MOCCA

Conception d'un circuit
sous alliance

MOCCA — 2023 — Cordic

Automate d'états finis synchrones

MOCCA — 2023 — Cordic

Automate d'états finis synchrones

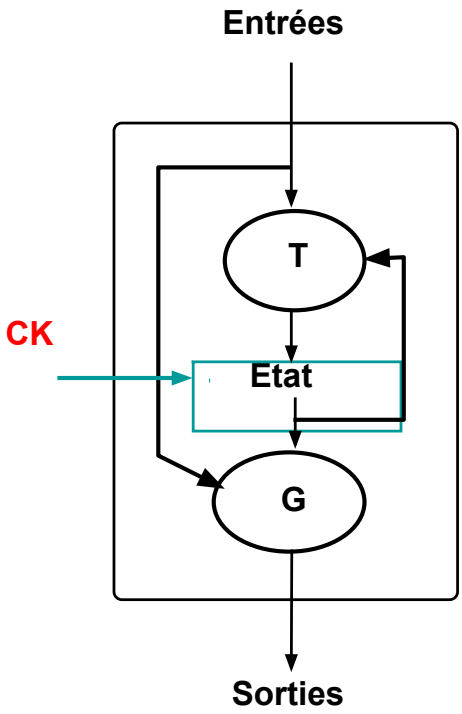
La théorie des automates est une méthode de représentation extrêmement générale du comportement d'un système matériel numérique synchrone.

Le comportement de n'importe quel système synchrone, (simple compteur 4 bits, ou microprocesseur 32 bits complet) peut - en principe - être représenté par ce modèle.

Le comportement d'un système complexe est souvent décrit en interconnectant des automates plus simples.

Il existe une méthode systématique permettant de construire le schéma en portes logiques réalisant le comportement défini par un automate abstrait.

Principe Général



Tout système numérique synchrone peut être modélisé par un automate :

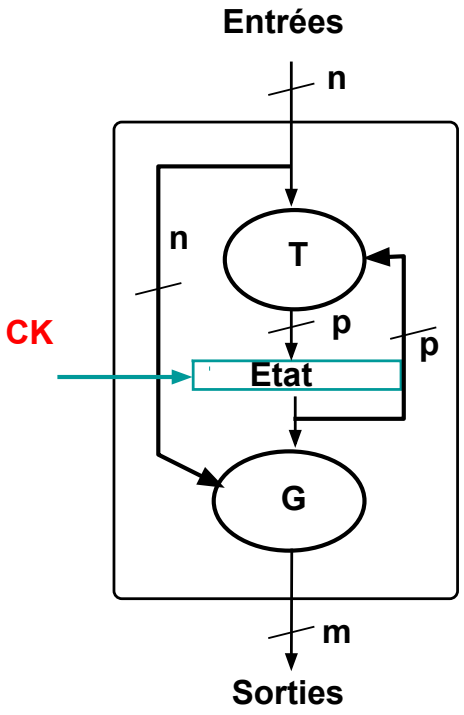
Fonction de transition :

NextEtat <= T(Etat, Entrées)

Fonction de génération :

Sorties <= G(Etat, Entrées)

Fonctions de génération et de transition



Si on a :
- n bits d'entrée Ei
- m bits de sortie Sj
- p bits mémorisés Rk

Il faut définir :

- p fonctions booléennes dépendant de (n+p) variables pour la transition

$$NR_k = T_k(E_i, R_k)$$

- m fonctions booléennes dépendant de (n+p) variables ou la génération

$$S_j = \Gamma j(E_i, R_k)$$

Représentation abstraite

- Dans la définition générale d'un automate, les fonctions de transition et de génération sont définies pour :
  - un ensemble de valeurs symboliques pour les entrées
  - un ensemble de valeurs symboliques pour les sorties
  - un ensemble de valeurs symboliques pour les états

  ... mais le code binaire associée à chacune de ces valeurs n'est pas défini.

- En pratique, les signaux d'entrée et de sortie sont très souvent définis au niveau Booléen : n bits d'entrée correspondent à 2^n valeurs possibles pour les entrées. Idem pour les sorties.

Seules les valeurs stockées dans les registres ne sont pas définies au niveau Booléen : le codage des états n'est pas explicite.

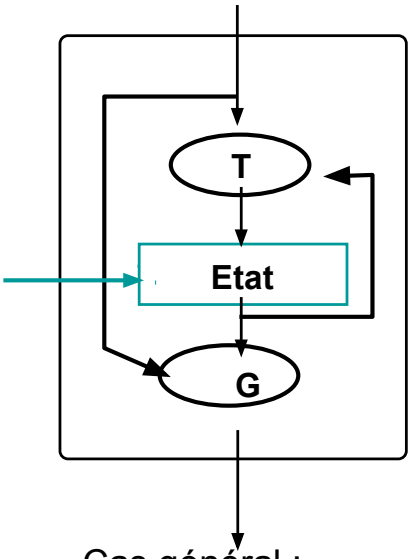
Initialisation

- Puisqu'un automate contient un état interne, il faut définir un état initial pour obtenir un comportement totalement déterministe.

- Les automates synchrones possèdent très souvent un signal d'entrée particulier (signal NRESET), qui n'agit que sur la fonction de transition : lorsque le signal NRESET est actif (état bas), on force la valeur de l'état initial dans le registre d'état lors du prochain front du signal d'horloge...

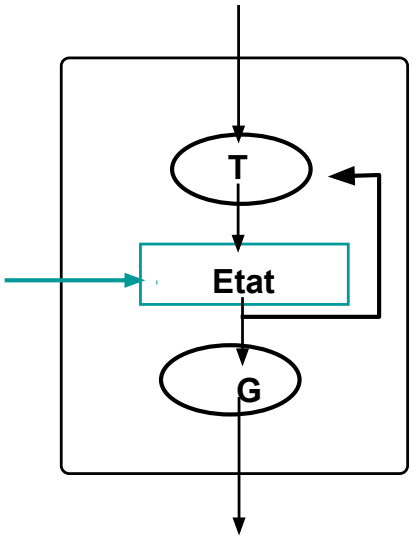
  - quel que soit l'état actuel de l'automate
  - quelle que soit la valeur des autres entrées.

Automates de Moore et de Mealy



Cas général :

Automate de Mealy



Cas particulier : La fonction de génération ne dépend que de l'état !

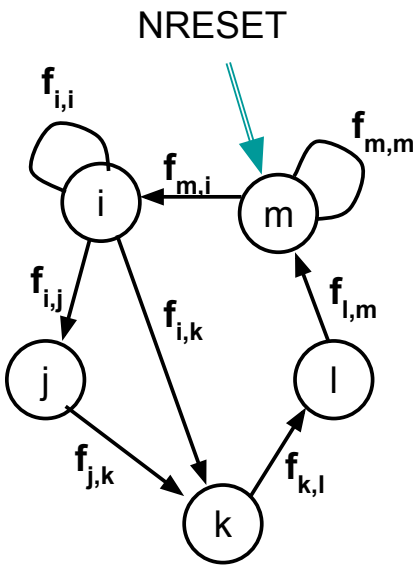
Automate de Moore

Représentation graphique d'un automate

- Les nœuds {i} représentent les états
- Les arcs (i,j) représentent les transitions

Chaque arc (i,j) est étiqueté par une expression Booléenne $\mathbf{f_{i,j}}$ ne dépendant que des signaux d'entrée, et définissant la condition de transition.

Dans le cas d'un automate de Moore, les signaux de sortie ne dépendent que de l'état : chaque nœud est donc étiqueté par la valeur des signaux de sortie.



Automate déterministe

Pour qu'un automate possède un comportement déterministe, il faut respecter les conditions suivantes :

- Existence d'un mécanisme matériel d'initialisation permettant de forcer l'automate dans un état initial connu.
- Condition d'orthogonalité : pour tout état i, et pour toute configuration des entrées, il y a un seul état successeur de l'état i.

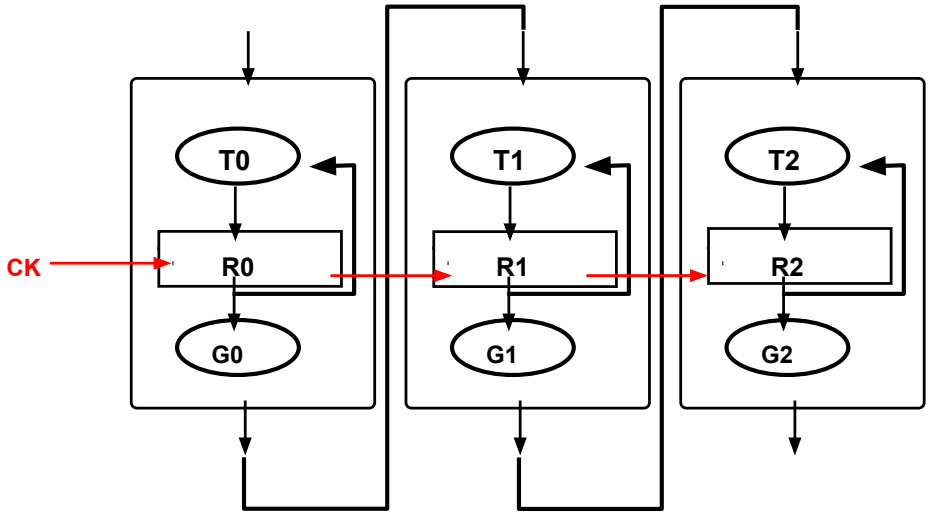
Pour tout état i, $\mathbf{f_{i,j}} \cdot \mathbf{f_{i,k}} = 0$ si j différent de k

- Condition de complétude : pour toute configuration des entrées, il y a toujours un état successeur de l'état i.

Pour tout état i, $\Sigma_j \ \mathbf{f_{i,j}} = 1$

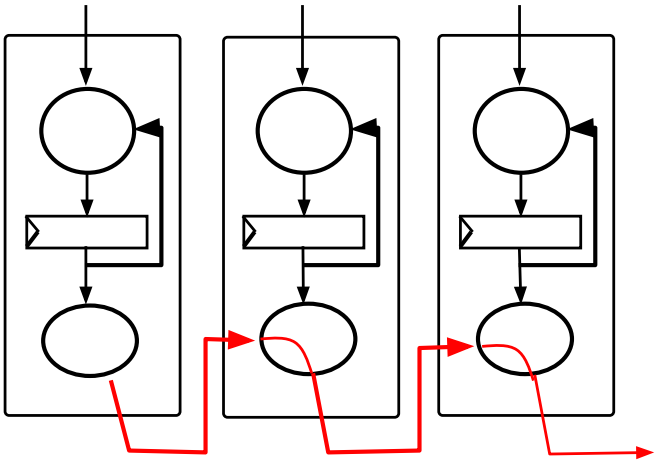
Automates communicants

Un système numérique synchrone est souvent conçu comme
un ensemble d'automates « simples » fonctionnant en parallèle,
et communiquant entre eux :
les entrées d'un automate sont les sorties d'un autre automate.



Inconvénient des Automates de mealy

Dans un automate de mealy, Il existe une dépendance combinatoire entre les entrées et les sorties !



ceci introduit des chaînes longues traversant plusieurs composants.

Il devient impossible de caractériser le comportement temporel de chaque automate indépendamment des autres.

Exemple : Allocateur de bus

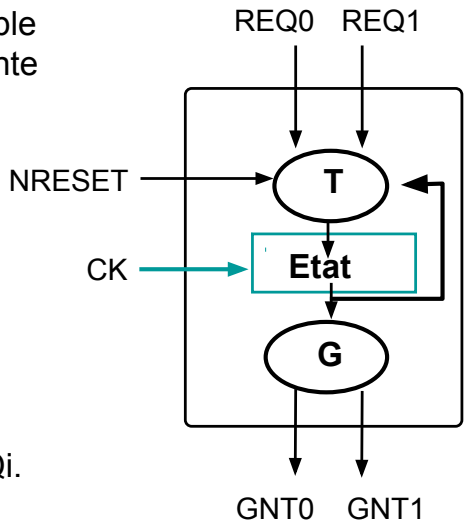
On cherche à réaliser un allocateur de bus équitable entre 2 utilisateurs (respectant une priorité tournante ou « round robin »).

Le signal NRESET initialise l'automate dans un état où le bus n'est pas alloué.

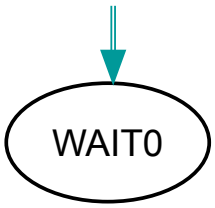
Les deux requêtes REQ0 et REQ1 sont actives à l'état haut, et indépendantes.

L'utilisateur possédant le bus signale la fin de l'utilisation en forçant la valeur 0 sur le signal REQi.

