Resumo CORDIC (para aplicar em VHDL)

1. Modo Rotação

   1.1. Entradas:
   - modo $= 0$
   - start $= 1$
   - $x_{\text{in}} =$ tanto faz
   - $y_{\text{in}} =$ tanto faz
   - $z_{\text{in}} =$ ângulo z

   1.2. Setup dos valores internos (antes de iterar):
   - done $= 0$
   - $x = 0.6072529351$
   - $y = 0$
   - $z = z_{\text{in}}$
   - $i = 0$

   1.3. Processo iterativo:

   if $z_i \geq 0$:

   $$x_{i+1} = x_i - (y_i \gg i)$$
   $$y_{i+1} = y_i + (x_i \gg i)$$
   $$z_{i+1} = z_i - \text{lista\_arctans}(i)$$

   else :

   $$x_{i+1} = x_i + (y_i \gg i)$$
   $$y_{i+1} = y_i - (x_i \gg i)$$
   $$z_{i+1} = z_i + \text{lista\_arctans}(i)$$

   $$i = i + 1$$

   1.4. Parar processo iterativo quando $i = 16$

   1.5. Saídas:
   - done $= 1$
   - $x_{\text{out}} = \cos z$
   - $y_{\text{out}} = \sin z$
   - $z_{\text{out}} \approx 0$

2. Modo Vetorização

   2.1. Entradas:
   - modo $= 1$
   - start $= 1$
   - $x_{\text{in}} =$ valor x da coordenada
   - $y_{\text{in}} =$ valor y da coordenada
   - $z_{\text{in}} =$ tanto faz

   2.2. Setup dos valores internos (antes de iterar):
   - done $= 0$
   - $x = x_{\text{in}}$
   - $y = y_{\text{in}}$
   - $z = 0$
   - $i = 0$

   2.3. Processo iterativo:

   if $y_i \geq 0$:

   $$x_{i+1} = x_i + (y_i \gg i)$$
   $$y_{i+1} = y_i - (x_i \gg i)$$
   $$z_{i+1} = z_i + \text{lista\_arctans}(i)$$

   else :

   $$x_{i+1} = x_i - (y_i \gg i)$$
   $$y_{i+1} = y_i + (x_i \gg i)$$
   $$z_{i+1} = z_i - \text{lista\_arctans}(i)$$

   $$i = i + 1$$

   2.4. Parar processo iterativo quando $i = 16$

   2.5. Ajustar x com a constante K:
   - $x = x \cdot 0.6072529351$

   2.6. Saídas:
   - done = 1
   - $x_{\text{out}} = \|\vec{v}\| = \sqrt{x^2 + y^2}$
   - $y_{\text{out}} \approx 0$
   - $z_{\text{out}} = \arctan\left(\frac{y}{x}\right)$

3. Observações:

3.1. Na explicação, é dito para múltiplicar o vetor (x, y) por $K = 0.6072529351$ ao final do processo, porém, no modo Rotação, é mais prático não fazer isso, mas ao invés disso já definir $x_0 = K$, antes do processo iterativo. Porém, no modo Vetorização, é preciso multiplicar tanto o x e y finais por K depois do processo iterativo.

3.1.1. Utilizaremos a representação de número de ponto fixo Qn.m para representar K.

- Qn.m: n bits para a parte inteira, m bits para a parte decimal.
- Consequentemente, teremos que representar todos os nossos valores em Qn.m também, para nossos calculos estarem dentro da mesma representação; explicação de Q2.16 abaixo.

3.2. Na teoria, o processo de iteração do CORDIC para quando z tende a 0 (no modo Rotação) ou quando y tende a 0 (no modo Vetorização), porém, em VHDL, a gente vai parar quando um número definido de iterações acontecer (pois os valores já estarão muito próximos de 0)

- Por exemplo nós definiremos um valor fixo para i (número de iterações), que será o número de bits de precisão que temos.
- Assim, o sinal *done* é emitido quando o número da iteração chega no valor i que definimos.

3.3. Os valores de $\arctan(2^{-i})$ são pré-computados: uma lista de 16 valores, um para cada i-ésima iteração.

- Portanto, faça uma lista dos valores de $\arctan(2^{-i})$: $\{\arctan(2^{-0}), \arctan(2^{-1}), ..., \arctan(2^{-15})\}$,

4. Representação de ponto fixo Q2.14:
   - Nosso projeto atua sobre os seguintes valores possivelmente não-inteiros:

$$K = 0.6072529351,\ \ \cos z : [-1, +1],\ \ \sin z : [-1, +1],\ \ \arctan z : \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right],\ \ 2^i : \left(0, \frac{1}{2}\right],$$
$$z : \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right] , x : [?, ?],\ \ y : [?, ?]$$

   - Tirando x e y:
     - valor mínimo: $-\frac{\pi}{2} = -1.570796327$
     - valor máxixo: $+\frac{\pi}{2} = +1.570796327$
   - Para representar esse intervalo: Número fixo sinalizado Q2.16
     - 2 bits à esquerda: parte inteira
       - intervalo de valores do lado esquerdo: $[-2.0, 1.0]$
     - 16 bits à direita: parte decimal
       - intervalo de valores do lado direito: $[0.0, 1.0)$
       - 2^{16} 1's $= 65536 \approx 1.0$
     - Intervalo de Q2.16: $[-2, 2)$
   - Para simplificar, pode ser relevante representar x_0 e y_0 como Q2.16 também, o que nos permitiria escolher entre x's e y's entre -2 e 2, o que são muitos valores, por serem "números com vírgula".
   - Exemplos:
     - $(01_{\text{C2}}$ na esquerda$) + (65536_{10}$ na direita$) = 1.0 + 1.0 = 2.0$
     - $(00_{\text{C2}}$ na esquerda$) + (65536_{10}$ na direita$) = 0.0 + 1.0 = 1.0$
     - $(00_{\text{C2}}$ na esquerda$) + (32767_{10}$ na direita$) = 0.0 + 0.5 = 0.5$
     - $(10_{\text{C2}}$ na esquerda$) + (32767_{10}$ na direita$) = (-2.0) + 0.5 = -1.5$
   - Conversão de número em VHDL:

```
use ieee.numeric_std.all;

signal x: signed(15 downto 0);

-- Número real -> Q2.16
q1_15 <= to_signed(integer(0.6072529351 * 2.0**16), 16)

-- Q2.16 -> Número real (para debugging)
valor_real := to_integer(q1_15)/2.0**16
```