Synthèse logique de CORDIC

La lecture du cours est nécessaire pour comprendre les principes de l'algorithme CORDIC utilisé pour la rotation.

Dans ce TME, vous devez écrire le modèle VHDL du circuit CORDIC à partir d'une description de l'algorithme en décrit en C.

- Le circuit réalise la rotation d'un vecteur (x,y) par un un angle a.
- Le circuit prend en entrée
  - ♦ les coordonnées x_p et y_p qui sont des nombres entiers signés de -127 à +127.
  - ♦ L'angle a_p est exprimé en radian et il est représenté par un nombre en virgule fixe **non** signé 3-7.
    - ◊ À l'intérieur du circuit, c'est un nombre en virgule fixe 1-8-7.
    - ◊ Mais à l'interface, j'ai choisi une représentation non signée 3-7 (port a_p) pour avoir des angles entre 0 et presque 8 radians. La conversion se fait dans le circuit en recopiant les 10 bits de a_p dans les 10 bits de poids faible d'un registre de 16 bits représentant l'angle, puis en complétant avec des `0`à gauche. C'est un choix pour réduire le nombre de broches, mais vous pouvez faire un choix plus "propre" en codant l'angle en 1-3-7 et faire une conversion avec extension du signe.
  - ♦ le circuit reçoit aussi une horloge.
- Le circuit produit en sortie les coordonnées (nx_p, ny_p) du vecteur après rotation.
- Le protocole de communication en entrée et en sortie est FIFO.

```
//-------------------------------------------------------------------------------
// Calcul des points d'un cercle par l'algorithme Cordic
//-------------------------------------------------------------------------------
// cossin :  utilisation des fonctions cos et sin de la bibliothèque maths
// cordic :  cordic en virgule fixe 8 chiffres après la virgules
//-------------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

void cossin(double a_p, char x_p, char y_p, char *nx_p, char *ny_p)
{
    *nx_p = (char) (x_p * cos(a_p) - y_p * sin(a_p));
    *ny_p = (char) (x_p * sin(a_p) + y_p * cos(a_p));
}

short F_PI = (short)((M_PI) * (1<< 7));
short ATAN[8] = {
    0x65,                // ATAN(2^-0)
    0x3B,                // ATAN(2^-1)
    0x1F,                // ATAN(2^-2)
    0x10,                // ATAN(2^-3)
    0x08,                // ATAN(2^-4)
    0x04,                // ATAN(2^-5)
    0x02,                // ATAN(2^-6)
    0x01,                // ATAN(2^-7)
};

void cordic(short a_p, char x_p, char y_p, char *nx_p, char *ny_p)
{
    unsigned char i, q;
    short a, x, y, dx, dy;

    // conversion en virgule fixe : 7 chiffres après la virgule
    a = a_p;
```

```
    x = x_p << 7;
    y = y_p << 7;

    // normalisalion de l'angle pour être dans le premier quadrant
    q = 0;
    while (a >= F_PI/2) {
        a = a - F_PI/2;
        q = (q + 1) & 3;
    }

    // rotation
    for (i = 0; i <= 7; i++) {
        dx = x >> i;
        dy = y >> i;
        if (a >= 0) {
            x -= dy;
            y += dx;
            a -= ATAN[i];
        } else {
            x += dy;
            y -= dx;
            a += ATAN[i];
        }
    }

    // produit du résultat par les cosinus des angles : K=0x4E=1001110
    x = ((x>>6) + (x>>5) + (x>>4) + (x>>1))>>7;
    y = ((y>>6) + (y>>5) + (y>>4) + (y>>1))>>7;

    // placement du points dans le quadrant d'origine
    switch (q) {
    case 0:
        dx = x;
        dy = y;
        break;
    case 1:
        dx = -y;
        dy = x;
        break;
    case 2:
        dx = -x;
        dy = -y;
        break;
    case 3:
        dx = y;
        dy = -x;
        break;
    }
    *nx_p = dx;
    *ny_p = dy;
}

int main()
{
    FILE *f;
    double t,K;
    int i;
    printf("PI/2 = %x %x\n", F_PI/2i, 128<<7);
    for (i=0, t=1, K=1; i < 8; i++, t=t/2) {
        printf("B_%d = 0x%x\n", i, (short)(round(atan(t)*(1<< 7))));
        K = K * cos (atan(t));
    }
    printf("K = %g, 0x%x\n", K, (short)(round(K*(1<< 7))));

    f = fopen("cossin.dat", "w");
    for (double a = 0; a <= M_PI * 2; a += 1. / 256) {
```

```
        char nx_p;
        char ny_p;
        cossin(a, 127, 0, &nx_p, &ny_p);
        fprintf(f, "%4d %4d\n", nx_p, ny_p);
    }
    fclose(f);

    f = fopen("cordic.dat", "w");
    for (short a = 0; a <= 2 * F_PI ; a += 1) {
        char nx_p;
        char ny_p;
        cordic(a, 127, 0, &nx_p, &ny_p);
        fprintf(f, "%4d %4d\n", nx_p, ny_p);
    }
    fclose(f);

    return 0;
}
```