Les deux signaux GNT0 et GNT1 ne peuvent être actifs en même temps, et il y a toujours un cycle non alloué (GNT0 = GNT1 = 0) entre deux allocations



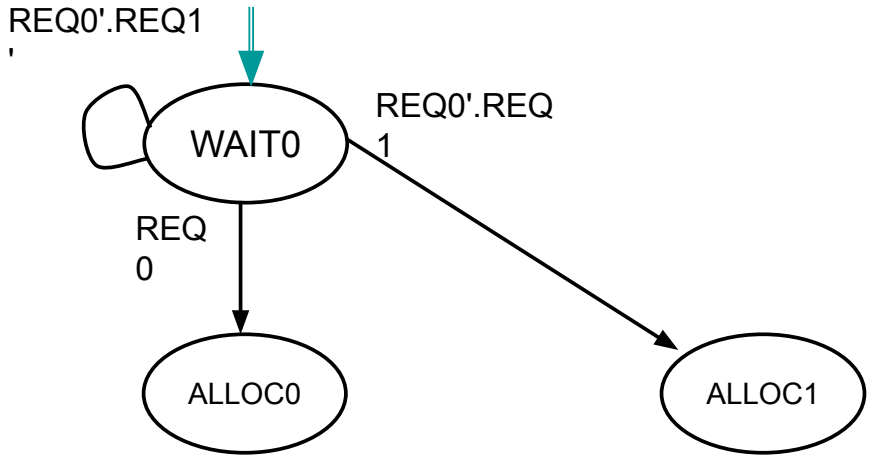
représentation graphique



On devine un état initial :

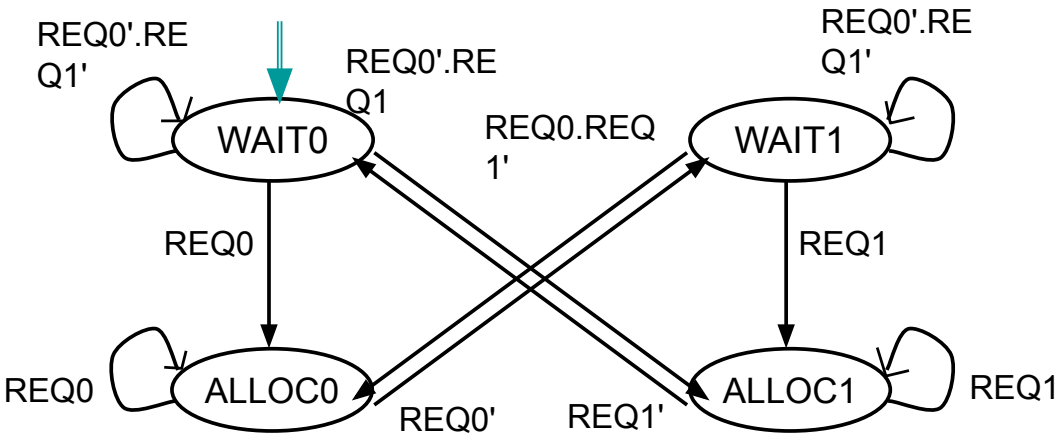
Ici l'état WAIT0 dans lequel l'automate attend une requête alors que c'est l'utilisateur 0 qui a eu le bus en dernier

représentation graphique



Pour l'état de départ,
On étudie toutes les combinaisons de sortie
et on créé de nouveaux états.
Puis on recommence pour chaque nouvel état.

représentation graphique

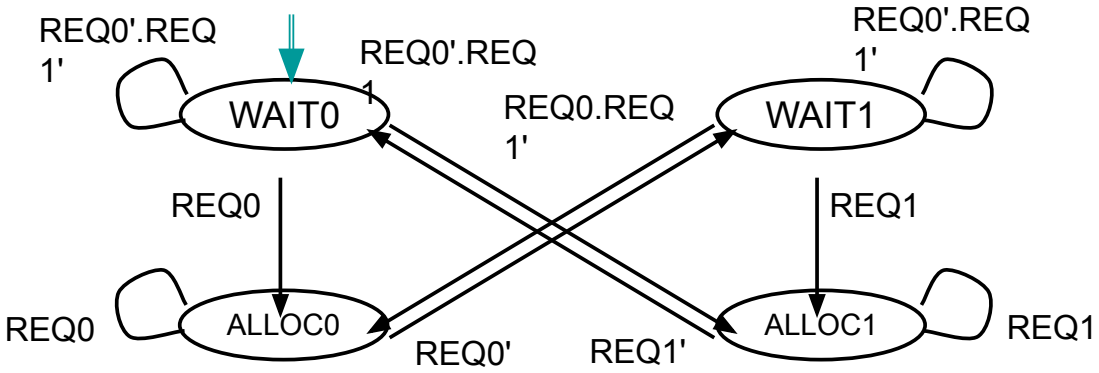


Signification des états :

-WAIT0 : bus non alloué / utilisateur 0 prioritaire
-WAIT1 : bus non alloué / utilisateur 1 prioritaire
-ALLOC0 : bus alloué à l'utilisateur 0
-ALLOC1 : bus alloué à l'utilisateur 1

A : codage one-hot /1

- En codage one-hot le registre d'état a autant de bits qu'il y a d'états, un bit par état.
- Définir la fonction de transition consiste à définir l'expression de chaque état.
- Le codage étant connu,
  il est possible de décrire l'automate directement en vbe



A : codage one-hot /2

```
entity allocateur is
-- liste des ports d'entrée-sortie
port (ck          : in std_logic ;
        req0        : in std_logic ;
        req1        : in std_logic ;
        nreset      : in std_logic ;
        gnt0        : out std_logic ;
        gnt1        : out std_logic ) ;
end allocateur ;

architecture fsm of allocateur is
-- déclaration des états
signal
  s_wait0,  r_wait0, -- attente req0 prioritaire
  s_wait1,  r_wait_0,-- attente req1 prioritaire
  s_alloc0, r_alloc0,-- bus alloué par req0
  s_alloc1, r_alloc1 -- bus alloué par req1
  : std_logic;

begin

 -- fonction de génération
   gnt0 <= r_alloc0;
   gnt1 <= r_alloc1;
```

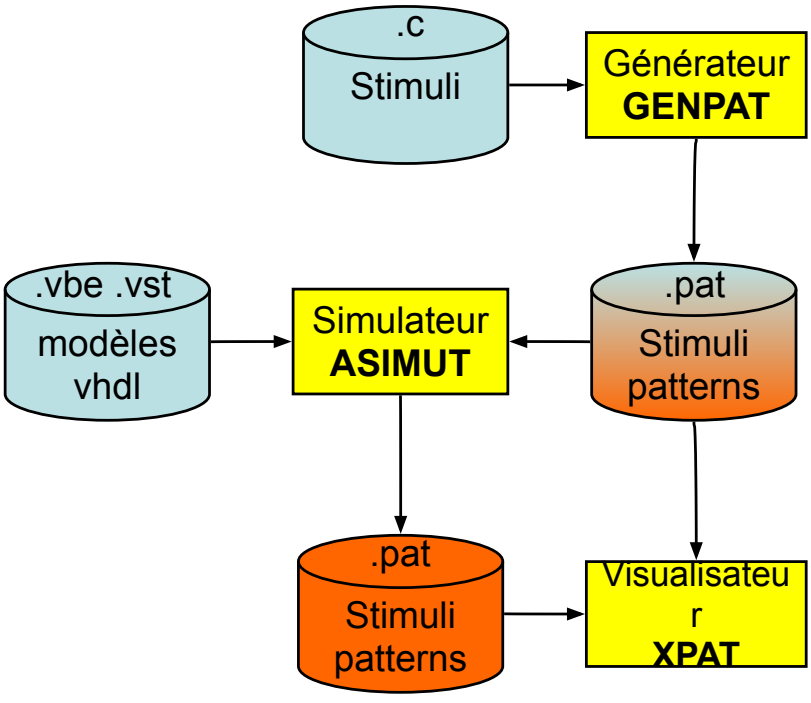
```
-- fonction de transition
    s_wait0  <= (r_wait0 and not req0 and not req1)
             or (r_alloc1 and not req1);
    s_wait1  <= (r_wait1 and not req0 and not req1)
             or (r_alloc1 and not req0);
    s_alloc0 <= (r_wait0 and req0)
             or (r_alloc0 and req0);
    s_alloc1 <= (r_wait1 and req1)
             or (r_alloc1 and req1);

-- mise à jour du registre d'état
   reg : process (ck) begin
     if (ck and ck'event) then
       if (nreset = '0') then
          r_wait0  <= '1';
          r_wait1  <= '0';
          r_alloc0 <= '0';
          r_alloc1 <= '0';
       else
          r_wait0  <= s_wait0;
          r_wait1  <= s_wait1;
          r_alloc0 <= s_alloc0;
          r_alloc1 <= s_alloc1;
       end if;
     end if;
  end process reg;
end fsm;
```

B : Modèle VHDL allocateur : architecture fsm

```
entity allocateur is
-- liste des ports d'entrée-sortie
port (ck          : in std_logic ;
      req0        : in std_logic ;
      req1        : in std_logic ;
      nreset      : in std_logic ;
      gnt0        : out std_logic ;
      gnt1        : out std_logic ) ;
end allocateur ;
architecture fsm of allocateur is

-- définition du type énuméré
type etat_type is (wait0, wait1,alloc0 ,alloc1);

-- déclaration des signaux
signal present, futur : etat_type ;

-- directives pour la synthèse :
-- pragma current_state present
-- pragma next_state futur
-- pragma clock ck
Begin
   -- processus « update »
   process ( ck ) begin
        if (ck and ck' event) then
       present <= futur;
        end if ;
   end process;
```

```
-- processus « combinatoire »
process ( present, req0, req1, nreset ) begin
    -- fonction de transition
    if (nreset = '0') then   futur <= wait0 ;
    else
      case present is
      when wait0 => if (req0) then futur <= alloc0 ;
                    elsif (req1) then futur <= alloc1 ;
                    end if ;
      when wait1 => if (req1) then futur <= alloc1 ;
                    elsif (req0) then futur <= alloc0 ;
                    end if ;
      when alloc0 => if ((req0)='0') then futur <= wait1;
                     end if ;
      when alloc1 => if ((req1)='0') then futur <= wait0;
                     end if ;
      end case ;
    end if ;

-- fonction de génération
    if (present = alloc0)
      then gnt0 <= '1' ;
      else gnt0 <= '0' ;
    end if ;
    if (present = alloc1)
      then gnt1 <= '1' ;
      else gnt1 <= '0' ;
     end if ;
   end process ;
end fsm;
```


Chaine de simulation



format PAT

- Le format de fichier .pat utilisé par Asimut permet de décrire les patterns appliqués et les résultats attendus.

- Le format .pat contient:
  - L'interface du circuit
  - La séquence des patterns
  - Les actions sur le simulateur

- Documentation : **`man 5 pat`**

Exemple de .pat

```
in       A (0 to 15) X;
in       B (0 to 15) X;
in       Cin;
out      Cout;
signal   S (0 to 15) X;
register Accu.A (0 to 15) X;

begin

<   0 ns > pattern_0 : F0F0 0A0A 1 ?0 ?FAFA ?6DE7;
< +10 ns > pattern_1 : 0F0F F6F0 0  +  **** ?54FC;

end;
```

Interface .pat

mode ***nom de l'entrée-sortie*** ***format*** ***[spy] ; [;]***

- in
- out
- inout
- signal
- register

- nom
- group (nom1, nom2, …)
- chemi-nom pour les signaux ou registres internes
  - .vbe : root.nom
  - .vst : nom
    instance.nom

- B
- X
- O
- *rien*

Attention

Les noms de vecteurs de signaux ont le format VHDL

Exemple

```
in        A (0 to 15) X;
in        B (0 to 15) X;
in        Cin ;;
out       Cout;
signal    S (0 to 15) X;
register  Accu.A (0 to 15) X;
```


Séquence de patterns

```
begin
    [< date >]  [étiquette] : valeurs_des_signaux ; [;]
    …
end ;
```

- **date** :: [+] *nombre_entier unité*
  - unité :: ps | ns | us | ms
  - [+] :: délai depuis le précédent pattern
- **étiquette** :: sert à s'y retrouver dans la séquence de patterns
- **valeurs_des_signaux** ::
  - entrées = 0 | 1 | + | - | 00110 | 0256 | DEAD
  - sorties prédites = ?0 | ?1 | ?+ | ?- | ?00110 | ?0256 | ?DEAD
  - sorties non prédites = * | ****
- **[;]** :: permet d'ajouter une ligne blanche dans le fichier produit

Le simulateur ASIMUT

Documentation : **`man asimut`**
asimut [options] [root_file] [pattern_file] [result_file]

options essentielles

-b	si le fichier à simuler est uniquement comportemental (.vbe)
-c	asimut fait seulement une lecture du modèle
-i val	initialise tous les signaux à val (0 ou 1)
-zd	force la simulation sans délai

autres options

-i file initialise tous les signaux à partir du fichier **file** (sauvé par la save)

-inspect inst_name
produit un fichier de pattern avec les signaux à l'interface de inst_name.

