



## CORDIC modo rotador

Circuitos lógicos programables

Escrito por:

Karen Tatiana Zamudio

## Revisión A

Nicolas Álvarez

Universidad de Buenos Aires

11 abril 2024

# Historial de cambios

Cuadro 1: Registro de Revisiones

Revisión	Detalles de los cambios realizados	Fecha
0	Creación del documento	11 de abril 2024
1	Entrega	23 de abril 2024

# Circuitos lógicos programables

## Tabla de contenido

<b>1. Introducción</b>	<b>5</b>
1.1. Objetivos . . . . .	5
1.2. Alcance . . . . .	6
<b>2. CORDIC</b>	<b>6</b>
2.1. CORDIC trigonométrico . . . . .	6
2.2. CORDIC vectorial . . . . .	8
2.3. CORDIC logarítmico . . . . .	8
<b>3. Implementación CORDIC rotator</b>	<b>9</b>
3.1. Configuración de Tiempo y Declaración de Módulo . . . . .	9
3.2. Declaración de la Tabla <code>atan_table</code> . . . . .	10
3.3. Inicialización de registros y asignación de cuadrante . . . . .	10
3.3.1. Cuadrante I . . . . .	10
3.3.2. Cuadrante II . . . . .	11
3.3.3. Cuadrante III . . . . .	11
3.3.4. Cuadrante IV . . . . .	12
3.4. Banco de pruebas . . . . .	12
3.4.1. Parámetros y variables locales . . . . .	12
3.4.2. Inicialización . . . . .	13
3.4.3. Generación de formas de onda . . . . .	13
3.4.4. Generación del reloj . . . . .	14
3.4.5. Salida del módulo . . . . .	14
3.5. Simulación . . . . .	14
3.6. Simulación en vivado . . . . .	15

---

3.7. Esquema . . . . .	16
3.8. Generación del Archivo .bit . . . . .	16

## 1. Introducción

En el contexto del diseño digital y la implementación de algoritmos, el presente documento aborda la creación y análisis de un módulo de rotación basado en el algoritmo CORDIC (Coordinate Rotation Digital Computer). El algoritmo CORDIC es ampliamente utilizado en sistemas digitales para calcular funciones trigonométricas y transformaciones de coordenadas de manera eficiente.

Este documento proporcionará una visión general del diseño del módulo cordic rotator, destacando su funcionamiento, estructura interna y su implementación en una FPGA (Field-Programmable Gate Array). Además, se presentarán los resultados de simulaciones y se analizará el uso de recursos de la FPGA.

### 1.1. Objetivos

Analizar y documentar el diseño, implementación y funcionamiento del módulo cordic rotator basado en el algoritmo CORDIC, así como evaluar su eficiencia y aplicabilidad en sistemas digitales.

1. Describir el algoritmo CORDIC y su aplicación en rotación de coordenadas: proporcionar una explicación detallada del algoritmo CORDIC y cómo se utiliza para realizar rotaciones de coordenadas cartesianas en el diseño del módulo cordic rotator.
2. Implementar el módulo cordic rotator en Verilog: crear una implementación en Verilog del módulo cordic rotator, asegurando que el código refleje correctamente la lógica del algoritmo CORDIC y que sea adecuado para su uso en sistemas digitales.
3. Realizar simulaciones del módulo y analizar su comportamiento: ejecutar simulaciones del módulo cordic rotator en un entorno de simulación, utilizando diferentes casos de prueba para evaluar su funcionamiento en una variedad de escenarios. Analizar los resultados de las simulaciones para verificar la precisión y eficacia del diseño.
4. Evaluar el uso de recursos de la FPGA: realizar un análisis exhaustivo del uso de recursos de la FPGA por parte del diseño del módulo cordic rotator, incluyendo el número de LUTs, FFs y otros recursos utilizados. Identificar áreas de mejora en términos de eficiencia y optimización del diseño.

## 1.2. Alcance

El alcance de este documento abarca la descripción detallada del diseño e implementación del módulo cordic rotator utilizando el algoritmo CORDIC en Verilog. Se incluirá una explicación exhaustiva del funcionamiento del algoritmo CORDIC y su aplicación en la rotación de coordenadas cartesianas. Además, se proporcionará el código Verilog del módulo cordic rotator junto con explicaciones detalladas de su estructura interna y su funcionamiento.

Se realizarán simulaciones del módulo en un entorno adecuado para evaluar su comportamiento en diversos escenarios, y se analizarán los resultados obtenidos. Asimismo, se llevará a cabo un análisis del uso de recursos de la FPGA por parte del diseño, con el objetivo de evaluar su eficiencia y escalabilidad en sistemas digitales.

## 2. CORDIC

CORDIC (Coordinate Rotation Digital Computer) es un algoritmo utilizado para calcular funciones trigonométricas, como seno, coseno, arcotangente, y para realizar rotaciones y transformaciones de coordenadas de manera eficiente en sistemas digitales. Fue desarrollado en la década de 1950 por Jack E. Volder en Bell Labs.

