

MINISTERE DES ETUDES SUPERIEUR ET DE LA RECHERCHE  
SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE D'ALGER

DEPARTEMENT ELECTRONIQUE



---

## Projet N°1 : Processeur CORDIC

---

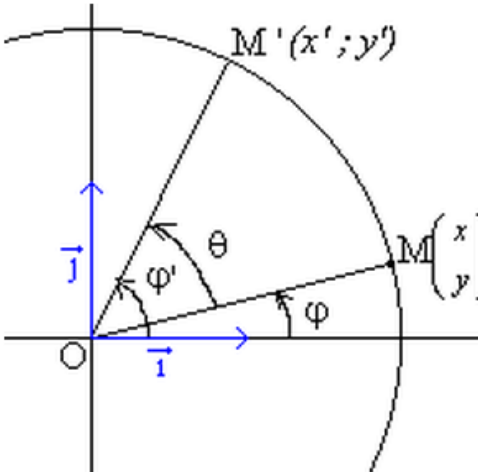
FERHAT Hiba El Batoul

Septembre 2020

CORDIC (COordinate Rotation DIGital Computer) est un algorithme basé sur le concept de la matrice rotation afin d'obtenir des fonctions trigonometriques.

### Volder algorithme :

Le passage entre deux vecteurs dans le meme cercle trigonometrique avec un angle  $\theta$  s'effectue avec une matrice rotation.

$$\begin{aligned}
 v_i &= \begin{bmatrix} x_i \\ y_i \end{bmatrix} = R_{i-1} v_{i-1} \\
 &= v_0 \prod_0^{i-1} R_i \\
 R_i &= \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \\
 &= \cos(\theta_i) \begin{bmatrix} 1 & -\tan(\theta_i) \\ \tan(\theta_i) & 1 \end{bmatrix} \\
 &= \frac{1}{\sqrt{1 + \tan(\theta_i)^2}} \begin{bmatrix} 1 & -\tan(\theta_i) \\ \tan(\theta_i) & 1 \end{bmatrix}
 \end{aligned}$$


Si on fixe  $\theta_i = \arctan(2^{-i})$  afin que la multiplication par la tangente devienne une multiplication par une puissance de 2 qui est une opération aisée à réaliser puisqu'en binaire il s'agit d'un décalage de bits. On obtient :

$$v_i = v_0 \prod_0^{i-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix}$$

La direction de l'angle  $\theta_i$  change de valeur selon l'angle prédéterminé. Ainsi  $\theta_i$  converge vers ce dernier.

Avec cet algorithme on peut déterminer les coordonnées de n'importe quel vecteur disposé entre 0 et  $\frac{\pi}{2}$ . L'application de ces formules mathématiques peut se faire en deux modes :

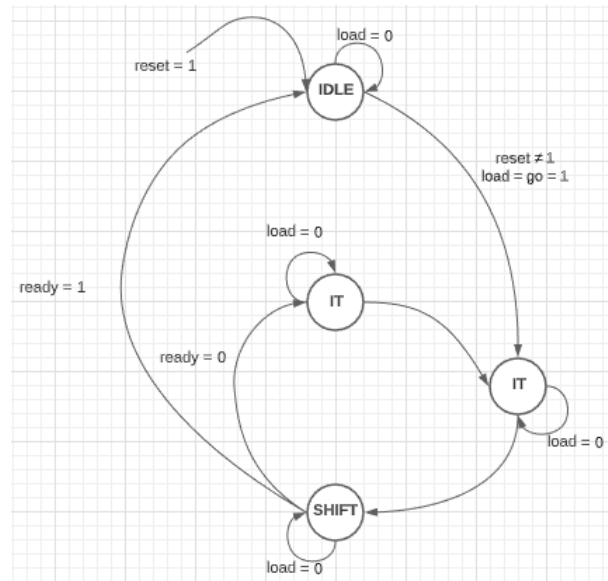
1. Mode rotatif : Il calcule  $v_i$  et  $K_i$  au fur et à mesure jusqu'à obtenir le résultat final. J'ai utilisé ce mode.
2. Mode vectoriel : Il calcule que la partie matricielle de  $v_i$  après dix itérations il multiplie le résultat par la valeur de  $\prod_0^9 K_i = 0.6072$  afin d'obtenir le résultat final.

