Universidad de Buenos Aires
Facultad de Ingeniería

Especialización en Microelectrónica

Arquitectura de Sistemas Digitales

Algoritmo CORDIC

Alumno: Agustín Galdeman
Fecha: 11 de mayo de 2025

Buenos Aires, Argentina

1. CORDIC de Rotación y Vectorización con Iteraciones Configurables

Este núcleo implementa el algoritmo **CORDIC** (Coordinate Rotation DIgital Computer) en coordenadas cartesianas, soportando:

- **Modo rotación** (`rot0_vec1 = 0`) : aplica un ángulo z_0 a un vector (x_0, y_0).
- **Modo vectorización** (`rot0_vec1 = 1`) : calcula módulo y ángulo de (x_0, y_0).
- **Iteraciones configurables** $1 \ldots 16$ seleccionadas en tiempo de ejecución mediante `n_iter`.

Los datos usan formato Q2.16 signed de punto fijo.

Objetivo Matemático

$$\text{Rotación:} \quad (x_0, y_0) \xrightarrow{z_0} (x_n, y_n) = K_n \begin{pmatrix} \cos z_0 \\ \sin z_0 \end{pmatrix}$$

$$\text{Vectorización:} \quad (x_0, y_0) \longrightarrow (x_n, 0, z_n), \qquad z_n = \operatorname{atan2}(y_0, x_0) \pm \pi$$

K_n es el factor de escala CORDIC ($\approx 0{,}824$ a 16 iteraciones).

Pre–acondicionamiento (sólo vectorización)

Para asegurar convergencia se rota previamente π si $x_0 < 0$:

$$(x_0', y_0', z_0') = \begin{cases} (-x_0, -y_0, z_0 + \pi) & y_0 > 0 \\ (-x_0, -y_0, z_0 - \pi) & y_0 < 0 \end{cases}$$

Algoritmo Iterativo

Para cada iteración i $(0 \le i < n_{\text{iter}})$:

$$d_i = \begin{cases} \operatorname{sign}(z_i) & \text{rotación} \\ \operatorname{sign}(y_i) & \text{vectorización} \end{cases}$$

$$x_{i+1} = x_i - d_i \, (y_i \gg i)$$

$$y_{i+1} = y_i + d_i \, (x_i \gg i)$$

$$z_{i+1} = z_i - d_i \, \varphi_i$$

donde $\varphi_i = \arctan(2^{-i})$ està precargado en una LUT de 16 entradas.

2. Descripción de los Módulos

`cordic_stage.v` – etapa combinacional

```
wire sign_z = z_i[17];
wire sign_y = y_i[18];
wire di = rot0_vec1 ? sign_y : sign_z;

assign x_ip1 = di ? x_i + (y_i >>> i) : x_i - (y_i >>> i);
assign y_ip1 = di ? y_i - (x_i >>> i) : y_i + (x_i >>> i);
assign z_ip1 = di ? z_i + phi_i       : z_i - phi_i;
```

Registros relevantes en `cordic_top.v`

```
reg  [18:0] x_q, y_q;   // vector iterativo
reg  [17:0] z_q;        //  ngulo  restante
reg  [ 4:0] i;          // contador de iteraciones
reg         busy, done; // FSM flags
```

FSM de Control

S_IDLE espera `start`.

S_RUN itera mientras $i < n\_iter$.

S_DONE genera pulso `done` y vuelve a IDLE.

3. Ejemplo Numérico

Rotación $(1, 0)$ por $\pi/16$ rad con $n_{iter} = 16$:

$$x_{16} = 0{,}8076 \quad (\approx K \cos \frac{\pi}{16})$$
$$y_{16} = 0{,}1607 \quad (\approx K \sin \frac{\pi}{16})$$
$$z_{16} \approx 0$$

Vectorización $(-0{,}6, -0{,}8)$:

$$(x_{16}, y_{16}) \approx (-1{,}468, -0{,}748), \; z_{16} \approx 0{,}927 \text{ rad}$$

Restando la ganancia $1/K$ se recupera módulo $\approx 1{,}0$.

4. Testbench y Validación

Arquitectura

- Clock de 100 MHz (`#5 clk = ~clk;`).
- Dos casos dirigidos: rotación y vectorización.
- Espera `done`; imprime resultados en entero y flotante.
- Timeout de 1µs para evitar bucles infinitos.

Compilación y Ejecución (Icarus Verilog)

```
iverilog -g2005-sv -o cordic_tb  \
         tb_cordic.v cordic_top.v cordic_stage.v
vvp cordic_tb
```

Genera `cordic.vcd`. Con **GTKWave** se visualizan las señales, aplicando *BitsToReal* (escala 65536) para leer los valores como números reales.

Resultados Observados

Cuadro 1: Valores Q2.16 al finalizar cada prueba

Prueba	x_n	y_n	z_n
Rotación	52924	10527	1
Vectorización	-96159	-48975	58343

Rotación: $52924/65536 = 0{,}8076$, Vectorización: $58343/65536 = 0{,}890$ rad $\approx 0{,}927 - K\_\varphi$

Ambos casos cumplen el error máximo de ± 2 LSB.