El algoritmo CORDIC se basa en una serie de rotaciones y desplazamientos sucesivos que permiten aproximar funciones trigonométricas y realizar operaciones de rotación con un bajo costo computacional. Una de las características clave del algoritmo CORDIC es su capacidad para realizar estas operaciones utilizando únicamente operaciones de suma, resta y desplazamiento bit a bit, lo que lo hace adecuado para implementaciones en hardware digital.

El algoritmo CORDIC es especialmente útil en aplicaciones donde se requieren cálculos trigonométricos rápidos y precisos, como en sistemas de procesamiento de señales, sistemas de comunicación digital, gráficos por computadora, entre otros. Además, es altamente versátil y puede adaptarse para calcular una amplia gama de funciones trigonométricas y realizar diversas operaciones geométricas.

### 2.1. CORDIC trigonométrico

El algoritmo CORDIC (Coordinate Rotation Digital Computer) Trigonométrico es un método iterativo utilizado para calcular funciones trigonométricas como el seno ( $\sin$ ), coseno

(cos), arcoseno (arcsin) y arcocoseno (arc cos). Funciona mediante iteraciones de rotación y desplazamiento, donde cada iteración aproxima gradualmente el valor deseado de la función trigonométrica. El algoritmo se basa en la representación de un ángulo en coordenadas cartesianas  $(X, Y)$  y utiliza rotaciones sucesivas para acercarse al ángulo objetivo. A medida que las iteraciones continúan, el valor de salida converge hacia el valor correcto con una precisión determinada.

En cada iteración del algoritmo CORDIC Trigonométrico, se realiza una rotación y un desplazamiento de las coordenadas  $(X, Y)$ . La rotación se realiza mediante el ajuste de los ángulos de las coordenadas, lo que permite acercarse al ángulo objetivo. El desplazamiento se utiliza para ajustar las magnitudes de las coordenadas y garantizar que la convergencia del algoritmo sea adecuada. Estas iteraciones se repiten hasta que se alcanza la precisión deseada en el cálculo de la función trigonométrica.

Por ejemplo, para calcular el seno de un ángulo dado utilizando el algoritmo CORDIC Trigonométrico, se inicializan las coordenadas  $(X, Y)$  con valores apropiados y se realizan iteraciones de rotación y desplazamiento hasta que se alcanza la precisión deseada. El valor de  $Y$  al final de las iteraciones representa el valor aproximado del seno del ángulo dado. De manera similar, se pueden calcular el coseno, el arcoseno y el arcocoseno utilizando adaptaciones del algoritmo CORDIC Trigonométrico.

La precisión y la convergencia del algoritmo CORDIC Trigonométrico dependen del número de iteraciones realizadas y del número de bits utilizados para representar los valores de las coordenadas. En general, el algoritmo converge rápidamente y proporciona resultados precisos con un número suficiente de iteraciones. Sin embargo, la precisión puede verse afectada por la representación finita de los números en sistemas digitales y por la resolución de las operaciones de rotación y desplazamiento.

El algoritmo CORDIC trigonométrico se utiliza en una amplia gama de aplicaciones en sistemas digitales, incluyendo procesamiento de señales, comunicaciones, gráficos por computadora, sistemas de navegación y más. Se utiliza para calcular funciones trigonométricas de manera eficiente y precisa en sistemas donde se requieren cálculos frecuentes de ángulos y funciones trigonométricas. Su implementación en hardware digital es especialmente adecuada debido a su simplicidad y eficiencia en términos de recursos computacionales.

Las ecuaciones para las iteraciones de rotación y desplazamiento en el algoritmo CORDIC Trigonométrico pueden ser expresadas como:



$$\begin{aligned}
X_{i+1} &= X_i - \delta_i \cdot Y_i \cdot 2^{-i}, \\
Y_{i+1} &= Y_i + \delta_i \cdot X_i \cdot 2^{-i}, \\
Z_{i+1} &= Z_i - \delta_i \cdot \operatorname{atanh}(2^{-i}),
\end{aligned} \tag{1}$$

donde  $i$  es el índice de iteración,  $\delta_i$  es el signo de la rotación en la iteración  $i$ ,  $X_i$ ,  $Y_i$  son las coordenadas en la iteración  $i$ , y  $Z_i$  es el ángulo acumulado hasta la iteración  $i$ .

## 2.2. CORDIC vectorial

El algoritmo CORDIC Vectorial utiliza un enfoque similar al CORDIC Trigonométrico, pero se enfoca en manipular vectores en lugar de solo ángulos. Permite realizar rotaciones y transformaciones de coordenadas en sistemas de coordenadas cartesianas o polares de manera eficiente.

En un sistema de coordenadas cartesianas, las operaciones del CORDIC Vectorial pueden utilizarse para rotar un vector en el plano XY. Por otro lado, en un sistema de coordenadas polares, el algoritmo puede utilizarse para convertir coordenadas cartesianas a polares y viceversa, así como para realizar rotaciones de vectores en el plano radial.