-core file
produit un fichier .cor avec l'état de tous les signaux dès la première erreur.

variables d'environnement

- MBK_CATA_LIB répertoires contenant les descriptions et les patterns
- MBK_WORK_LIB répertoire de travail avec les descriptions et les patterns et où sont écrits les fichiers produits
- MBK_CATAL_NAME nom du fichier catalogue (placé dans MBK_WORK_LIB)
- MBK_IN_LO extension (type) des fichiers netlist (al ou vst)
- VH_MAXERR nombre maximum d'erreurs autorisées avant l'arrêt de la simulation

fichier CATAL

- Le fichier CATAL permet d'indiquer quelles sont les feuilles de l'arborescence dans le cas de la simulation d'une netlist.
- Les noms présents dans le fichier ont un modèle comportemental.
- format

  bloc1 C

  bloc2 C

  bloc3 C

Langage GENPAT

- Documentation : `man genpat`
- Le but est d'exprimer dans un langage procédural, les transactions sur les signaux.
- C'est une bibliothèque de fonctions C:
  - pour définir l'interface
  - et les transactions
  - pour générer un fichier de patterns au format .pat
- Le langage C permet l'écriture de boucles et de fonctions
- Un script permet de lancer le compilateur C puis l'exécution du programme :

```
> genpat [-v] [-k] file
```

API Genpat (essentiel)

- DEF_GENPAT("nom")
  définit le nom de fichier dans lequel les patterns seront placés.
- SAV_GENPAT()
  sauve le fichier sur disque.
- DECLAR("ident",":nb_space","format",mode,"size","option")
  déclare le nom du signal
  - ident : nom du signal
  - nb_space : nombre d'espaces entre les colonnes,
  - format : format d'affichage (B,O,X),
  - type : IN, OUT, INOUT, SIGNAL, REGISTER
  - size : rien, X to Y, Y downto X
  - option : rien, S
- AFFECT("pattern_date", "ident", "value")
  définit une transaction sur le signal
  - pattern_date : "nombre" date absolue de la transaction, ou "+nombre" date relative au dernier AFFECT() ou INIT()
  - ident : nom du signal
  - value : nombre dans la base définie par format

API Genpat : Exemple

fonction transformant un entier en chaine de caractères

fichier vecteurs.pat

déclaration de l'interface et des signaux internes

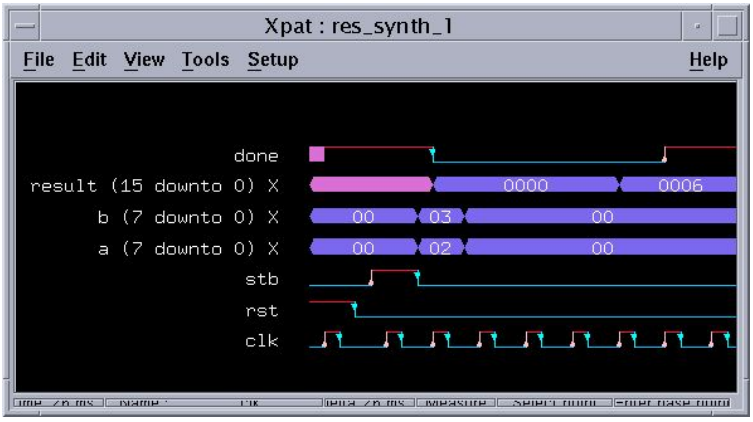
boucles imbriquées produisant 256 patterns

```
#include <stdio.h>
#include "genpat.h"

char *itoa(int entier)   {
    char * str = malloc (32);
    sprintf (str, "%d",entier);
    return(str);
}

main ()  {
    int i, j, cur_vect = 0;

    DEF_GENPAT("vecteurs");

    DECLAR ("a", ":2", "X", IN, "0  to 3");
    DECLAR ("b", ":2", "X", IN, "0  to 3");
    DECLAR ("s", ":2", "X", OUT, "0 to 3");

    for (i=0; i<16; i++) {
        for (j=0; j<16; j++) {
            AFFECT (itoa(cur_vect), "a", itoa(i));
            AFFECT (itoa(cur_vect), "b", itoa(j));
            cur_vect++;
        }
    }
    SAV_GENPAT ();
}
```

Le visualisateur de patterns XPAT

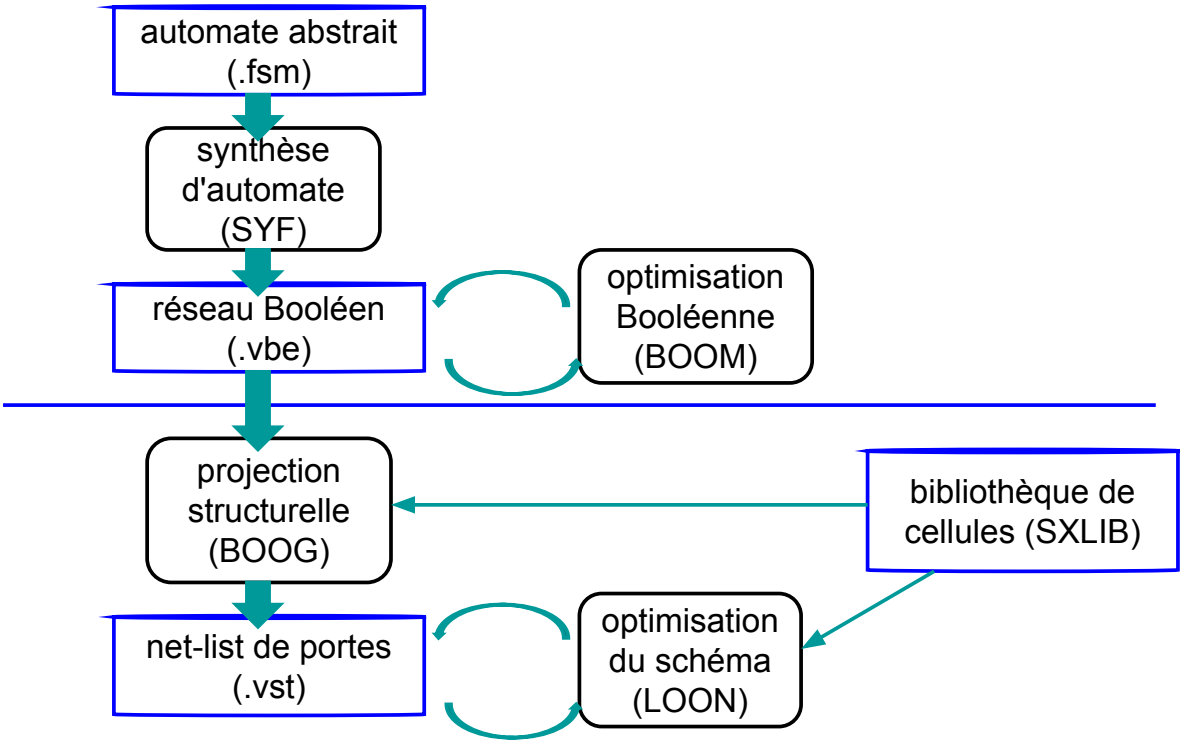
XPAT est l'un des nombreux outils de visualisation
- avantage : fonctionnement simple et intuitif.
- inconvenient : peu de fonctions

`xpat [-l file]`



Chaîne de synthèse

Les étapes de la synthèse Alliance



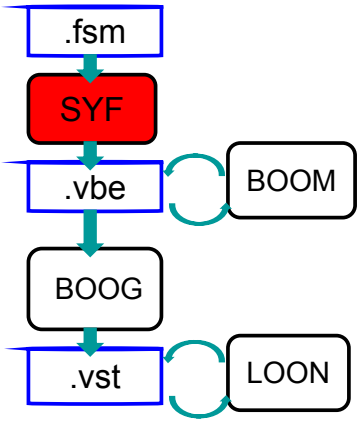
La synthèse d'automate

Les outils de synthèse d'automate (exemple SYF)
appliquent la méthode générale vue précédemment :

- Construction du graphe représentant l'automate abstrait
- Choix d'un codage pour les états
- Construction des fonctions de transition et de génération
- Simplification des expressions Booléennes
- Génération du réseau Booléen

La principale intervention du concepteur
porte sur le choix d'un type de codage.

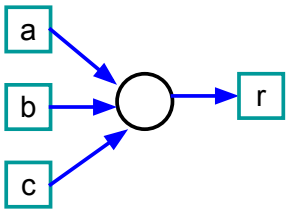
Contrairement à l'intuition, le codage
« one-hot » donne très souvent
de bons résultats !

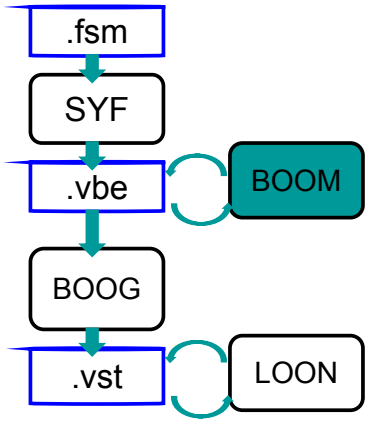


Optimisation Booléenne / a

Les outils d'optimisation Booléenne (exemple : BOOM) cherchent à « simplifier » le réseau Booléen. Cette optimisation étant indépendante du procédé de fabrication choisi, la fonction de coût est le « nombre de littéraux ».

L' optimisation locale vise la simplification de l'expression Booléenne associée à un noeud particulier du réseau Booléen.





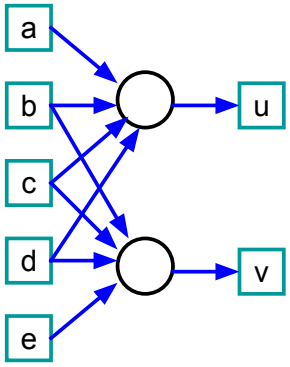
forme canonique : 12 littéraux
r <= a.b.c' + a.b'.c + a'.b.c + a.b.c

forme optimisée : 6 littéraux
r <= a.b + a.c + b.c

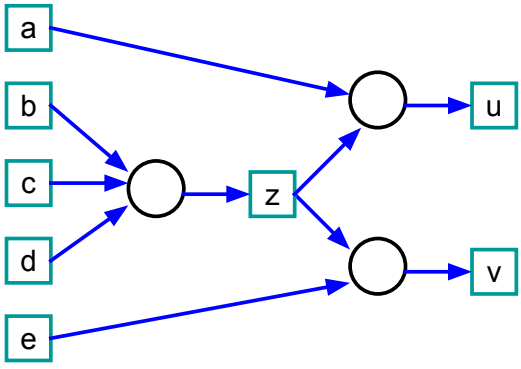
Optimisation Booléenne / b

L'optimisation globale utilise des techniques de factorisation, qui peuvent modifier la structure du réseau Booléen :

Forme initiale :
u <= a.(b.c + b'.d)
v <= (b.c + b'.d) + e



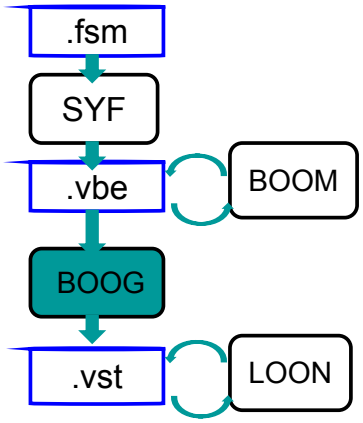
Forme factorisée :
z <= b.c + b'.d
u <= a.z
v <= z + e



Projection structurelle

Les outils de projection structurelle (exemple : BOOG) transforment une expression Booléenne associée à un noeud du réseau Booléen en un schéma en portes logiques.

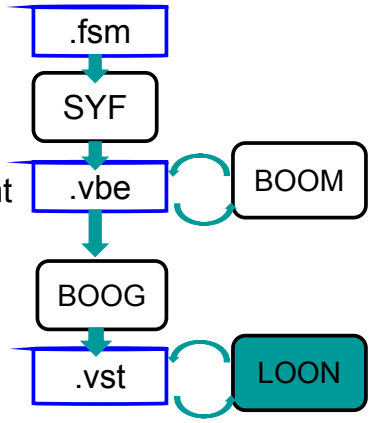
- Ces outils s'appuient sur une bibliothèque de cellules précaractérisées.
- Le traitement est local : chaque expression Booléenne est traitée indépendamment.
- Ils utilisent des techniques de reconnaissance de forme pour reconnaître des sous-expressions.
- Ils exploitent les informations de caractérisation associées aux cellules pour optimiser
  - la surface totale du bloc synthétisé
  - les performances temporelles



Optimisation du schéma / a

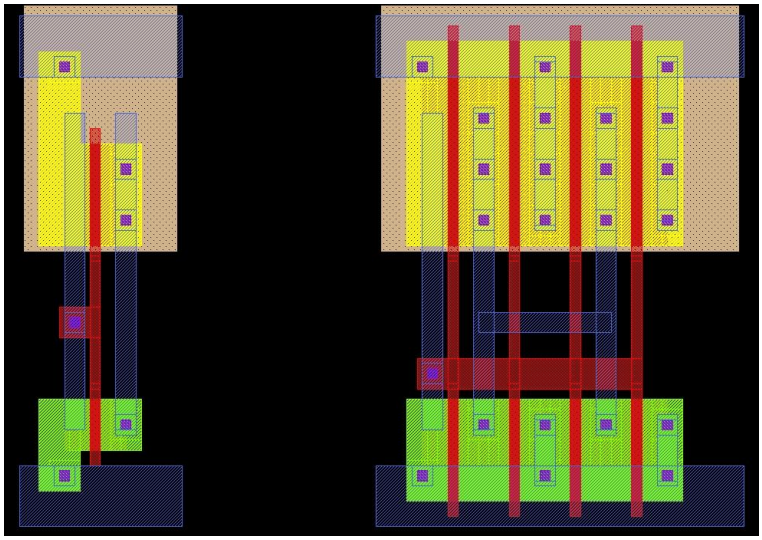
Les outils d'optimisation de schéma (exemple: LOON) visent principalement l'optimisation des performances temporelle. Ils cherchent à réduire les temps de propagation sur les chemins critiques du bloc synthétisé.