### Remarques :

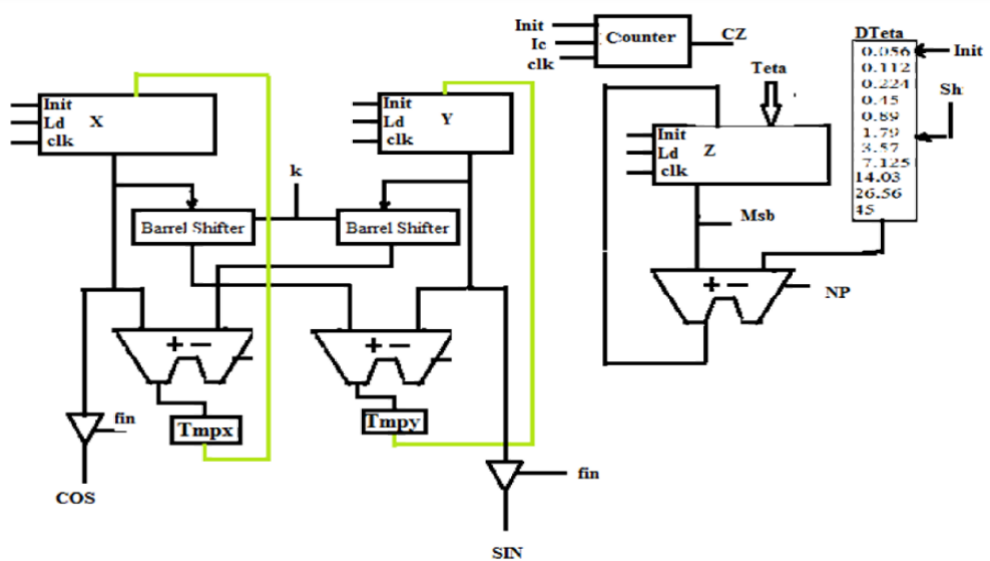
1. Généralement dix itérations suffisent pour atteindre une bonne précision.
2. Cet algorithme ne fonctionne que si  $\theta \leq \sum_0^{n-1} \arctan(2^{-i}) = 1.7$ . Les autres angles seront amenés par les formules trigonométriques vers le premier quart du cercle trigonométrique.
3. La précision du cordic diminue pour les angles très petits car les  $\theta_i$  sont prédéterminées et n'atteindront pas l'angle voulu.

## Logique :

### 1. Machine d'état :



### 2. Chemin de données :



## Conception :

### 1. Unite de controle (finite state machine) :

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.math_real.all;
5  use ieee.fixed_pkg.all;
6
7  entity fsm_cordic is
8  port( clk, reset, load : in std_logic;
9        angle : in sfixed(15 downto -14);
10        cz : in std_logic;
11        init, go : out std_logic;
12        theta : buffer real;
13        cas : out integer);
14  end entity;
15
16  architecture fsm of fsm_cordic is
17  type state is (idle, it, cnt, shift);
18  signal etat : state := idle;
19  begin
20
21  process(clk, load, reset)
22  begin
23  if reset = '1' then
24  etat <= idle;
25  elsif load = '1' then
26  if rising_edge(clk) then
27  if cz = '0' then
28  case etat is
29  when idle => etat <= it;
30  when it => etat <= cnt;
31  when cnt => if cz = '1' then etat <= idle; else etat <= shift; end if;
32  when shift => etat <= it;
33  end case;
34  end if;
35  end if;
36  end if;
37  end process;
38
39  process(clk, angle)
40  variable alpha : real;
41  begin
42
43  if math_2_pi < to_real(angle) then alpha := to_real(angle) mod math_2_pi;
44  else alpha := to_real(angle);
45  end if;
46
47  if (0.0 < alpha) and (alpha < (math_pi_over_2)) then
48  cas <= 1;
49  elsif ((math_pi_over_2) < alpha) and (alpha < math_pi) then
50  alpha := (math_pi) - alpha; cas <= 2;
51  elsif (math_pi < alpha) and (alpha < (math_3_pi_over_2)) then
52  alpha := alpha - math_pi; cas <= 3;
53  elsif ((math_3_pi_over_2) < alpha) and (alpha < (math_2_pi)) then
54  alpha := (math_2_pi) - alpha; cas <= 4;
55  end if;
56  theta <= alpha;
57
58  if (angle'event or reset = '1') then init <= '1'; else init <= '0'; end if;
59  go <= load;
60  end process;
61  end architecture;
```

### 2. Chemin de données (Datapath) :

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.math_real.all;
5  use ieee.fixed_pkg.all;
6
7  entity dp_cordic is
8  port(
9      clk, reset, load : in std_logic;
10     angle : in real;
11     cas : in integer;
12     cos, sin : out sfixed(15 downto -14);
13     ready : buffer std_logic);
14 end entity;
15
16 architecture dp of dp_cordic is
17     signal x : real := 1.0;
18     signal y : real := 0.0;
19     signal cnt : integer := 0;
20     signal theta : real;
21     signal atan : real;
22 begin
23     atan <= 0.7853 when cnt = 0
24         else 0.4636 when cnt = 1
25         else 0.2449 when cnt = 2
26         else 0.1243 when cnt = 3
27         else 0.0624 when cnt = 4
28         else 0.0312 when cnt = 5
29         else 0.0156 when cnt = 6
30         else 0.0078 when cnt = 7
31         else 0.0039 when cnt = 8
32         else 0.0019 when cnt = 9
33         else 0.0009 when cnt = 10;
34
35     ITX : process(clk, load, reset)
36     variable K : real := 1.0;
37     variable tmpy : real := 0.0;
38     variable signe : real;
39     begin
40         if reset = '1' then
41             x <= 1.0;
42         elsif load = '1' then
43             if rising_edge(clk) then
44                 if cnt < 10 then
45                     tmpy := y*2**(-real(cnt));
46                     K := 1.0 / sqrt(1.0 + 2.0**(-real(cnt+1)));
47                     signe := theta/abs(theta);
48                     if signe = 1.0 then x <= K*(x - tmpy); else x <= K*(x + tmpy); end if;
49                 end if;
50             end if;
51         end if;
52     end process;
53