La eficiencia y precisión del CORDIC Vectorial lo hacen adecuado para una variedad de aplicaciones en sistemas digitales, como procesamiento de señales, modulación y demodulación en sistemas de comunicaciones, y transformaciones geométricas en gráficos por computadora. Además, su estructura iterativa permite su implementación eficiente en hardware digital, lo que lo convierte en una opción popular en sistemas embebidos y sistemas de tiempo real donde los recursos son limitados.

## 2.3. CORDIC logarítmico

El algoritmo CORDIC logarítmico utiliza una serie de iteraciones para aproximar funciones logarítmicas y exponenciales, así como operaciones relacionadas como la raíz cuadrada y el exponente. A través de un proceso iterativo, el algoritmo calcula las aproximaciones de estas funciones con una precisión deseada.

Una de las características clave del CORDIC logarítmico es su capacidad para calcular estas funciones de manera eficiente y precisa utilizando un enfoque iterativo y estructuras de datos simples. Esto lo hace adecuado para su implementación en sistemas digitales donde se requieren cálculos de funciones no trigonométricas con recursos limitados.

El CORDIC logarítmico tiene una amplia gama de aplicaciones en sistemas digitales, incluyendo procesamiento de señales, compresión de datos, criptografía y más. Se utiliza en situaciones donde se requiere el cálculo rápido y preciso de funciones logarítmicas y exponenciales, así como operaciones relacionadas como la raíz cuadrada y el exponente. Su eficiencia y precisión lo hacen valioso en una variedad de aplicaciones de sistemas digitales.

### 3. Implementación CORDIC rotator

En esta sección, se detalla la implementación del módulo CORDIC Rotator en Verilog. El módulo CORDIC Rotator es una implementación hardware del algoritmo CORDIC (Coordinate Rotation Digital Computer), diseñado para realizar rotaciones de vectores en el plano cartesiano. El algoritmo CORDIC se utiliza ampliamente en sistemas digitales para calcular funciones trigonométricas, transformaciones de coordenadas y otras operaciones matemáticas de manera eficiente. En esta sección, se explicará paso a paso la implementación del módulo CORDIC rotator, incluyendo la configuración de parámetros, la definición de la tabla de arco tangente, las iteraciones del algoritmo y la salida del módulo. Cada parte de la implementación se analizará en detalle, acompañada de su correspondiente código en Verilog.

#### 3.1. Configuración de Tiempo y Declaración de Módulo

El código comienza con la configuración del `timescale` y la definición de parámetros utilizados en el módulo. El parámetro `c_parameter` determina el ancho de bits de los datos de entrada y salida, mientras que `STG` se utiliza para definir el ancho de bits de los vectores `X` e `Y`. Luego, se declara el módulo `cordic_rotator` con sus entradas (`clock`, `angle`, `Xin`, `Yin`) y salidas (`Xout`, `Yout`).

```
'timescale 1ns/100ps
```

```
module cordic_rotator (
    input clock ,
    input signed [31:0] angle ,
    input signed [c_parameter - 1:0] Xin ,
    input signed [c_parameter - 1:0] Yin ,
    output signed [c_parameter:0] Xout ,
```

```

    output signed [c_parameter:0] Yout
);

```

```

parameter c_parameter = 16;
localparam STG = c_parameter;

```

### 3.2. Declaración de la Tabla atan\_table

La tabla `atan_table` contiene los valores de la función arco tangente utilizados en el algoritmo CORDIC. Estos valores están precalculados y se utilizan para realizar las operaciones de rotación en el módulo. La tabla está definida como un arreglo de 31 bits con 31 entradas, cada una representando un ángulo específico.

```

wire signed [31:0] atan_table [0:30];
assign atan_table[00] = 32'b00100000000000000000000000000000;
// Resto de asignaciones de la tabla atan_table...

```

### 3.3. Inicialización de registros y asignación de cuadrante

Se inicializan los registros `X`, `Y` y `Z`, que se utilizan para almacenar los datos en cada etapa del algoritmo CORDIC. Luego, se calcula el cuadrante en el que se encuentra el ángulo de rotación para determinar cómo se deben tratar los datos de entrada.

#### 3.3.1. Cuadrante I

En el cuadrante I, el ángulo de rotación se encuentra en el rango de 0 a  $\pi/2$ . En este caso, no es necesario realizar ninguna modificación en los datos de entrada, ya que están en el rango adecuado para ser procesados por el algoritmo CORDIC. Por lo tanto, los valores de las coordenadas `X` e `Y` del vector de entrada (`Xin` y `Yin`) se asignan directamente a los registros `X[0]` y `Y[0]`, respectivamente. Además, el ángulo de rotación (`angle`) se asigna directamente al registro `Z[0]`.

```

case (quadrant)
    2'b00: begin // Cuadrante I
        X[0] <= Xin;

```

```

Y[0] <= Yin ;
Z[0] <= angle ;

```

Este caso se activa cuando los bits de signo del ángulo son 0, indicando que el ángulo se encuentra en el primer cuadrante.

