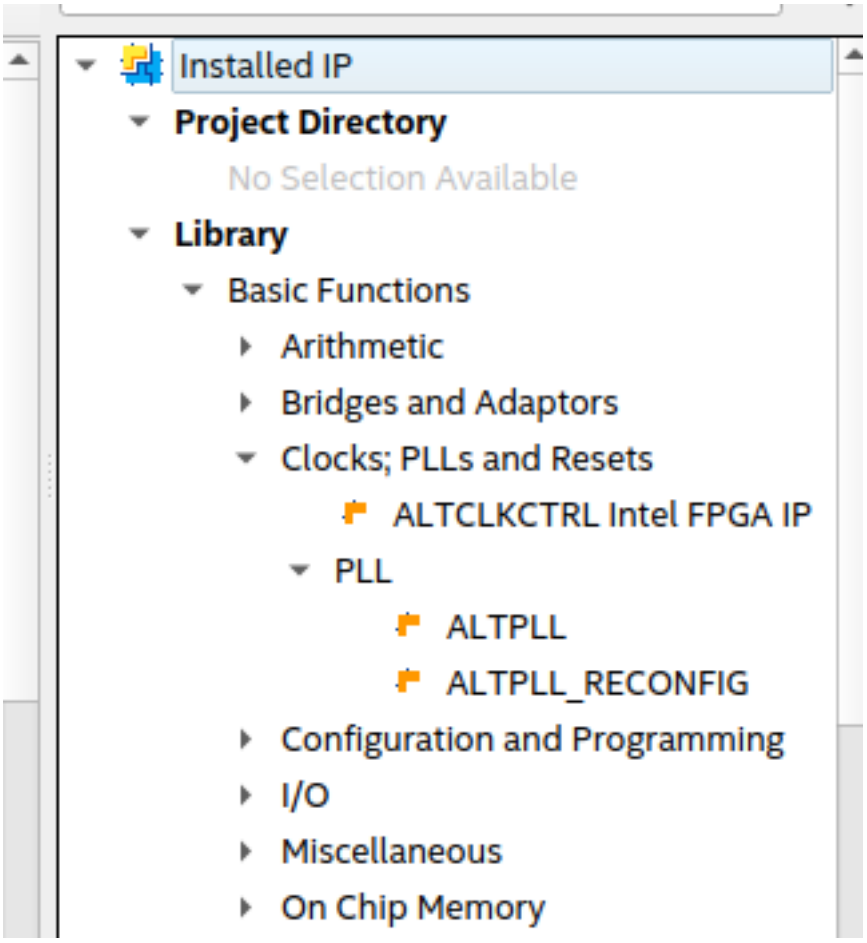
Fecha de entrega

4 de octubre de 2023

Índice

Introducción

En el desarrollo de está práctica generamos lo que es un divisor de frecuencia mediante un contador crado mediante software en verilog así como mediante un PLL usando la herramienta dada por el mismo software en la parte de “Installed IP” dentro de este apartado podemos visualizar diferentes modulos ya pre armados, mediante está práctica veremos como implementar este tipo de modulos ya prehechos con anterioridad



Marco teorico

FPGA (Field-Programmable Gate Array): Un FPGA es un dispositivo semiconductor que se puede programar después de su fabricación. A diferencia de los microcontroladores o microprocesadores, que tienen una función fija, los FPGAs se pueden reconfigurar para realizar cualquier función lógica.

Un PLL (Phase-Locked Loop) es un sistema de control que genera una señal de salida cuya fase está relacionada con la fase de una señal de entrada. La arquitectura básica de un PLL consiste en un oscilador controlado por voltaje (VCO), un comparador de fase y un filtro de lazo. El comparador de fase genera un error de fase que es filtrado y usado para ajustar la frecuencia del VCO de manera que la fase de la señal de salida coincida con la fase de la señal de entrada.