- Les deux principales techniques sont
  - l'utilisation de portes de puissance
  - l'insertion de buffers
- Pour optimiser les performances temporelles sans trop augmenter la surface totale du bloc, seules les portes logiques et les signaux se trouvant sur un chemin critique doivent être modifiés.
- L'analyse des chemin critique dépend des temps d'arrivées des signaux sur les ports d'entrée, et des temps requis sur les ports de sortie



Optimisation du schéma / b

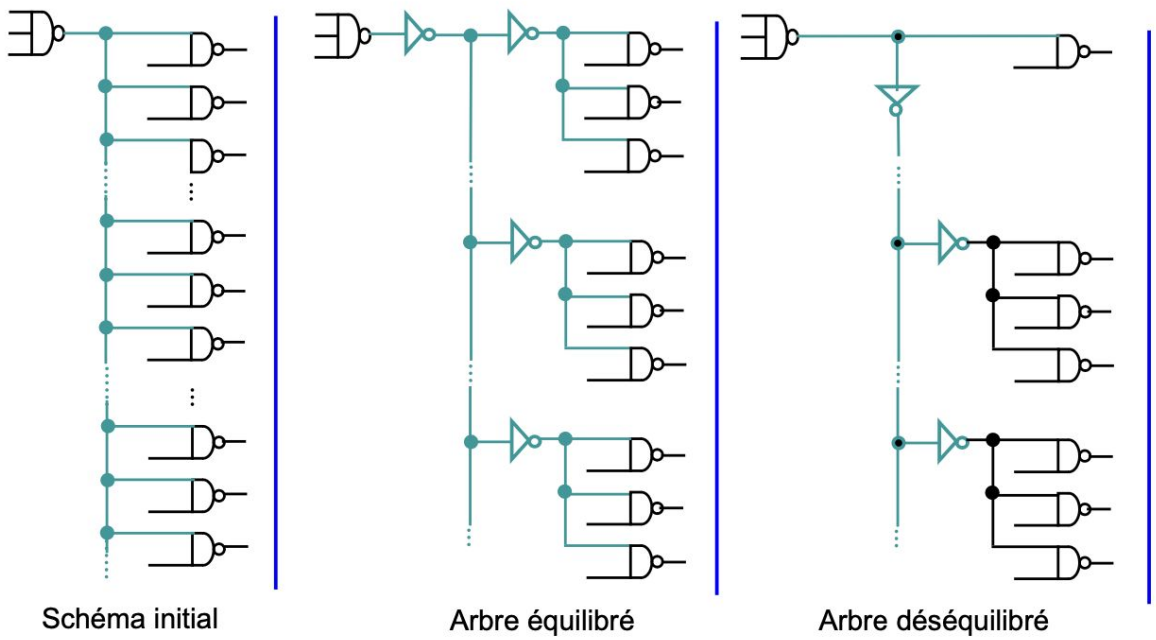
Ajustement de la puissance des portes :



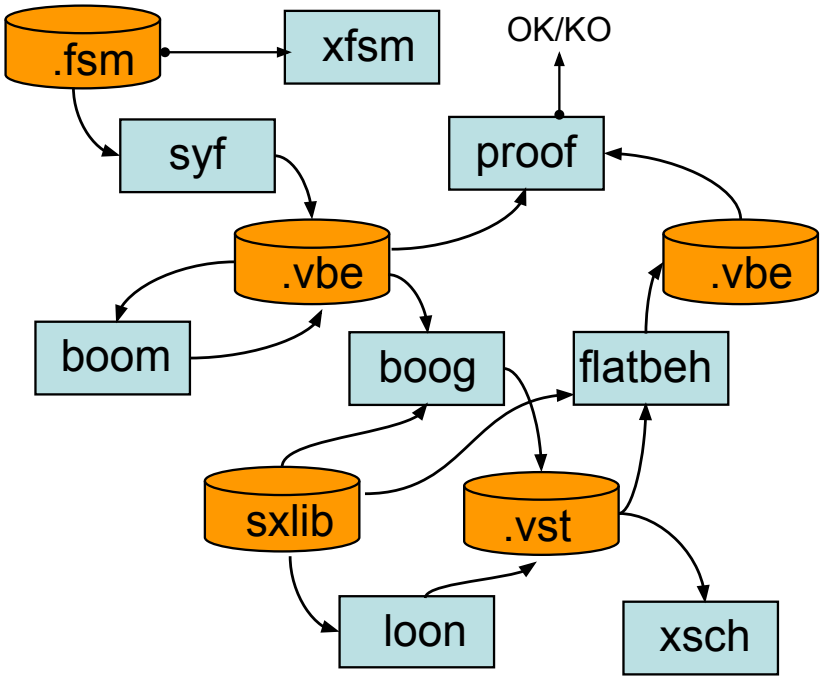
inv_x1 : WN = 5 / WP = 10 inv_x8 : WN = 40 / WP = 80

Optimisation du schéma / c

Insertion d'arbres de buffers (en cas de fanout très grand)



Flot des outils de synthèse Alliance



syf synthèse fsm

boom optimisation comportementale

boog projection structurelle

loon optimisation électrique

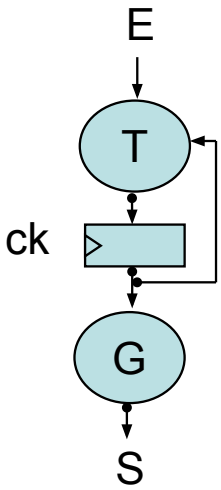
flatbeh mise à plat

xsch visualisation du schéma

xfsm visualisation du graphe fsm

syf

syf prend une machine d'états finis décrite en vhdl, choisit un codage et produit un modèle au format vbe.

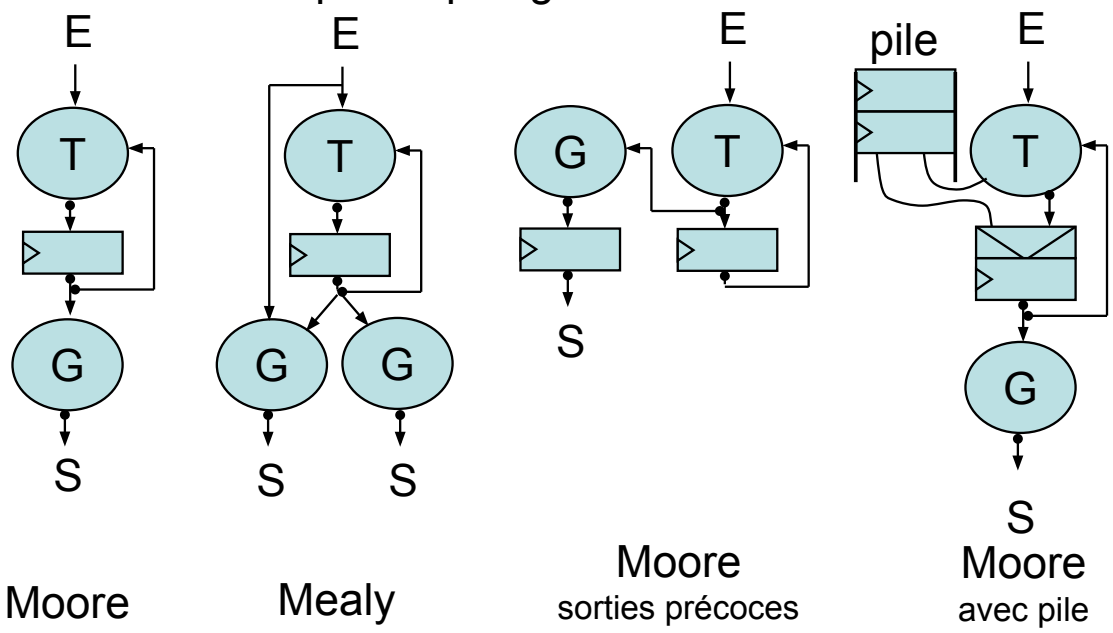


principe de fonctionnement

- le comportement est décrit en vhdl
  - un process ***Transition*** définissant définissant l'état futur en fonction de l'état courant et des entrées
  - un process ***Génération*** définissant les sorties en fonction de l'état courant
- **syf** choisit un codage (plusieurs algos)
- **syf** détermine les expressions de tous les bits du registre d'état et tous les bits de sorties

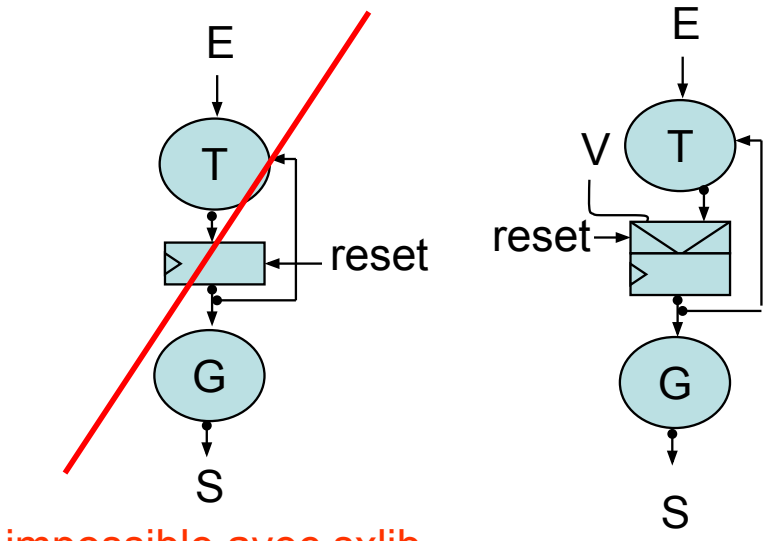
syf

Quelques topologies d'automates.



syf

L'initialisation peut être ou ne pas être synchronisée



syf

Codage d'un automate d'un FSM

syf -j|a|m|o|u|r [-CDEOPRSTV] input [output]

paramètres:

-j -a -m : trois algos de codage (asp, jedi, mustang)
-o : codage one-hot
-u : codage utilisateur dans le fichier input.enc
liste des couples: noms_d'état code_hexa
-r : codage random

-C : vérifie la complétude et l'orthogonalité
-E : sauve l'encodage (syntaxe de –u)
-P : ajoute un scanpath
-R : utilise une ROM et un micro séquenceur
-V : verbose mode

environnement

MBK_WORK_LIB : répertoire de sortie

boom

Optimisation Booléenne

boom [-VTOAP] [-l num] [-d num] [-i num] [-a num]
[-sjbgpwtmorn] input [output]

paramètres

-V : verbose
-A : procède à des optimisations locales en conservant la majorité des signaux internes.
-P : lit le fichier input.boom contenant
- les signaux à conserver dans le fichier produit
- les expressions à conserver
-l val : effort de 0 (faible) à 3 (fort)
-d val : type d'optimisation de 0 (délai) à 100 (surface)
-i val : nb d'itérations pour l'algorithme
-a val : amplitude pendant le réordonnancement des bdd
-sjbgpwtmorn : algorithme choisi

environnement

MBK_WORK_LIB répertoire de sortie

boog

boog prend une description comportementale
et produit une netlist de cellules précaractérisées.