### 3.3.2. Cuadrante II

En el cuadrante II, el ángulo de rotación se encuentra en el rango de  $\pi/2$  a  $\pi$ . Para ajustar los datos de entrada al rango adecuado para el algoritmo CORDIC, se realiza una pre-rotación del ángulo. En este caso, el ángulo se resta de  $\pi$  para asegurar que esté dentro del rango de  $-\pi/2$  a 0. Los valores de las coordenadas X e Y del vector de entrada se asignan a los registros X[0] y Y[0], respectivamente, de la misma manera que en el cuadrante I.

```

2'b01: begin // Cuadrante II
X[0] <= Xin ;
Y[0] <= Yin ;
Z[0] <= 32'b10000000000000000000000000000000 - angle ;

```

Este caso se activa cuando el bit de signo más significativo del ángulo es 0 y el siguiente bit es 1, indicando que el ángulo se encuentra en el segundo cuadrante.

### 3.3.3. Cuadrante III

En el cuadrante III, el ángulo de rotación se encuentra en el rango de  $-\pi$  a  $-\pi/2$ . Al igual que en el cuadrante II, se realiza una pre-rotación del ángulo restando  $\pi$  para asegurar que esté dentro del rango de  $-\pi/2$  a 0. Los valores de las coordenadas X e Y del vector de entrada se asignan a los registros X[0] y Y[0], respectivamente, de la misma manera que en los cuadrantes I y II.

```

2'b10: begin // Cuadrante III
X[0] <= Xin ;
Y[0] <= Yin ;
Z[0] <= angle - 32'b10000000000000000000000000000000 ;

```

Este caso se activa cuando el bit de signo más significativo del ángulo es 1 y el siguiente bit es 0, indicando que el ángulo se encuentra en el tercer cuadrante.

### 3.3.4. Cuadrante IV

En el cuadrante IV, el ángulo de rotación se encuentra en el rango de  $-\pi/2$  a 0. Para ajustar los datos de entrada al rango adecuado para el algoritmo CORDIC, se realiza una pre-rotación del ángulo restando  $2\pi$  para asegurar que esté dentro del rango de  $-\pi/2$  a  $-\pi$ . Los valores de las coordenadas X e Y del vector de entrada se asignan a los registros `X[0]` y `Y[0]`, respectivamente, de la misma manera que en los cuadrantes I, II y III.

```
2'b10: begin // Cuadrante III
        X[0] <= Xin;
        Y[0] <= Yin;
        Z[0] <= angle - 32'b10000000000000000000000000000000;
```

Este caso se activa cuando los bits de signo del ángulo son 1, indicando que el ángulo se encuentra en el cuarto cuadrante.

## 3.4. Banco de pruebas

El banco de pruebas proporcionado se encarga de simular el módulo `cordic_rotator` y generar formas de onda de entrada para probar su funcionalidad. Aquí tienes una explicación detallada de su funcionamiento:

### 3.4.1. Parámetros y variables locales

Se definen algunos parámetros y variables locales para el tamaño de los datos de entrada (`SZ`), los valores booleanos (`FALSE` y `TRUE`), y el valor de la señal de entrada (`VALUE`), que se reduce para ajustarse a la ganancia del sistema.

```
localparam SZ = 16;
```

```
reg [SZ-1:0] Xin, Yin;
```

```
reg [31:0] angle;
```

```
wire [SZ:0] Xout, Yout;
```

```
reg CLK_100MHZ;
```

```
localparam FALSE = 1'b0;
```

```
localparam TRUE = 1'b1;
```

```
// reducido por un factor de 1.647 ya que es la ganancia del sistema (k)
```

```
localparam VALUE = 32000/1.647;
```

### 3.4.2. Inicialización

Antes de comenzar la simulación, se inicializan las variables y las señales de entrada. Se establece **start** en **FALSE**, se inicializa la señal de reloj **CLK\_100MHZ**, el ángulo en cero, y se establece un valor inicial para **Xin** y **Yin**.

```
initial
```

```
begin
```

```
    start = FALSE;
```

```
    CLK_100MHZ = 1'b0;
```

```
    angle = 0;
```

```
    Xin = VALUE;
```

```
    Yin = 1'd0;
```

```
    i = 0;
```

```
    #1000;
```

```
    @(posedge CLK_100MHZ);
```

```
    start = TRUE;
```

### 3.4.3. Generación de formas de onda

Durante la simulación, se genera una señal de reloj de 100 MHz (**CLK\_100MHZ**). Después de un breve período de espera, se activa **start** para indicar que la simulación comienza y se comienza a generar una secuencia de ángulos para probar el módulo **cordic\_rotator**.

```
for (i = 0; i < 360; i = i + 1)
```

```
begin
```

```
    @(posedge CLK_100MHZ);
```

```
    #11700
```

```
    start = FALSE;
```

```

    angle = ((1 << 32)*(i+1))/360;
end

```

#### 3.4.4. Generación del reloj

Se genera una señal de reloj de 100 MHz (CLK\_100MHZ) con una duración de tiempo determinada por CLK100\_SPEED.

```

parameter CLK100_SPEED = 10; // 100Mhz = 10nS

```

```

initial
begin
    CLK_100MHZ = 1'b0;
    $display("CLK_100MHZ iniciado");
    #5;
    forever
    begin
        #(CLK100_SPEED/2) CLK_100MHZ = 1'b1;
        #(CLK100_SPEED/2) CLK_100MHZ = 1'b0;
    end
end

```

#### 3.4.5. Salida del módulo

Finalmente, se asignan las salidas **Xout** y **Yout** del módulo con los valores calculados después de completar todas las iteraciones del algoritmo CORDIC. Estas salidas representan las coordenadas rotadas del vector de entrada.

```

assign Xout = X[STG - 1];
assign Yout = Y[STG - 1];

```

### 3.5. Simulación

En este documento, se describe el proceso para realizar la simulación del diseño en Vivado y posteriormente generar el archivo .bit que se utilizará para la programación de la FPGA.