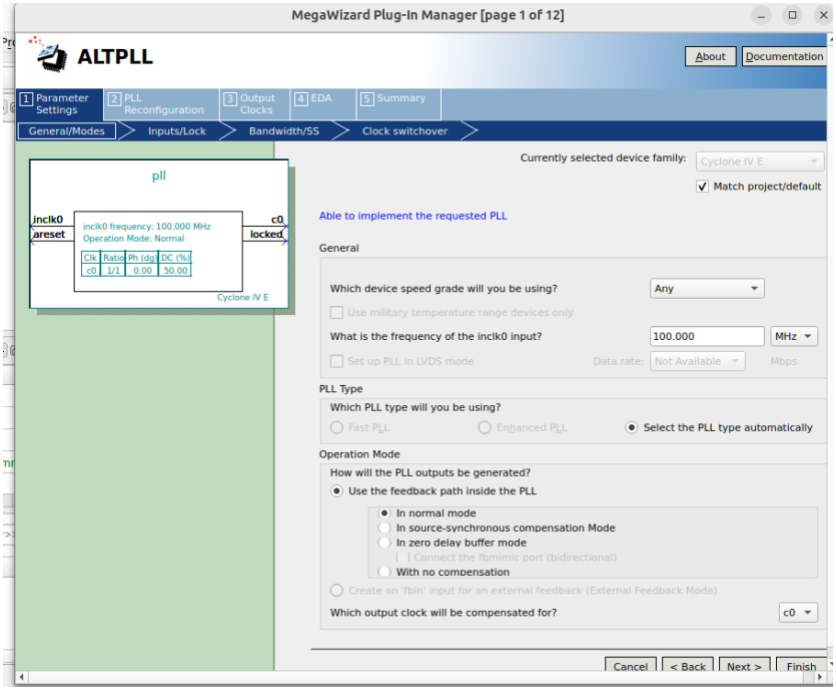
Un divisor de frecuencia es un circuito que reduce la frecuencia de una señal de entrada por un factor entero. En Verilog, un divisor de frecuencia simple se puede implementar usando un flip-flop. Por ejemplo, un divisor de frecuencia por 2 puede ser creado con un flip-flop tipo D que captura el inverso de su propio estado en cada flanco ascendente de la señal de entrada.

DE0 Nano: Es una tarjeta de desarrollo FPGA que utiliza el chip Cyclone IV de Altera (ahora Intel). Esta tarjeta es compacta y ofrece una amplia variedad de recursos, incluidos LEDs, botones, interfaces de E/S, entre otros.

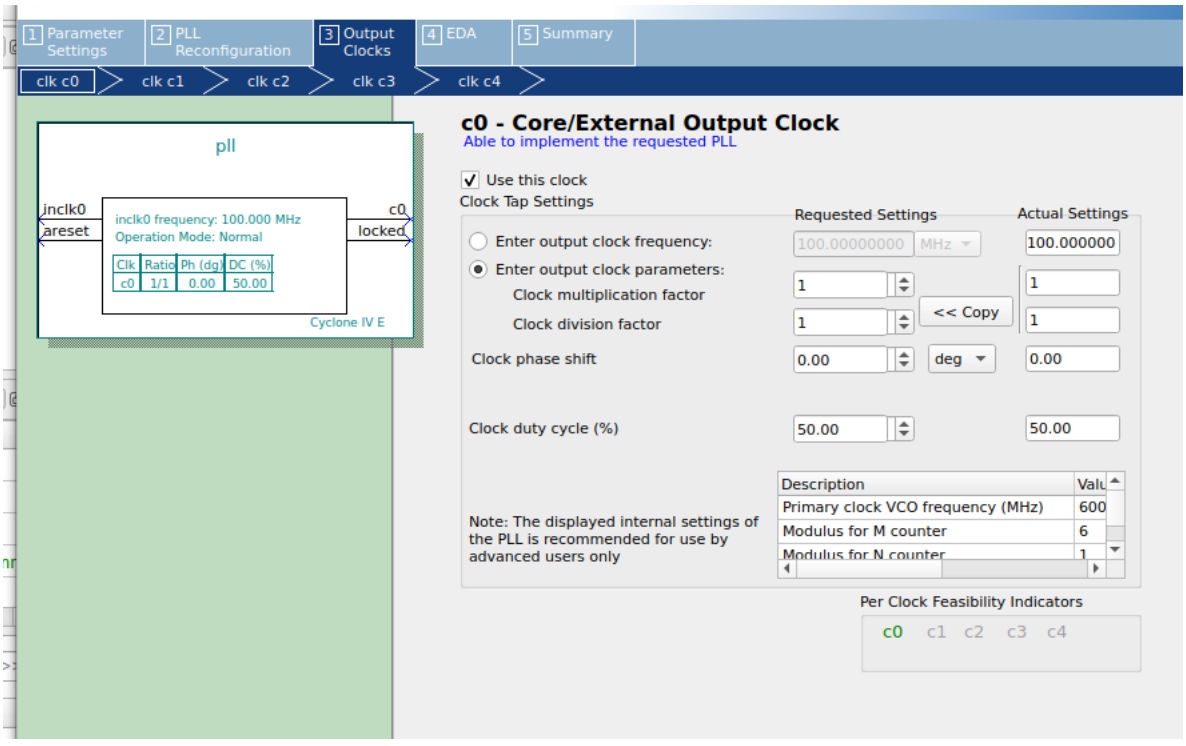
Desarrollo

Especificación del PLL: Para el DE0 Nano, se puede configurar un PLL utilizando herramientas como Quartus de Altera. El PLL tomará la frecuencia de reloj de la tarjeta (por ejemplo, 50MHz) y la multiplicará o dividirá para obtener la frecuencia deseada.