- **boog** ne sait traiter que des expressions produites par **boom.**
- **boog** n'utilise que des portes à une sortie et de faible sortance.
- **boog** connaît les caractéristiques des portes
  grâce à des génériques dans les vbe des portes
- La projection d'un ET à 8 entrées
  s <= a and b and c and d and e and f and g and h
  8 and à 2 entrées
  s <= a and (b and (c and (d and (e and (f and (g and h))))))
  2 nand à 4 entrées + 1 nor à 2 entrées
  s <= not(not (a and b and c and d) or not (e and f and g and h))
- **boog** pourrait se contenter de :
  - nand 2, nor 2, basculeD, inverseur, xor
- **boog** n'est pas déterministe !

boog

Projection structurelle
boog [-hmxold] input output [lax_file]

paramètres

-h	: help
-m val	: optimisation de 0 (surface) à 4 (delai)
-x val	: génération d'un fichier de coloration des signaux 0 (chemin critique), 1 (dégradé en fonction du délai)

environnement

MBK_CATA_LIB	liste des répertoires contenant les fichiers sources
MBK_TARGET_LIB	répertoire de la bibliothèque cible
MBK_OUT_LO	format de la netlist de sortie
MBK_WORK_LIB	répertoire de sortie

loon

Optimisation électrique locale

loon [-hmxlo] input_file output_file [lax_file]

paramètres

-h	: help
-m val	: optimisation de 0 (surface) à 4 (délai)
-x val	: génération d'un fichier de coloration des signaux 0 (chemin critique), 1 (dégradé en fonction du délai)

environnement

MBK_CATA_LIB	liste des répertoires contenant les fichiers sources
MBK_TARGET_LIB	répertoire de la bibliothèque cible
MBK_IN_LO	format de la netlist d'entrée
MBK_OUT_LO	format de la netlist de sortie
MBK_WORK_LIB	répertoire de sortie

flatbeh

flatbeh réalise la mise à plat d'une netlist et produit le modèle comportemental.

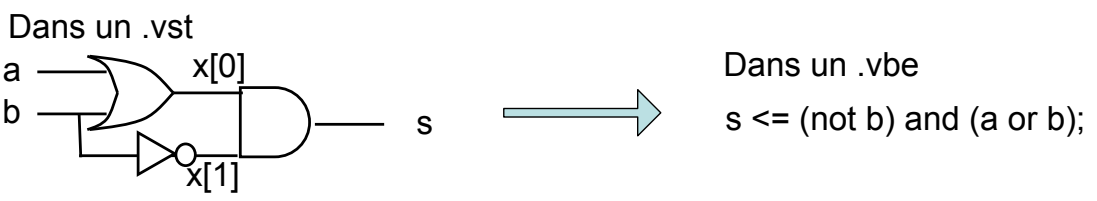
flatbeh root_structural_file [output_file]

paramètre

CATAL	fichiers contenant les cellules feuilles

environnement

MBK_CATA_LIB	liste des répertoires contenant les fichiers sources
MBK_IN_LO	format de la netlist d'entrée
MBK_OUT_LO	format de la netlist de sortie
MBK_WORK_LIB	répertoire de sortie



*(on aura intérêt à utiliser **boom** pour simplifier le résultat)*

proof

proof compare formellement deux descriptions comportementales et affiche les différences.

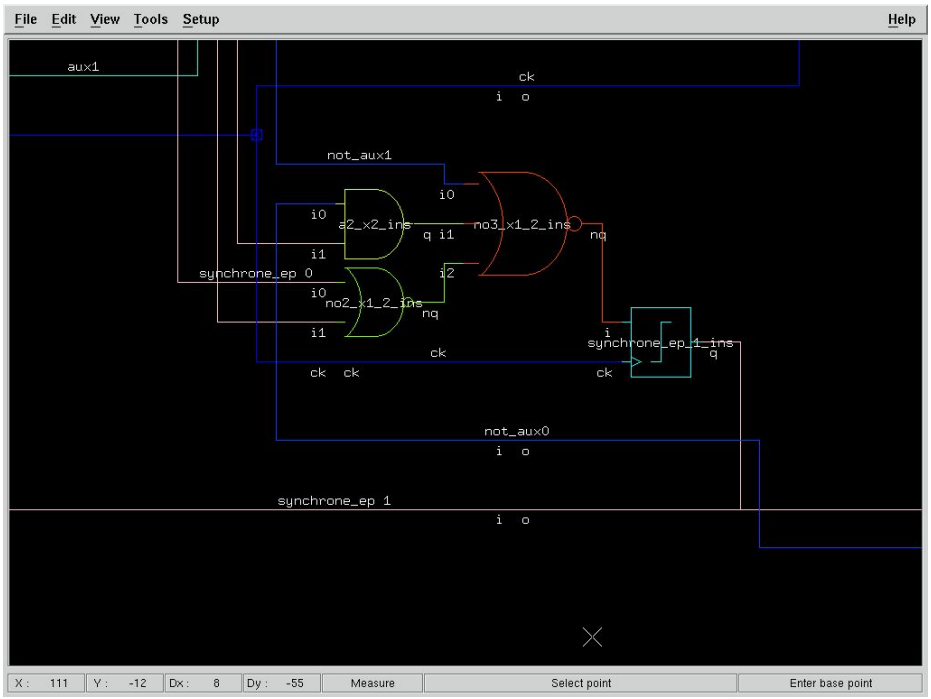
- Les deux descriptions doivent avoir
  - les mêmes entrées/sorties
  - les mêmes registres

- Chaque description est représentée par une structure ROBDD (Reduced Oriented Binary Decision Diagram)

- La représentation d'une expression Booléenne par un ROBDD est canonique
  ⇒ si les représentations sont superposables, c'est qu'elles sont égales

xsch

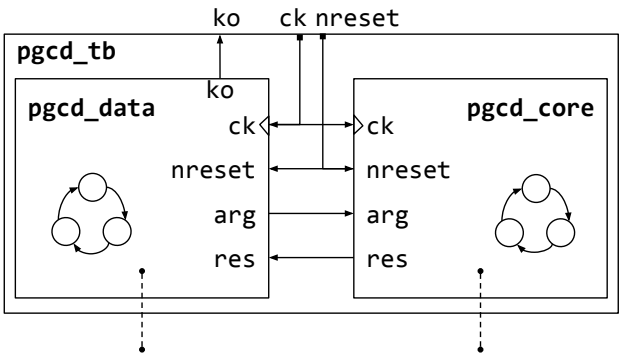
xsch représente une netlist par un schéma



PGCD

un exemple de ce qu'on peut faire
pour valider un circuit simple

Premier exemple : un PGCD



envoie des nombres
en argument et attend
le résultat qu'il vérifie
et signale les erreurs

reçoit les nombres
en argument, calcule
le résultat et l'envoie

Le PGCD est un modèle plus simple que CORDIC, mais la méthode de validation est semblable.
C'est un échauffement :-)

Objectif

- Modéliser en vhd les modèles tb, data et core
- Valider les modèles

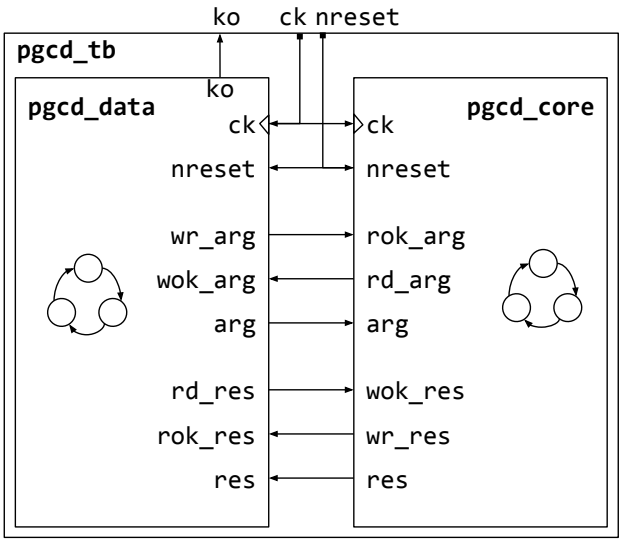
Usage d'Alliance

- `vasy` vhd → vst / vbe
- `genpat` générateur de patterns
- `asimut` simulateur

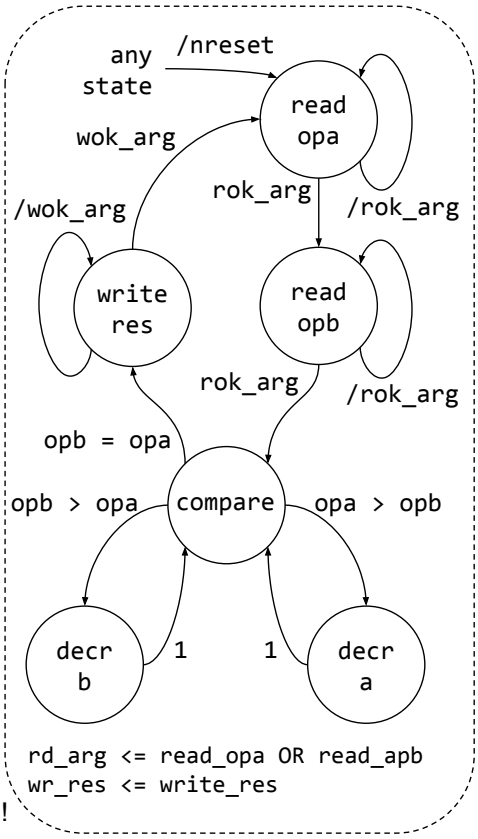
Mais aussi de

- `Makefile`
- `gcc`
- `awk (pour le fun)`

PGCD

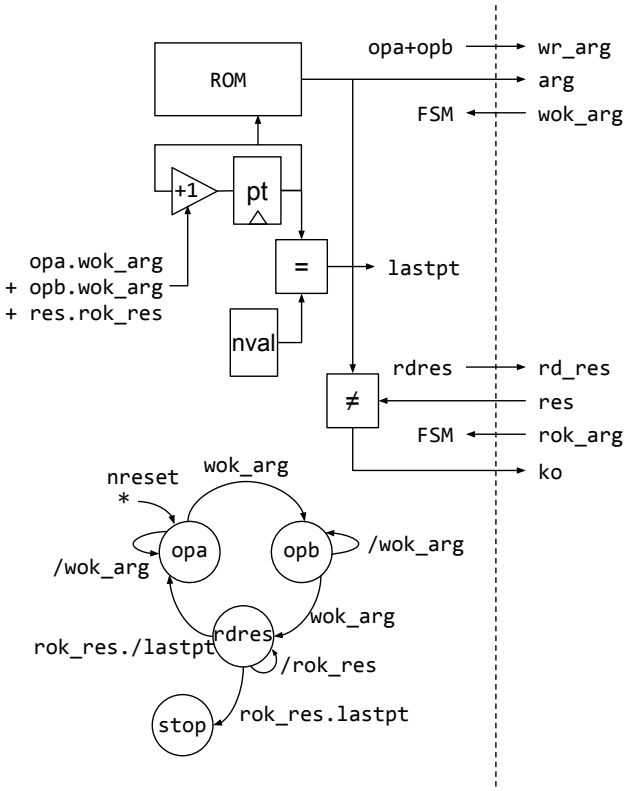



```
unsigned pgcd( unsigned opa,  unsigned opb) {
    while (opa != opb) {
            if (opa > opb) opa -= opb;
        else if (opa < opb) opb -= opa;
    }
    return opa;
}
```



Remarque: write et read sont des ordres, donc toujours des sorties !

pgcd_data en vhdl



- Ce modèle envoie des nombres pris dans une ROM en utilisant le protocole FIFO, puis attend le résultat qu'il compare à ce qu'il a dans sa ROM.
- L'intérêt de cette technique, c'est qu'il n'est pas nécessaire de connaître la durée de calcul, il faut juste connaître le résultat attendu.
- Ici la rom va être produite par un programme en C, puis insérée dans le modèle vhdl de pgcd_data

pgcd_data en vhdl ... *avec du C pour la rom*

interface et signaux internes

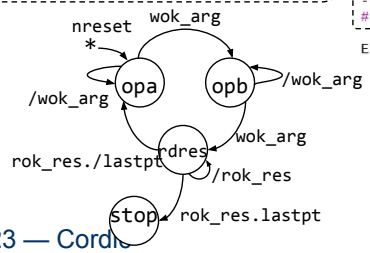
```
ENTITY pgcd_data IS
PORT(
    ck          : IN  std_logic;
    nreset      : IN  std_logic;

    wr_arg_p    : OUT std_logic;
    arg_p       : OUT std_logic_vector(VALWD-1 DOWNTO 0);
    wok_arg_p   : IN  std_logic;

    rd_res_p    : OUT std_logic;
    res_p       : IN  std_logic_vector(VALWD-1 DOWNTO 0);
    rok_res_p   : IN  std_logic;

    ko_p        : OUT std_logic
);
END pgcd_data;

ARCHITECTURE vhd OF pgcd_data IS

    SIGNAL          -- FSM states
        opa,        -- set first operande
        opb,        -- set second operande
        res,        -- get result
        stop,       -- it's over
        lastpt      -- 1 when pt = address of the last filled
    : std_logic;

    SIGNAL
        pt          -- rom_pointer
    : std_logic_vector(ADDRWD-1 downto 0);

    SIGNAL
        value       -- rom_value
    : std_logic_vector(VALWD-1 downto 0);
```

transition one-hot et pointeur de rom

```
BEGIN
    REG : PROCESS (ck) begin
    if ((ck = '1') AND NOT(ck'STABLE)) then
        if (nreset = '0') then
            opa  <= '1';
            opb  <= '0';
            res  <= '0';
            stop <= '0';
            pt   <= (others=>'0');
        else
            opa  <= (res AND rok_res_p AND not lastpt)
                 OR (opa AND not wok_arg_p);
            opb  <= (opa AND wok_arg_p)
                 OR (opb AND not wok_arg_p);
            res  <= (opb AND wok_arg_p)
                 OR (res AND not rok_res_p);
            stop <= (res AND rok_res_p AND lastpt)
                 OR stop;
            if ((opa AND wok_arg_p) OR (opb AND wok_arg_p) OR (res AND rok_res_p)) then
                pt   <= pt + 1;
            end if;
        end if;
    end if;
    end process REG;

    lastpt   <= (pt = LASTPT);
    wr_arg_p <= opa OR opb;
    rd_res_p <= res;
    arg_p    <= value;
    ko_p     <= res AND rok_res_p AND (value /= res_p);

--  #include <rom.txt> incudes a file with a generated ROM, defined as below
--  value       <= x"12"    when pt = 0
--             else x"60"    when pt = 1
--             else x"06"    when pt = 2
--             else x"00";
#   include "rom.txt"

END vhd;
```