### 3.6. Simulación en vivado

Para realizar la simulación en Vivado, sigue estos pasos:

1. **Crear un proyecto:** Abre Vivado y crea un nuevo proyecto. Selecciona la opción

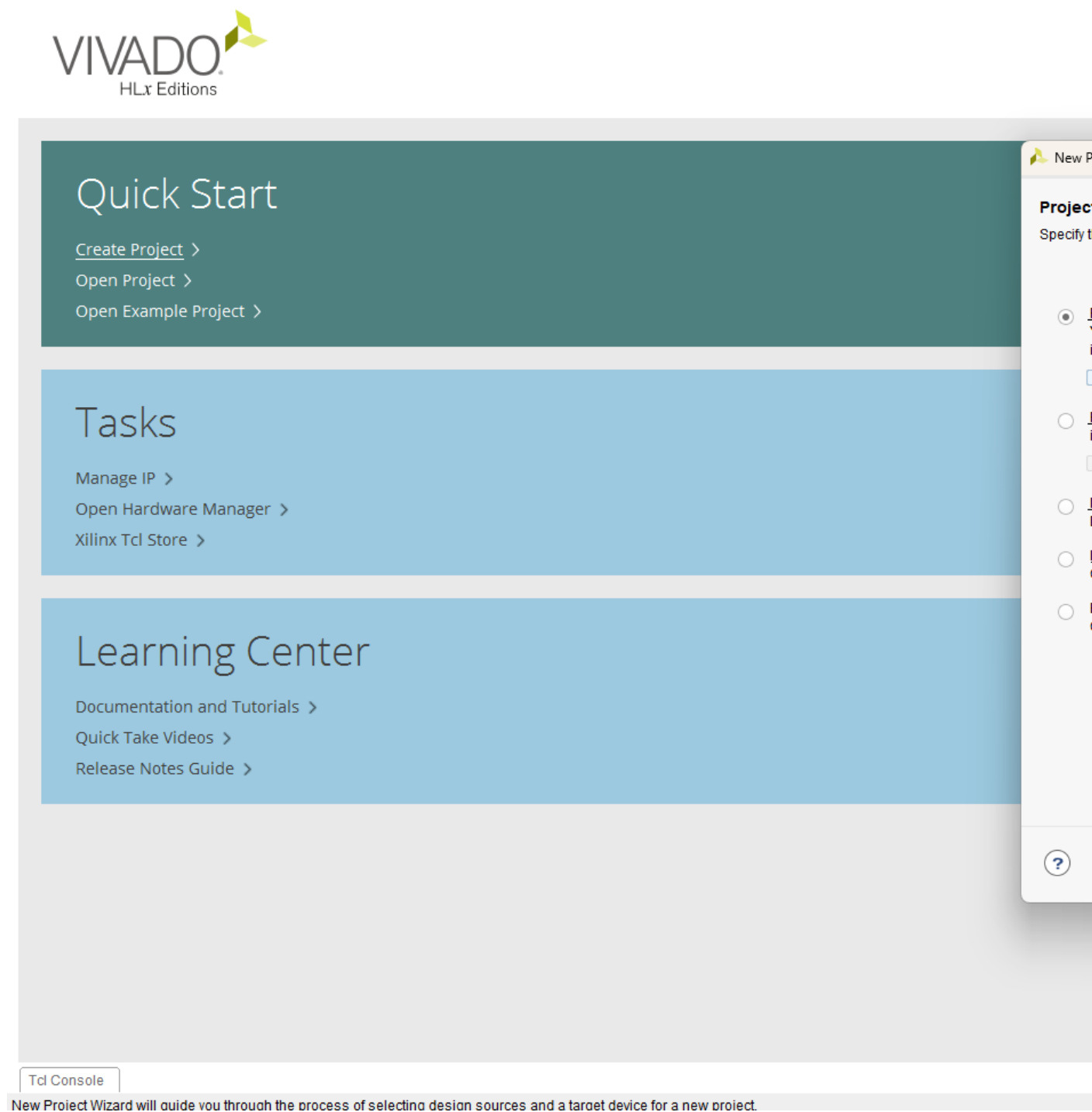


Figura 1: Crear nuevo proyecto en Vivado

2. **Configurar simulación:** una vez que hayas creado el proyecto. Ve a la pestaña "Flow Navigator" selecciona "Simulation". Aquí puedes configurar las opciones de simulación,



como el tipo de simulador a utilizar y los archivos de simulación a incluir.