Para la realización del modulo PLL con las caracteristicas que buscamos simplemente tenemos que hacer uso de los modulos pre diseñados en el “installed ip” después buscar el apartado de PLL y finalmente empezar con las especificaciones del mismo que nos da el programa como se puede ver en la siguiente figura donde dentro de la misma nos pide la especificación de que frecuencia de salida queremos para el primero reloj en este caso para inclk0 así como en que modo se generara este PLL



Posteriomente, daremos la información necesaria como la configuración a la que deseamos llegar y la configuración que tenemos actualmente, para esto podemos generar 4 relojes diferentes



Concluyendo la configuración de los relojes que queremos simplemente falta decir de que manera queremos que se nos genere nuestro modulo con el PLL necesario, para esto usaremos el que dice “inst” para poder instanciarlo después en nuestro modulo principal de

nuestro programa con esto seremos capaces de instanciar nuestro modulo PPL y generar lo que nos resta del programa que en este caso haremon un contador con leds

/home/jeremy/Escritorio/chips/PLL/

File	Description
☑ pll.v	Variation file
☑ pll.ppf	PinPlanner ports PPF file
☐ pll.inc	AHDL Include file
☐ pll.cmp	VHDL component declaration file
☐ pll.bsf	Quartus Prime symbol file
☑ pll_inst.v	Instantiation template file
☐ pll_bb.v	Verilog HDL black-box file

El código utilizando el PLL es el siguiente:

```
module PLL#(
parameter DATA_WIDTH=8,      // Ancho de datos para RAM
parameter ADDR_WIDTH=6       // Ancho de dirección para RAM
)(
input clk50_i,              // Reloj de entrada
input rst_ni,               // Reset (activo en bajo)
output [7:0]leds_o          // Salida hacia LEDs
);

wire clk1_w;                // Reloj de salida de mipll_u0
reg [5:0] counter_w;        // Contador de 6 bits
wire we;                    // Señal de escritura para la RAM

assign we = 1'b0;           // La señal de escritura se mantiene en 0 (no hay escritura)

// Declaración de la RAM
reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

// Inicialización de la RAM desde un archivo hex
initial
begin
$readmemh("mem_init.hex",ram);
end

// Variable para almacenar la dirección de lectura registrada
reg [ADDR_WIDTH-1:0] addr_reg;
```

```
always @ (posedge clk1_w)
begin
// Escribir
if (we)
ram[counter_w] <= 8'b0;  // Si hay escritura, se escribe 0 en la RAM

addr_reg <= counter_w;       // Registrar la dirección actual del contador
end

// La lectura desde la RAM se realiza siempre
assign q = ram[addr_reg];

// Instancia del PLL2
PLL2 mipll_u0(
.clk_i  (clk50_i),
.clk_o  (clk1_w),
.rst_ni (rst_ni)
);

// Incrementa el contador con cada ciclo de reloj
always @(posedge clk1_w, negedge rst_ni)
begin
if (!rst_ni)
counter_w = 6'b0;       // Si hay reset, el contador se reinicia
else
counter_w = counter_w + 1'b1;  // Incrementa el contador
end

// La salida a los LEDs se asigna al valor del contador
assign leds_o = counter_w;

endmodule
```

Donde en cada comentario podemos saber que esta realizando línea por línea el código, aún así procedermos a explicarlo. El módulo "PLL" tiene una pequeña memoria RAM que se inicializa a partir de un archivo llamado "mem_init.hex". Esta RAM se lee basada en un contador que incrementa con cada pulso del reloj "clk1_w". El módulo también instancia un generador de reloj "PLL2" para producir el reloj "clk1_w". El contador se utiliza para mostrar un valor en LEDs y también como dirección para la lectura de la RAM.

El módulo "PLL2" actúa como un divisor de frecuencia. Genera un pulso de reloj de salida ("clk_o") que cambia su estado cada vez que el contador llega a 50000000. Esto implica que, para un reloj de entrada de 100MHz, "clk_o" tendría una frecuencia de 1Hz.

El código de nuestro PLL2 que en parte fue posible generado desde un primer instante por el proceso que hicimos pero después modificamos para hacer uso de un contador normal es el siguiente

```
module PLL2(
  input clk_i,       // Reloj de entrada
  input rst_ni,      // Reset (activo en bajo)
  output reg clk_o   // Reloj de salida
);

  reg [31:0] counter; // Contador de 32 bits

  // Controla el pulso de reloj basado en un contador
  always @(posedge clk_i or negedge rst_ni)
  begin
    if (!rst_ni)
    begin
      counter <= 32'b0;     // Si hay reset, el contador se reinicia
    end
    else
    begin
      counter <= counter + 1'b1;  // Incrementa el contador
      if (counter == 32'd50000000)
      begin
        counter <= 32'b0;   // Reinicia el contador
        clk_o <= ~clk_o;    // Cambia el estado del reloj de salida
      end
    end
  end

endmodule
```

Posibles razones por las que no se observa nada en la salida de la DE0 Nano:

A pesar de haber implementado el diseño en Verilog y haberlo cargado en la tarjeta DE0 Nano, lamentablemente, no se observa ninguna salida en los LEDs.