ROM

génération Moore & Mealy



il faut utiliser le préprocesseur du C
pour insérer le contenu de la ROM

rom.c

On veut produire ça

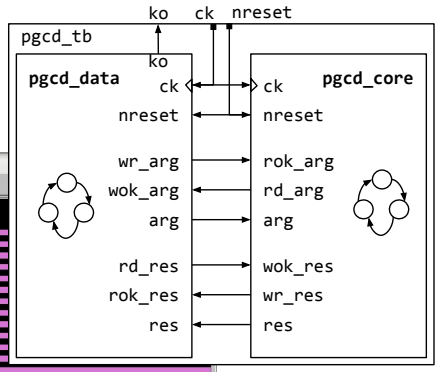
```
value <= x"82" when pt = 0
    else x"91" when pt = 1
    else x"05" when pt = 2
    else x"b4" when pt = 3
    else x"0a" when pt = 4
    else x"0a" when pt = 5
    else x"7a" when pt = 6
    else x"aa" when pt = 7
    else x"02" when pt = 8
    else x"6f" when pt = 9
    else x"0d" when pt = 10
    else x"01" when pt = 11
    else x"be" when pt = 12
    else x"a4" when pt = 13
    else x"02" when pt = 14
    else x"c5" when pt = 15
    else x"de" when pt = 16
    else x"01" when pt = 17
    else x"00";
```

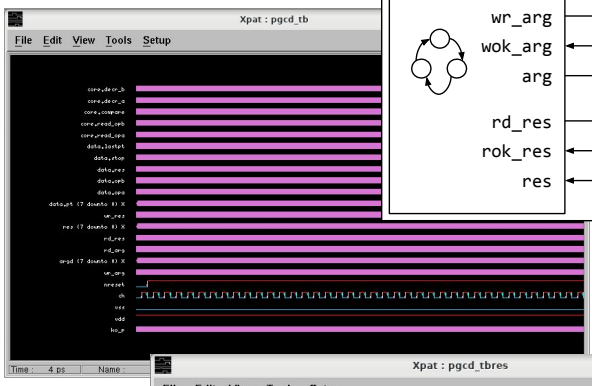
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef ADDRWDMAX   /* number of bits of rom address*/
#define ADDRWDMAX   16
#endif

void usage(char *message) {
    fprintf( stderr, "\nERROR : %s\n\n", message);
    fprintf( stderr, "USAGE rom <addrwd> <valwd>\n");
    fprintf( stderr, "  <addrwd>: number of bits of the rom address (max is %d)\n", ADDRWDMAX);
    fprintf( stderr, "            the number of triplets [opa,opb,pgcd(opa,opb)]\n");
    fprintf( stderr, "            is then the maximum number possible in 2**addrwd\n");
    fprintf( stderr, "  <valwd> : range of aperandes are between 1 to 2**valwd\n\n");
    fprintf( stderr, "Ex: rom 5 8 -> gives 10 triplets with operands from 1 to 2**8-1 (255)\n\n");
    exit (1);
}

// value returns a number from 1 to range
unsigned value( unsigned valrange) {
    return 1 + (rand()%(valrange-1));
}

// return the pgcd of opa and opb
unsigned pgcd( unsigned opa,  unsigned opb) {
    while (opa != opb) {
            if (opa > opb) opa -= opb;
        else if (opa < opb) opb -= opa;
    }
    return opa;
}

unsigned twopow(unsigned n) {
    unsigned res = 1;
    while (n--) res *= 2;
    return res;
}
```

```c
int main( int argc, char * argv[]) {

    if (argc < 3) usage("Too few arguments");
    if (argc > 3) usage("Too much arguments");
    unsigned addrwd = atoi(argv[1]);
    unsigned valwd  = atoi(argv[2]);
    if (addrwd > ADDRWDMAX) usage("<addrwd> too big (change ADDRWDMAX in source code)");

    unsigned valrange = twopow(valwd)-1;
    unsigned valuenb = twopow(addrwd)/3;

    char *name = "value";
    unsigned namelen = strlen(name);
    unsigned rangelen = 1+(valwd-1)/4;

    for(int i=0; i < valuenb; i++) {
        unsigned opa = value(valrange);
        unsigned opb = value(valrange);
        unsigned res = pgcd(opa, opb);
        if (i==0)
            printf ("%*s <= x\"%0*x\" when pt = %d\n", namelen, name, rangelen, opa, i);
        else
            printf ("%*s x\"%0*x\" when pt = %d\n", namelen+3, "else", rangelen, opa, 3*i);
        printf ("%*s x\"%0*x\" when pt = %d\n", namelen+3, "else", rangelen, opb, 3*i+1);
        printf ("%*s x\"%0*x\" when pt = %d\n", namelen+3, "else", rangelen, res, 3*i+2);
    }
    printf ("%*s x\"%0*x\";\n", namelen+3, "else", rangelen, 0);

    fprintf (stderr, "rom generated with %d triplets (opa, opb, res)\n", valuenb);
    return 0;
}
```

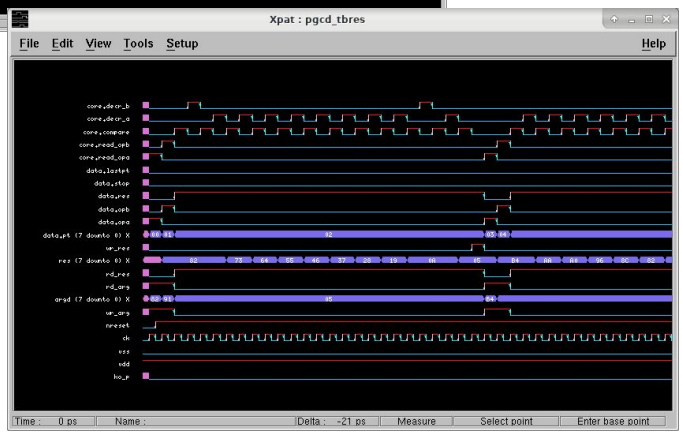
pgcd_pat.c

```
int main ()
{

// since it is not possible to get arguments with genpat,
// we use environment variables. The GETENV() macro try to get
// the variable env value and we can choose a default value for each
#   define GETENV(var,def) getenv(var)?getenv(var):def

    char*    PATNAME= GETENV("PATNAME","default");
    unsigned VALWD  = atoi(GETENV("VALWD" ,"1"));
    unsigned ADDRWD = atoi(GETENV("ADDRWD","1"));
    unsigned CYCLES = atoi(GETENV("CYCLES","100"));

    DEF_GENPAT (PATNAME);

    // interface
    // the 1st port must be "ko" port which is true in case of error
    DECLAR ("ko_p",      ":2", "B", OUT, "", "");

    DECLAR ("vdd",       ":2", "B", IN,  "", "");
    DECLAR ("vss",       ":2", "B", IN,  "", "");
    DECLAR ("ck",        ":2", "B", IN,  "", "");
    DECLAR ("nreset",    ":2", "B", IN,  "", "");

    // It is possible to see internal signals
    DECLAR ("wr_arg",    ":2", "B", SIGNAL, "", "");
    DECLAR ("argd",      ":2", "X", SIGNAL, vector(VALWD-1,0), "");
    DECLAR ("rd_arg",    ":2", "B", SIGNAL, "", "");
    DECLAR ("rd_res",    ":2", "B", SIGNAL, "", "");
    DECLAR ("res",       ":2", "X", SIGNAL, vector(VALWD-1,0), "");
    DECLAR ("wr_res",    ":2", "B", SIGNAL, "", "");

    DECLAR ("data.pt", ":1", "x", REGISTER, vector(ADDRWD-1,0), "");
    DECLAR ("data.opa", ":1", "B", REGISTER, "", "");
    DECLAR ("data.opb", ":1", "B", REGISTER, "", "");
    DECLAR ("data.res", ":1", "B", REGISTER, "", "");
    DECLAR ("data.stop", ":1", "B", REGISTER, "", "");
    DECLAR ("data.lastpt", ":1", "B", SIGNAL, "", "");

    DECLAR ("core.read_opa", ":1", "B", REGISTER, "", "");
    DECLAR ("core.read_opb", ":1", "B", REGISTER, "", "");
    DECLAR ("core.compare",  ":1", "B", REGISTER, "", "");
    DECLAR ("core.decr_a",   ":1", "B", REGISTER, "", "");
    DECLAR ("core.decr_b",   ":1", "B", REGISTER, "", "");

    AFFECT (cycle (0), "vdd", "1");
    AFFECT (cycle (0), "vss", "0");
    AFFECT (cycle (0), "nreset", "0");
    AFFECT (cycle (1), "nreset", "1");

    // la génération du signal d'horloge
    int c;
    for (c = 0; c <= CYCLES; c++) {
        AFFECT (cycle (c), "ck", inttostr (0));
        AFFECT (next_cycle (c), "ck", inttostr (1));
    }
    AFFECT (cycle (0), "nreset", "0");

    SAV_GENPAT ();
    return 0;
}
```





simulation asimut



Makefile

Makefile contenant le processus de validation du PGCD (ici ensemble de "script shell")

```
CFLAGS  = -Wall -O3 -std=c99 #-DDEBUG
LDFLAGS = -lm

MODEL   = pgcd#     model name to validate
ADDRWD  = 8#        bit width of addresses for the rom
VALWD   = 8#        bit width of operands for the rom
CYCLES  = 4000#     number of cycles to simulate (should be suffisant for all triplets)

# VALNB must contains the number of triplets [opa,opb,res]
# VALNB depends on the address width ADDRWD, since VALNB = (2**ADDRWD)/3
# LASTPT is the pointer (pgcd_data.pt) for the last res in rom (pgcd_data.pt is from 0 to 3*VALNB-1)
# the lines below show how to compute an expresssion in the makefile with awk command
VALNB   = $(shell awk -v ADDRWD=$(ADDRWD) 'BEGIN{print int((2**ADDRWD)/3) }')
LASTPT  = $(shell awk -v VALNB=$(VALNB) 'BEGIN{print 3*VALNB-1}')

valid_pgcd:
    $(CC) $(CFLAGS) rom.c -o rom
    ./rom   $(ADDRWD) $(VALWD) > rom.txt
    export PATNAME=$(MODEL)_tb ADDRWD=$(ADDRWD) VALWD=$(VALWD) CYCLES=$(CYCLES);\
    genpat $(MODEL)_pat
    gcc -w -E -DADDRWD=$(ADDRWD) -DVALWD=$(VALWD) -DLASTPT=$(LASTPT) $(MODEL)_data.vhd.c\
    | grep -v "^#" > $(MODEL)_data.vhd
    vasy -a -I vhd -p -o $(MODEL)_data $(MODEL)_data
    vasy -a -I vhd -p -o $(MODEL)_core $(MODEL)_core
    vasy -a -I vhd -p -o $(MODEL)_tb   $(MODEL)_tb
    asimut  $(MODEL)_tb $(MODEL)_tb $(MODEL)_tbres |\
    awk '/pattern/{printf("->"$$3" "$$4"\r")}END{print}'
    @grep ": ?1" $(MODEL)_tbres.pat || echo "Lucky no error"