```

```

54 IIV : process(clk, load, reset)
55   variable K : real := 1.0;
56   variable tmpx : real := 0.607;
57   variable signe : real;
58   begin
59   if reset = '1' then
60     y <= 0.0;
61   elsif load = '1' then
62     if rising_edge(clk) then
63       if cnt < 10 then
64         tmpx := x*2*(-real(cnt));
65         K := 1.0 / sqrt(1.0 + 2.0*(-real(cnt+1)));
66         signe := theta/abs(theta);
67         if signe = 1.0 then y <= K*(y + tmpx); else y <= K*(y - tmpx); end if;
68       end if;
69     end if;
70   end if;
71   end process;
72
73 COUNTER : process(clk, load, reset, cnt)
74   begin
75   if reset = '1' then
76     cnt <= 0; ready <= '0';
77   elsif load = '1' then
78     if rising_edge(clk) then
79       if cnt < 10 then cnt <= cnt + 1; end if;
80       if cnt = 10 then ready <= '1'; else ready <= '0'; end if;
81     end if;
82   end if;
83   end process;
84
85 SHIFT : process(clk, load, reset, cnt)
86   variable signe : real;
87   begin
88   if reset = '1' then
89     theta <= angle;
90     signe := theta/abs(theta);
91   elsif load = '1' then
92     if rising_edge(clk) then
93       signe := theta/abs(theta);
94       if cnt < 10 then
95         if signe = 1.0 then theta <= theta - atan;
96         elsif signe = -1.0 then theta <= theta + atan; end if;
97       end if;
98     end if;
99   end if;
100   end process;
101
102 AFFECTATION : process(clk, load, reset)
103   begin
104   if reset = '1' then
105     cos <= to_sfixed(x,15,-14); sin <= to_sfixed(y,15,-14);
106   elsif load = '1' then
107     if rising_edge(clk) then
108       if cas = 1 then

```

3. Processeur :

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library work;
5  use work.all;
6  use ieee.fixed_pkg.all;
7  use ieee.fixed_pkg.sfixed;
8
9  entity processor_cordic is
10 port(  clk, reset, load : in std_logic;
11         angle : in sfixed(15 downto -14);
12         cos, sin : out sfixed(15 downto -14);
13         ready : buffer std_logic);
14 end entity;
15
16 architecture FULL_VERSION of processor_cordic is
17 component fsm_cordic is
18 port(  clk, reset, load : in std_logic;
19         angle : in sfixed(15 downto -14);
20         cz : in std_logic;
21         init, go : out std_logic;
22         theta : buffer real;
23         cas : out integer);
24 end component;
25 component dp_cordic is
26 port(  clk, reset, load : in std_logic;
27         angle : in real;
28         cas : in integer;
29         cos, sin : out sfixed(15 downto -14);
30         ready : buffer std_logic);
31 end component;
32 signal go, init : std_logic;
33 signal theta : real;
34 signal cas : integer;
35 begin
36     FSM : fsm_cordic port map (clk, reset, load, angle, ready, init, go, theta, cas);
37     DP : dp_cordic port map (clk, init, go, theta, cas, cos, sin, ready);
38 end architecture;

```

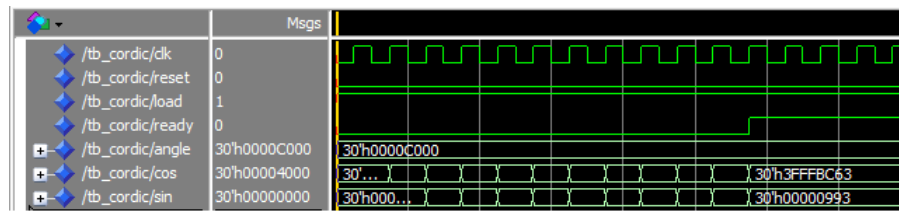
## 4. Testbench :

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library work;
5  use work.all;
6  use ieee.fixed_pkg.all;
7
8  entity tb_cordic is
9  end entity;
10
11 architecture testbench of tb_cordic is
12 component processor_cordic is
13 port(  clk, reset, load : in std_logic;
14         angle : in sfixed(15 downto -14);
15         cos, sin : out sfixed(15 downto -14);
16         ready : buffer std_logic);
17 end component;
18 signal clk, reset, load, ready : std_logic;
19 signal angle, cos, sin : sfixed(15 downto -14);
20 begin
21     UUT : processor_cordic port map (clk, reset, load, angle, cos, sin, ready);
22     reset <= '0';
23     load <= '1';
24     angle <= to_sfixed(3, angle);
25 process
26 begin
27     clk <= '0'; wait for 5 ns;
28     clk <= '1'; wait for 5 ns;
29 end process;
30 end architecture;

```

## 5. Simulation :



Afin de mieux visualiser le resultat on utilise le meme angle avec le datapath directement pour voir les signaux intermediaux réels.