3. **Ejecutar la simulación:** después de configurar la simulación, ejecútala haciendo clic en el botón "Run Simulation". Vivado ejecutará la simulación y generará resultados que podrás analizar.

### 3.7. Esquema

El esquema utilizado para conectar las entradas y salidas del rotador CORDIC durante la simulación en Vivado se muestra en la Figura 6.

4. **Análisis de Resultados:** Una vez completada la simulación, analiza los resultados en Vivado. Puedes revisar las formas de onda de las señales de entrada y salida, así como cualquier mensaje de advertencia o error que pueda haberse generado durante la simulación.

Con estos pasos, habrás completado tanto la simulación del diseño en Vivado como la generación del archivo .bit para programar la FPGA.

### 3.8. Generación del Archivo .bit

Una vez que hayas completado y verificado la simulación en Vivado, puedes proceder a generar el archivo .bit para programar la FPGA. Sigue estos pasos:

1. **Implementación del Diseño:** En Vivado, ve a la pestaña "Flow Navigator" selecciona "Implementation". Aquí, ejecuta el proceso de implementación para sintetizar y asignar el diseño a los recursos de la FPGA.
2. **Generación del Bitstream:** Después de completar la implementación, ve a la pestaña "Generate Bitstream" en Vivado. Ejecuta este proceso para generar el archivo .bit, que contiene la configuración del diseño para la FPGA.
3. **Programación de la FPGA:** Una vez generado el archivo .bit, puedes utilizarlo para programar la FPGA. Conecta la FPGA a tu computadora y utiliza el software de programación de Vivado para cargar el archivo .bit en la FPGA.

Con estos pasos, habrás completado tanto la simulación del diseño en Vivado como la generación del archivo .bit para programar la FPGA.

**Nota:** El trabajo desarrollado se realizó sobre una FPGA de la firma Xilinx. La placa de desarrollo utilizada es la Arty Z7-10 de la empresa Digilent.