clean:
    rm  Makefile.*\
        $(MODEL)_core.vbe\
        $(MODEL)_data.vbe\
        $(MODEL)_data.vhd\
        $(MODEL)_tb.vst\
        $(MODEL)_tb.pat\
        $(MODEL)_tbres.pat\
        default.pat\
        rom rom.txt\
        2> /dev/null || true
```

awk, n'est pas indispensable….

Si la séquence des opérations à réaliser est courte alors la description détaillée des dépendances de fichiers est inutile, cela alourdi la description du processus de construction et c'est une source d'erreurs

Cordic

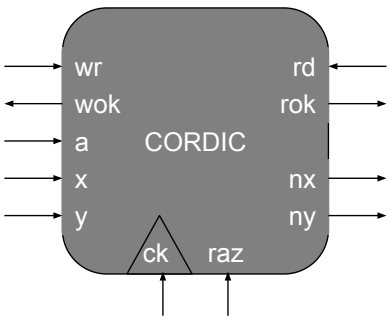
Objectif

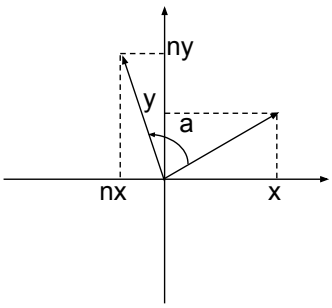
Conception d'un circuit ASIC* avec Alliance

- Passer d'un algorithme à un modèle RTL (modélisation et synthèse logique)
- Passer d'un modèle RTL au dessin des masques (synthèse physique)

ASIC utilisant l'algorithme CORDIC

- Calcul de la rotation d'un vecteur (x,y,a) → (nx,ny)
- L'algorithme est « simple », c'est une boucle for en C
- Il permet plusieurs modèles RTL en fonction des contraintes de réalisation



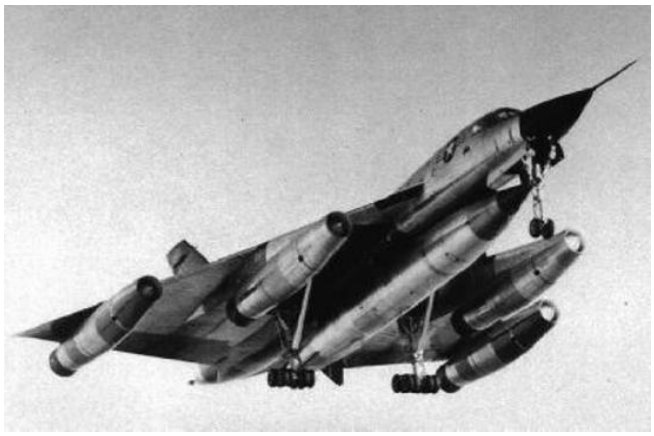


* ASIC (Application Specific Integrated Circuit) ≠ processeur

Algorithme CORDIC

CORDIC signifie COordinate Rotation DIgital Computer

- Algorithme conçu en 1956 par Jack Volder
- Remplacement du calculateur analogique (composant électromécanique) de navigation des B-58.



source : http://aviation-militaire.kazeo.com/convair-b-58-hustler-a121956030

Algorithme CORDIC

source Wikipédia

Usages de CORDIC

- CORDIC permet au départ de calculer les fonctions trigonométriques en utilisant seulement les opérateurs d'addition, de soustraction et de décalage.
- J. Walther a généralisé l'algorithme (Unified CORDIC) pour calculer aussi les fonctions hyperboliques, exponentielles, logarithmiques, de multiplication, de division et de racine carré.
- CORDIC a permis la conception de la 1ère calculatrice scientifique HP-35 en 1972 et il était présent dans les 1ers coprocesseurs de calcul jusqu'au 80487.

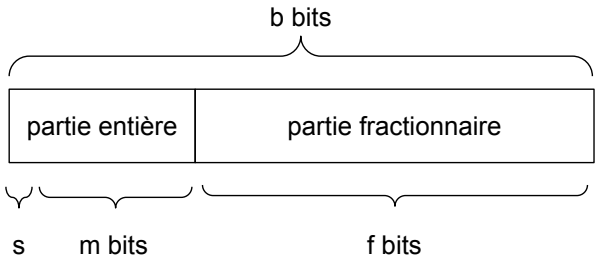
https://www.wikiwand.com/fr/HP-35

- CORDIC demande peu de matériel, peu d'énergie et il est très rapide.

Nombre à virgule fixe

Les nombres sont codés en virgule fixe

- Une partie entière et une partie fractionnaire de taille fixe
- Les nombres sont signés en complément à 2^n ⇒ -a = (2^n - a) ou -a = /a + 1

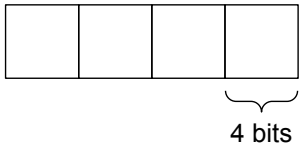


Exemple de codage sur 8 bits : 1-3-4
1 bit de signe
3 bits de partie entière
4 bits de partie fractionnaire

Quelques nombres :

```
1.0  ⇒ 0-001-0000
1.5  ⇒ 0-001-1000
-1   ⇒ 1-111-0000
```

- Dans les calculatrices, les nombres étaient codés en BCD (Décimal Codé Binaire) : chaque chiffre de 0 à 9 est codé sur 4 bits parce que ça simplifie l'affichage



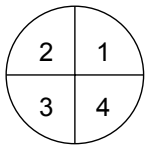
Quelques nombres (entiers naturels)

```
10  ⇒ 0000-0000-0001-0000
 8  ⇒ 0000-0000-0000-1000
```

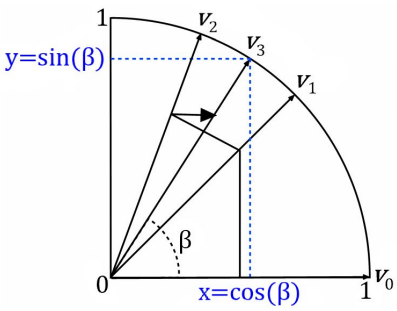

Principes de recherche par dichotomie

Objectif : Calculer le sin(ß) et cos(ß) du vecteur d'angle ß quelconque

Principe



- Supposons que l'angle ß soit entre -90° et +90° (1e et 4e quadrant)
- On utilise une approche "*dichotomique*" (division par 2 de l'angle):
  Rotations successives du vecteur V_0 d'angle 0°
  par des angles $\pm ß_i$ de plus en plus petits tels que
  $-45° <= ß_i <= 45°$ pour approcher le vecteur d'angle ß
- Chaque rotation élémentaire approche le bon résultat
  $ß = 0° + \sum \pm ß_i$ avec i compris entre 0 et n
- On connait le cos(0°) et on suppose être capable de calculer $cos(ß) \approx cos(0° + \sum \pm ß_i)$ (resp. pour sin(ß))
- L'astuce c'est de bien choisir les $ß_i$



Rotation élémentaire : choix des $ß_i$

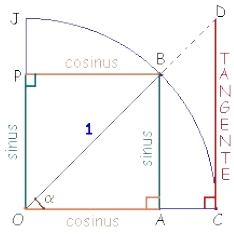
Rotation élémentaire

- La rotation positive élémentaire d'un vecteur v_i par un angle $ß_i$ rend v_{i+1}

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} cos(ß_i) & -sin(ß_i) \\ sin(ß_i) & cos(ß_i) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

- En mettant cos($ß_i$) en facteur

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = cos(ß_i) \begin{pmatrix} 1 & -tan(ß_i) \\ tan(ß_i) & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$



http://www.trigofacile.com/maths/trigo/notions/fonctions/tangente.htm

- Puisque -45° < $ß_i$ < 45° alors -1 <= $tan(ß_i)$ <= 1

- On choisi $ß_i = \sigma_i \arctan(2^{-i})$ ⇒ $\sigma_i \tan(ß_i) = \sigma_i \tan(\arctan(2^{-i})) = \sigma_i 2^{-i}$ avec $\sigma_i = \pm 1$ en fonction du sens de la rotation

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = cos(arctan(2^{-i})) \begin{pmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

⇒ Soit $K_i = cos(arctan(2^{-i}))$ et sachant que $a * 2^{-i} = a \gg i$ *(≫ décalage droite)*
alors on obtient : $x_{i+1} = K_i*(x_i - \sigma_i*y_i*2^{-i})$ $\quad x_{i+1} = K_i*(x_i - \sigma_i*y_i \gg i)$
$y_{i+1} = K_i*(y_i + \sigma_i*x_i*2^{-i})$ $\quad y_{i+1} = K_i*(y_i + \sigma_i*x_i \gg i)$

Algorithme complet

- On effectue **n+1** rotations du vecteur (1,0) *(**n** est le nombre de bits après la virgule)*
soit $K = \prod_{i=0}^{n} K_i = \prod_{i=0}^{n} cos(arctan(2^{-i}))$ *(si **n** est connu alors K est une constante)*

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = K \prod_{i=0}^{n} \begin{pmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- On connaît les angles $ß_i$ que l'on peut mettre dans une table
  - $ß_0 = \arctan(2^0) = 0{,}7853$ rad $= 45°$
  - $ß_1 = \arctan(2^{-1}) = 0{,}4636$ rad $= 26{,}57°$
  - $ß_2 = \arctan(2^{-2}) = 0{,}2449$ rad $= 14{,}04°$
  - …
  - $ß_7 = \arctan(2^{-7}) = 0{,}0078$ rad $= 0{,}45°$

on voit que les angles décroissent, c'est presque dichotomique (div 2))

- En résumé, l'algorithme consiste à : *(si l'angle de rotation ß est entre -90° et 90°)*
  - Faire **n** rotations de $\sigma_i ß_i$ avec $\sigma_i = \pm 1$ (sans multiplier par K_i)
  une rotation est une boucle d'itération : $x_{i+1} = x_i - \sigma_i * y_i \gg i$
  $y_{i+1} = y_i + \sigma_i * x_i \gg i$
  - Puis à la fin, à multiplier par la constante K (le produit des K_i)
- La précision du résultat dépend du nombre d'itérations (au plus **n+1**)
⇒ à chaque itération, on ajoute un nouveau chiffre après la virgule !

Principes (rotation)

Prologue et épilogue de l'algorithme

- Si l'angle de départ est dans les 2e, 3e ou 4e quadrants (> 90°)
  alors il faut faire des rotations de ±90° pour amener l'angle dans le 1e quadrants,
  il faut se souvenir de la rotation pour retrouver le signe du résultat

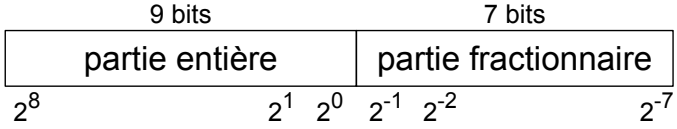
  En effet, on sait : *cos(a+b) = cos(a)cos(b) - sin(a)sin(b)*
  donc on a : *cos(ß+90°) = -sin(ß)*
  on montre aussi que : *sin(ß+90°) = cos(ß)*

- On part donc du vecteur (1,0) et on fait à la fin des itérations, on obtient un vecteur (x, y) qu'il faut multiplier par la constante K : (cos(x), sin(y)) = K ∗ (x, y)
- **On ne veut pas utiliser un multiplieur, on utilise des additions et des décalages**
  soit : $K = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_0 2^0$ où $a_i \in \{0, 1\}$
  $xr = cos(x) = xK = xa_n 2^n + xa_{n-1} 2^{n-1} + \ldots + xa_0 2^0$
  $xr = a_n (x \ll n) + a_{n-1} (x \ll (n-1)) + \ldots + a_0 x$
  ⇒ Si K a peu de bits à 1 (c.-à-d. peu de a_n à 1), il y a peu de termes à sommer

Choix d'un codage en virgule fixe

Codage des nombres pour le circuit réalisé

- Les nombres sont en complément à 2^n
- Nous allons utiliser un codage sur 16 bits : 1-8-7



Opérations

- La conversion est réalisée par de simples décalages
  - Soit E un nombre entier sur 8 bits en complément à 2: E ∈ [-128, 127]
  - Soit F un nombre avec 7 bits après la virgule sur 16 bits :
    On a 9 bits pour la partie entière mais on choisit de limiter F ∈ [-128, 128[
    - -128 = `0b1.1000.0000.0000.000`
    - +128 ≈ `0b0.0111.1111.1111.111` = 127,99609375
  - en 1-8-7 l'intervalle théorique est plus grand mais c'est pour éviter les dépassements de capacité lors des calculs…
  - Les conversions E ←→ F se font par des décalages
    F = E << 7 et E = F >> 7 (dans ce dernier cas, on perd la partie fractionnaire)
- Les opérations arithmétiques +/- restent inchangées

Multiplication par K avec le codage 1-8-7

Rappel de l'algo : pour faire une rotation ß d'un vecteur (a,b)

- on fait les 8 rotations élémentaires $ß_i$ pour obtenir le vecteur intermédiaire (x,y)
- puis, on multiplie par K pour obtenir, le vecteur correct (rx, ry)

$$\begin{pmatrix} x \\ y \end{pmatrix} = \prod_{i=0}^{7} \begin{pmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \qquad \begin{pmatrix} rx \\ ry \end{pmatrix} = K \begin{pmatrix} x \\ y \end{pmatrix}$$

On sait que :

si on multiplie des nombres à virgule, on fait comme s'il n'y avait pas de virgule, puis on place la virgule sur le résultat en sautant autant de chiffres qu'il y en avait dans les opérandes : aa,aa * bb,bb = cccc,cccc

Ici, les nombres sont codés en 1-8-7

⇒ on va multiplier des nombres de 16 bits et faire un décalage à droite de 14 bits

$$K_{real} = \prod_{i=0}^{7} K_i = \prod_{i=0}^{7} cos(arctan(2^{-i})) = 0.607259$$
$$K_{fixed} = 0.607259 \ll 7 = 78 = 0x4E = 0000000001001110$$
$$rx = ((x \ll 6) + (x \ll 3) + (x \ll 2) + (x \ll 1)) \gg 14$$

Pour ne pas avoir à coder inutilement ***rx*** sur 32 bits, on peut anticiper les décalages

$$rx = (((x \gg 7) \ll 6) + ((x \gg 7) \ll 3) + ((x \gg 7) \ll 2) + ((x \gg 7) \ll 1)) \gg 7$$
$$rx = ((x \gg 1) + (x \gg 4) + (x \gg 5) + (x \gg 6)) \gg 7$$

Exemple du calcul de cosinus avec CORDIC

Calcul de *cos(78°) = cos(1,3613 rad) = 0.2079*

- Ici, on va faire seulement 4 itérations et utiliser la base 10.
- On va donc faire 4 rotations d'angle $ß_i$ (i de 0 à 3) du vecteur (1,0)
  - $ß_0$ = arctan (2^0) = 0,7853 rad = 45°
  - $ß_1$ = arctan (2^{-1}) = 0,4636 rad = 26,57°
  - $ß_2$ = arctan (2^{-2}) = 0,2449 rad = 14,04°
  - $ß_3$ = arctan (2^{-3}) = 0,1243 rad = 7.125°
- *78 → 45 + 26,57 + 14,04 - 7,125 = 78.48°*
- K_0= cos(arctan(2^0)) = 0,7071 ; K_1= 0,8944 ; K_2= 0,9701 ; K_3= 0,9922 ;
- Etapes *(ici on multiplie par K_i à chaque étape pour plus de clarté)*

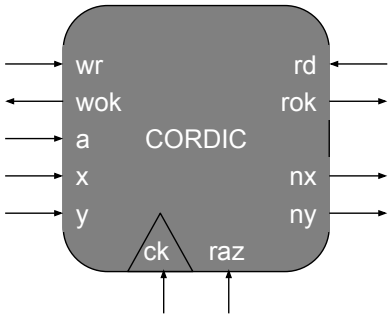
$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = K_0 \begin{pmatrix} 1 & -2^0 \\ 2^0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0.7071 \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}$$
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = K_1 \begin{pmatrix} 1 & -2^{-1} \\ 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = 0.8944 \begin{pmatrix} 0,7071 - 0,7071/2 \\ 0.7071/2 + 0,7071 \end{pmatrix} = 0,8944 \begin{pmatrix} 0,3535 \\ 1.060 \end{pmatrix} = \begin{pmatrix} 0.3162 \\ 0,9486 \end{pmatrix}$$
$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = K_2 \begin{pmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = 0.9701 \begin{pmatrix} 0.3162 - 0.9486/4 \\ 0.3182/4 + 0.9486 \end{pmatrix} = 0.9701 \begin{pmatrix} 0.079 \\ 1.028 \end{pmatrix} = \begin{pmatrix} 0.077 \\ 0,9974 \end{pmatrix}$$
$$\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} = K_3 \begin{pmatrix} 1 & 2^{-3} \\ -2^{-3} & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = 0.9922 \begin{pmatrix} 0,077 + 0.9974/8 \\ -0,077/8 + 0.9974 \end{pmatrix} = 0.9922 \begin{pmatrix} 0.2016 \\ 0,9877 \end{pmatrix} = \begin{pmatrix} 0.2000 \\ 0,9800 \end{pmatrix}$$

- cos(78°) = 0,2079 ≈ 0,200 (3,8% d'erreur)
  notez que l'on calcule aussi sin(78°) avec moins de 1% d'erreur (sin(78°)=0.9781)

Circuit CORDIC

Objectif

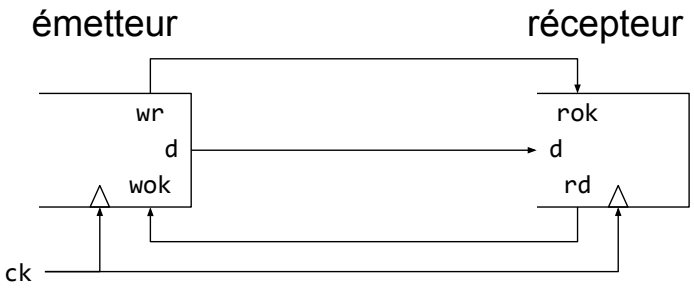
modéliser et valider un circuit CORDIC
calculant les coordonnées **(nx,ny)**
d'un vecteur **(x,y)** après la rotation
d'un angle a.



Caractéristiques

- CORDIC présente une interface FIFO en entrée et en sortie.
- Les nombres à l'interface sont des entiers sur 8 bits (pas de virgule)
- La latence et le débit des calculs dépendent de l'architecture interne

Interface de communication FIFO



- La communication est synchronisée sur une horloge ck
- L'émetteur et le récepteur sont indépendants
  chacun dit s'il est prêt à envoyer ou à recevoir une donnée
  - si **`wr`** est actif l'émetteur a une donnée **`d`** et informe le récepteur sur **`rok`**
  - si **`rd`** est actif le récepteur a une place pour **`d`** et informe l'émetteur sur **`wok`**
  - si wr et rd sont actifs au même cycle alors une donnée **`d`** est transmise
- Notez une chose importante concernant le choix du nom des signaux de contrôle (j'ai mis du temps à comprendre ça…) :
  - **wr** et **rd** sont des **ordres** donc des **sorties**
  - **`wok`** et **`rok`** sont des **informations** d'état donc des **entrées**

Modèle comportemental en C

Le comportement du circuit sans horloge