Se llevaron a cabo varias pruebas y diagnósticos, y se identificaron algunas posibles causas:

Conexión del reloj: Es posible que el reloj no esté correctamente conectado o configurado. A pesar de haber usado un PLL o un divisor de frecuencia, si el reloj base no está correctamente conectado, esto puede resultar en que el diseño no funcione como se espera.

Errores de compilación o sintetización: Aunque el código se haya escrito correctamente, pueden existir errores durante la etapa de sintetización que podrían afectar el funcionamiento. Es fundamental revisar los informes y warnings generados por el compilador.

Errores en la asignación de pines: Es crucial asegurarse de que los pines de la FPGA estén correctamente mapeados a los LEDs en la tarjeta DE0 Nano. Un mapeo incorrecto puede resultar en que los LEDs no muestren la salida esperada.

Problemas de alimentación o hardware: Si hay algún problema con la tarjeta DE0 Nano, como un fallo en la alimentación o un componente defectuoso, esto podría ser la razón del mal funcionamiento.

Errores en el código: Aunque el código de ejemplo es simple, siempre es posible que haya errores sutiles que impidan que funcione correctamente.

Es fundamental llevar a cabo un proceso de diagnóstico paso a paso, verificando cada uno de los componentes y etapas del diseño, para identificar y solucionar el problema.

Conclusiones

Comprender la importancia de la modularidad en el diseño de hardware: Al realizar este proyecto, es evidente que un diseño modular, como el que se llevó a cabo dividiendo el sistema entre PLL, divisor de frecuencia y el manejo de LEDs, no solo facilita la implementación sino que también hace más sencillo el proceso de depuración y validación. La modularidad permite que cada componente se diseñe, se pruebe y se valide por separado, reduciendo la complejidad y el riesgo de errores en el diseño global.

Reconocimiento de la versatilidad de FPGAs: La tarjeta DE0 Nano, siendo una FPGA, demostró ser una herramienta poderosa y adaptable. Los FPGAs ofrecen una flexibilidad inigualable en comparación con otros dispositivos de hardware, ya que permiten reconfigurar el diseño del circuito según las necesidades del proyecto. Esta reconfigurabilidad no solo es útil para prototipos, sino que también es esencial para aplicaciones en el mundo real donde las necesidades pueden cambiar con el tiempo.

Aplicación práctica de conceptos teóricos: A través del desarrollo del proyecto, pudimos aplicar conceptos teóricos relacionados con la generación y división de frecuencias, así como el manejo de dispositivos de salida como los LEDs. Esta experiencia reafirma la importancia de combinar la teoría con la práctica, ya que proporciona una comprensión más profunda y tangible de los conceptos.

Importancia de la optimización: Aunque el proyecto se centró principalmente en el manejo de LEDs, es esencial considerar la optimización en cualquier diseño de FPGA. La optimización puede referirse a la utilización de recursos, el consumo de energía o la velocidad de operación. En futuros desarrollos, se debe prestar atención a estas métricas para garantizar un diseño eficiente.

Profundidad en la comprensión de los componentes involucrados: El PLL y el divisor de frecuencia son dos componentes esenciales en muchos sistemas basados en FPGA. Al implementarlos en este proyecto, ganamos una comprensión más profunda de cómo funcionan y cómo pueden ser configurados para satisfacer las necesidades específicas del diseño.

El papel crucial de la documentación: Durante todo el proceso de desarrollo, es fundamental mantener una documentación adecuada. Esta no solo ayuda en las etapas de depuración y validación, sino que también es esencial para cualquier persona que pueda trabajar o modificar el diseño en el futuro.

Valor del aprendizaje práctico: Finalmente, este proyecto reafirma la importancia del aprendizaje práctico en el campo de la electrónica y el diseño de hardware. Aunque los conceptos teóricos son fundamentales, es a través de la implementación y la experimentación práctica que los ingenieros pueden realmente comprender y dominar su campo.

Referencias

[1] Best, R. E. "Phase-Locked Loops: Design, Simulation, and Applications". 6th ed., McGraw-Hill, 2007.
[2] Verilog HDL: A Guide to Digital Design and Synthesis, Samir Palnitkar, 2nd Edition, Prentice Hall, 2003.
[3] "Digital Design: An Embedded Systems Approach Using Verilog", Peter J. Ashenden, Elsevier, 2008.