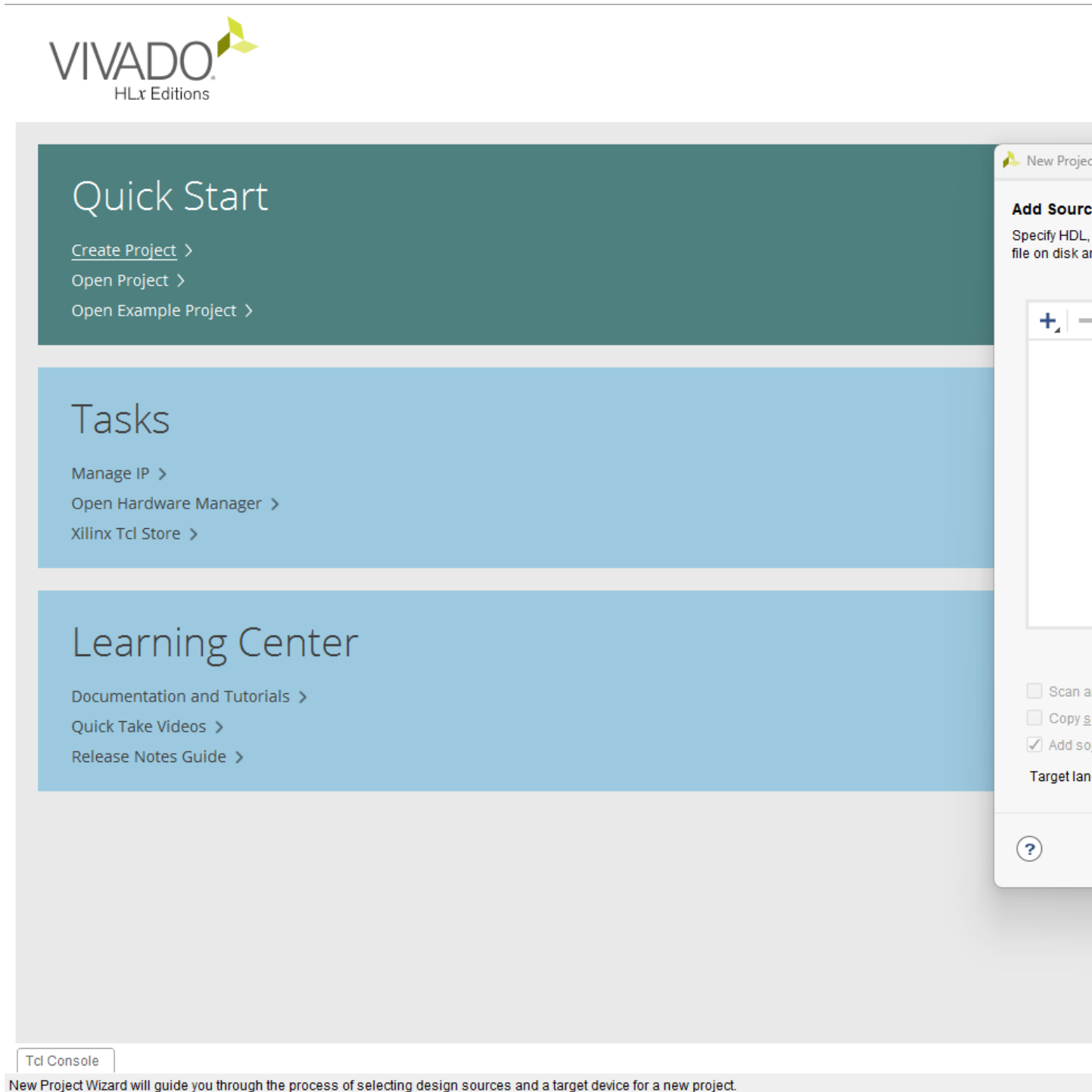


Figura 2: IMportacion de archivos fuente

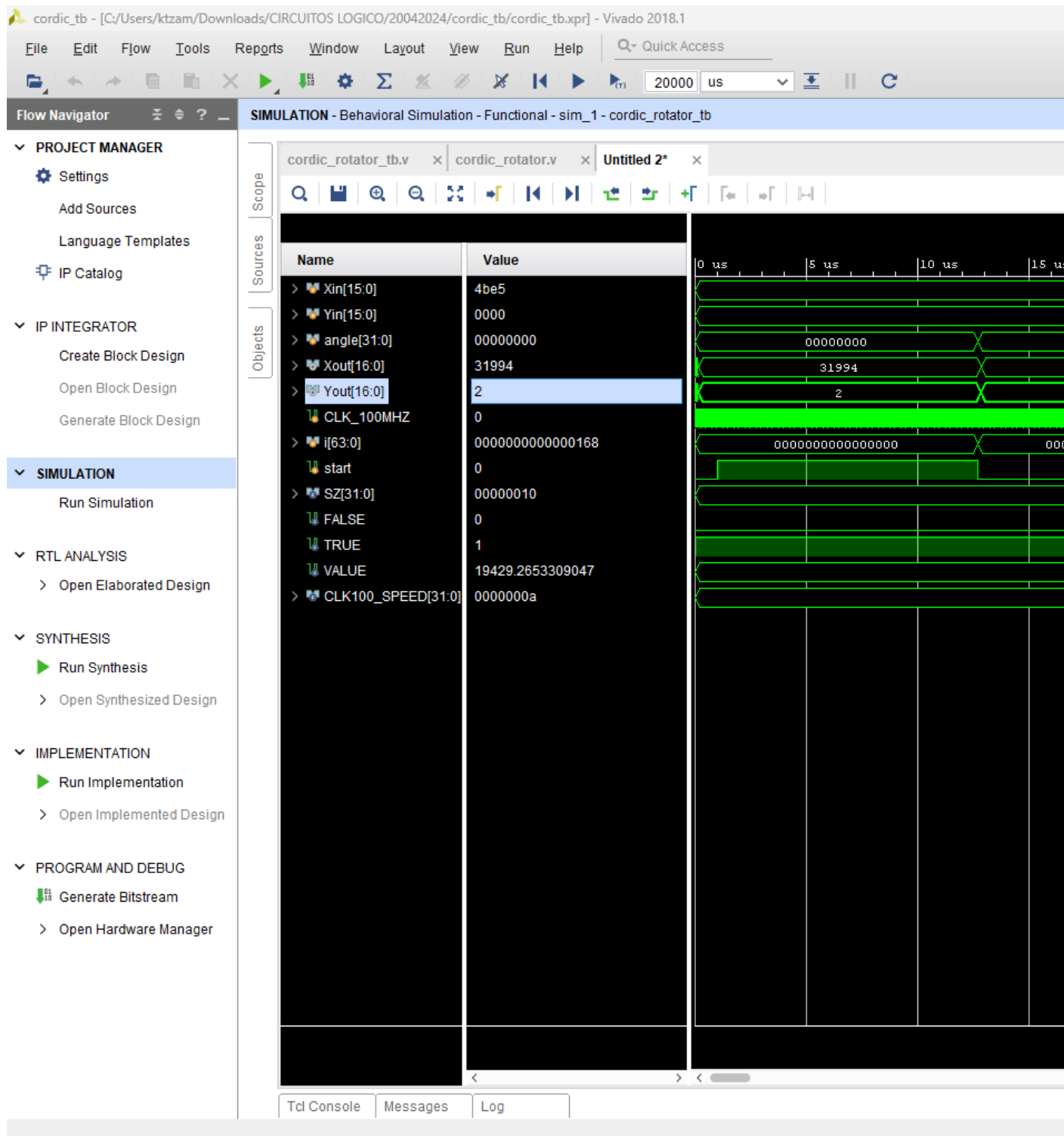


Figura 3: Esquema utilizado para la simulación en Vivado

Figura 4: Esquema utilizado para implementación de Cordic

Ángulo =	0, 00b60b60, Xout = 32765, Yout = 2
Ángulo =	1, 016c16c1, Xout = 32757, Yout = 574
Ángulo =	2, 02222222, Xout = 32743, Yout = 1146
Ángulo =	3, 02d82d82, Xout = 32718, Yout = 1714
Ángulo =	4, 038e38e3, Xout = 32685, Yout = 2288
Ángulo =	5, 04444444, Xout = 32639, Yout = 2857
Ángulo =	6, 04fa4fa4, Xout = 32585, Yout = 3425
Ángulo =	7, 05b05b05, Xout = 32522, Yout = 3993
Ángulo =	8, 06666666, Xout = 32445, Yout = 4560
Ángulo =	9, 071c71c7, Xout = 32362, Yout = 5124
Ángulo =	10, 07d27d27, Xout = 32266, Yout = 5690
Ángulo =	11, 08888888, Xout = 32163, Yout = 6251
Ángulo =	12, 093e93e9, Xout = 32046, Yout = 6815
Ángulo =	13, 09f49f49, Xout = 31924, Yout = 7368
Ángulo =	14, 0aaaaaaa, Xout = 31790, Yout = 7927
Ángulo =	15, 0b60b60b, Xout = 31651, Yout = 8478
Ángulo =	16, 0c16c16c, Xout = 31493, Yout = 9034

Figura 5: Esquema utilizado para implementación de Cordic

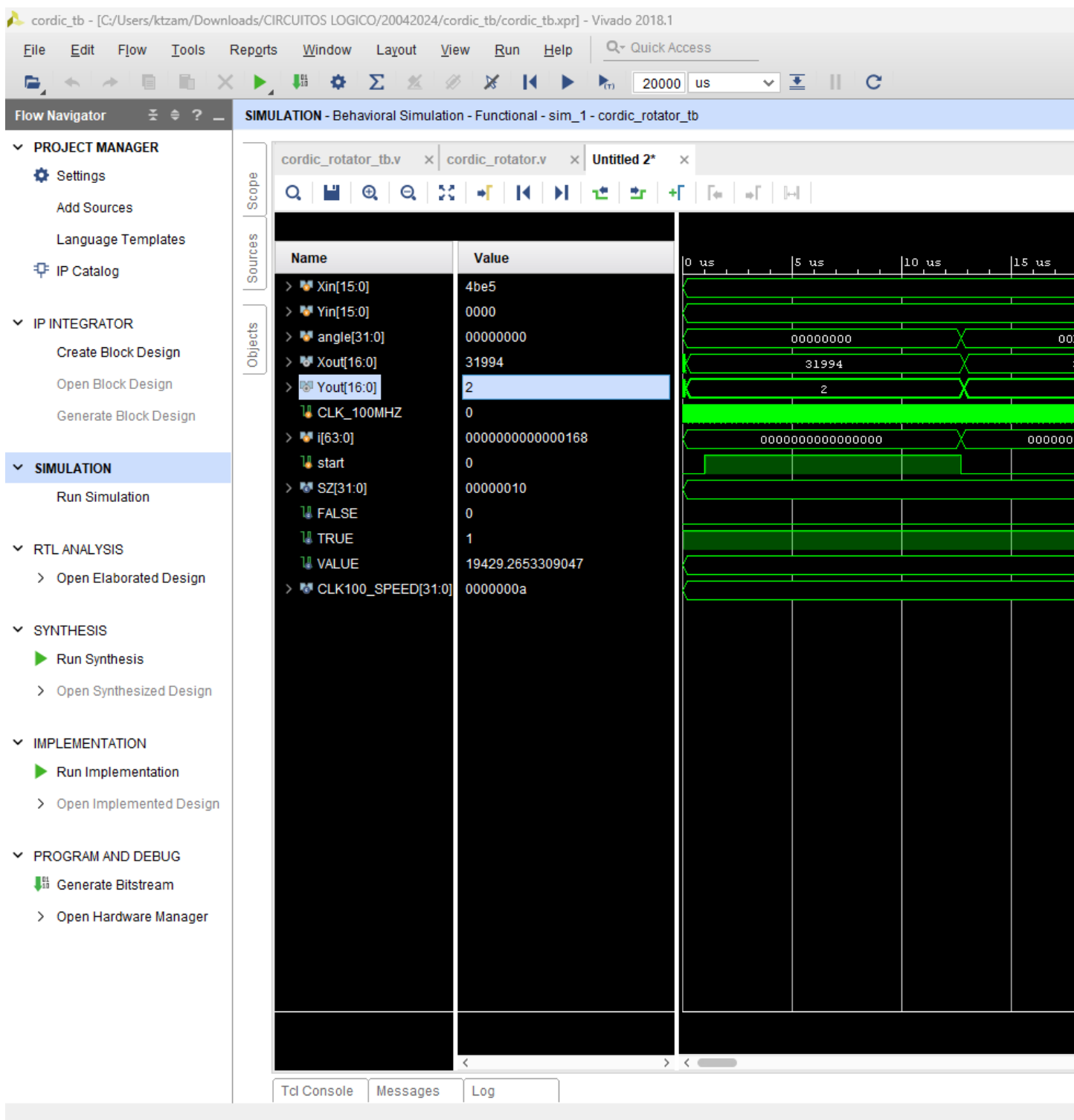


Figura 6: Esquema utilizado para la simulación en Vivado