```
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));
}
```


Modèle comportemental en C

```
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;          // angle de départ
    x = x_p << 7;     // coordonnée x initiale
    y = y_p << 7;     // coordonnée y initiale

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

    // 8 rotations successives
    for (i = 0; i <= 7; i++) {
        dx = x >> i;
        dy = y >> i;
        if (a >= 0) {     // rotation positive
            x -= dy;      // calcul des coordonnées
            y += dx;      // après rot de +ATAN[i]
            a -= ATAN[i]; // nouvel angle de rot.
        } else {          // rotation négative
            x += dy;      // calcul des coordonnées
            y -= dx;      // après rot de -ATAN[i]
            a += ATAN[i]; // nouvel angle de rot.
        }
    }
    // produit du résultat par les cosinus
    // des angles : K=0x4E=01001110
    x = ((x>>6) + (x>>5) + (x>>4) + (x>>1))>>7;
    y = ((y>>6) + (y>>5) + (y>>4) + (y>>1))>>7;

    // placement du point dans le quadrant initial
    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;
}
```

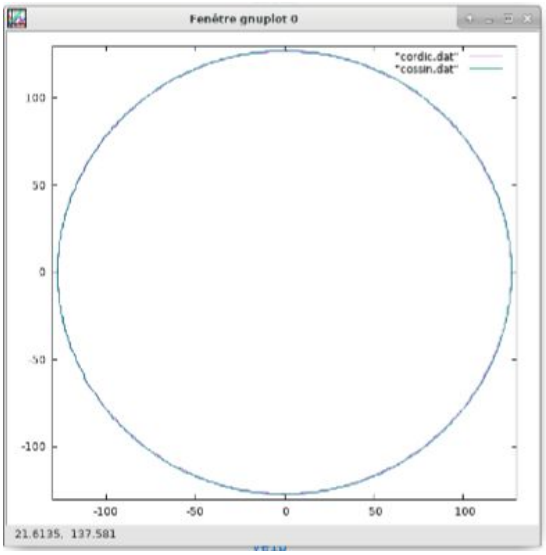
```
// variable globale
short ATAN[8] = {
    0x65, // ATAN(2^-0)
    0x3B, // ATAN(2^-1)
    0x1F, // ATAN(2^-2)
    0x10, // ATAN(2^-3)
    0x08, // ATAN(2^-4)
    0x04, // ATAN(2^-5)
    0x02, // ATAN(2^-6)
    0x01, // ATAN(2^-7)
};
```

cossin() = cordic()

```
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));
}
```

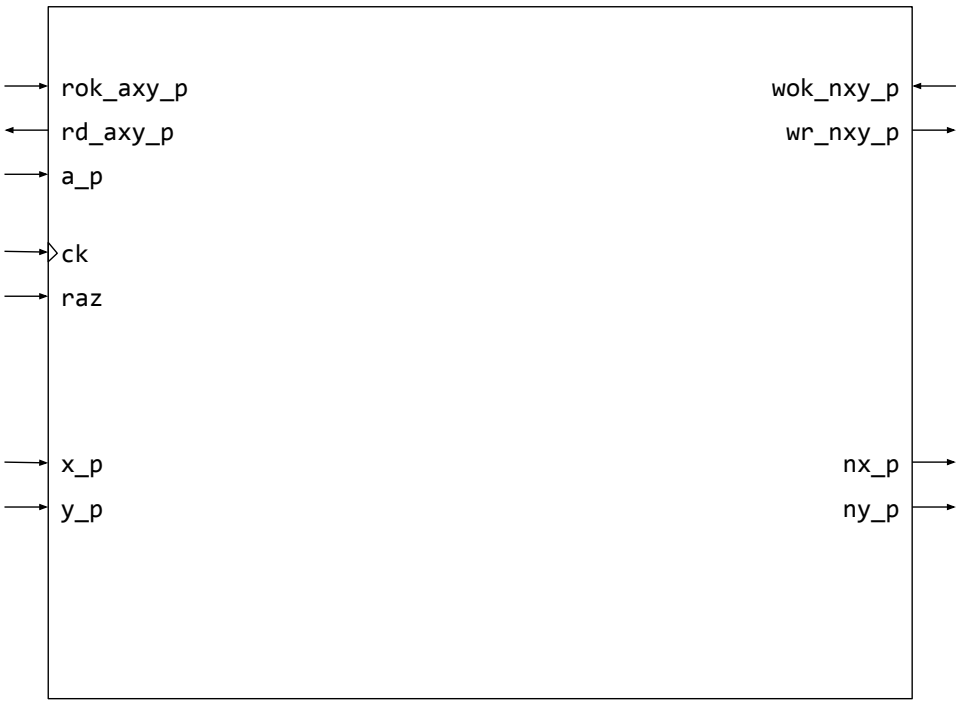
différences entre cossin() et cordic()

```c
int main()
{
    FILE *f;
    double t,K;
    int i;

    f = fopen("cossin.dat", "w");
    for (double a = 0; a <= M_PI * 2; a += 1. / 64) {
        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;
}
```

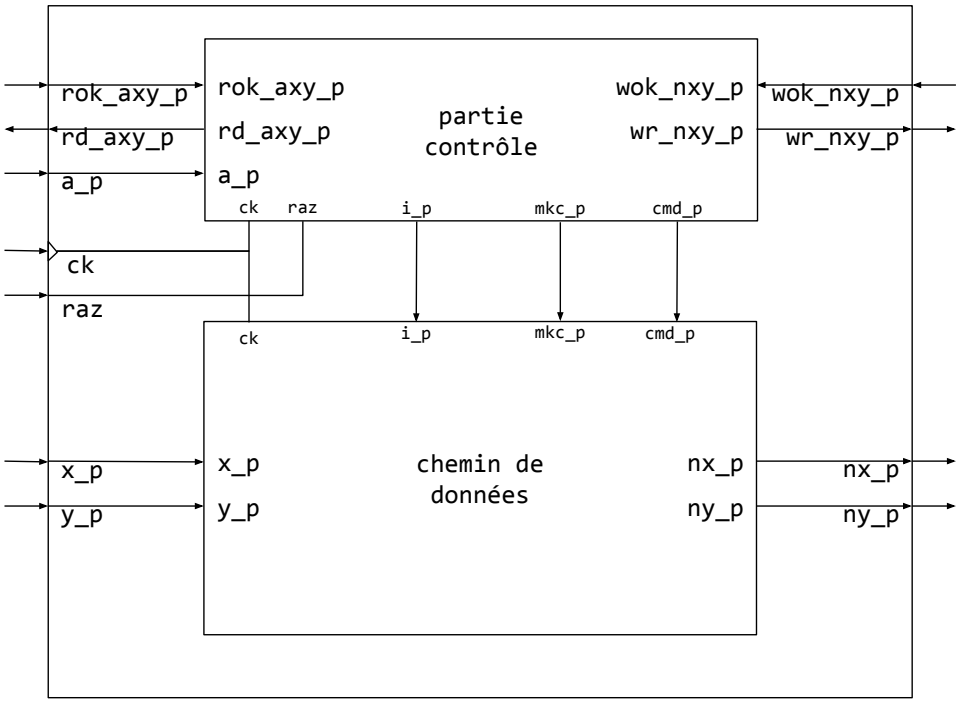



```
plot: cercle
    ./cercle
    gnuplot \
            -e 'plot [-130:130] [-130:130] "$(MODEL).dat" with lines;'\
            -e 'replot "cossin.dat" with lines;' \
            -e 'pause -1'
```

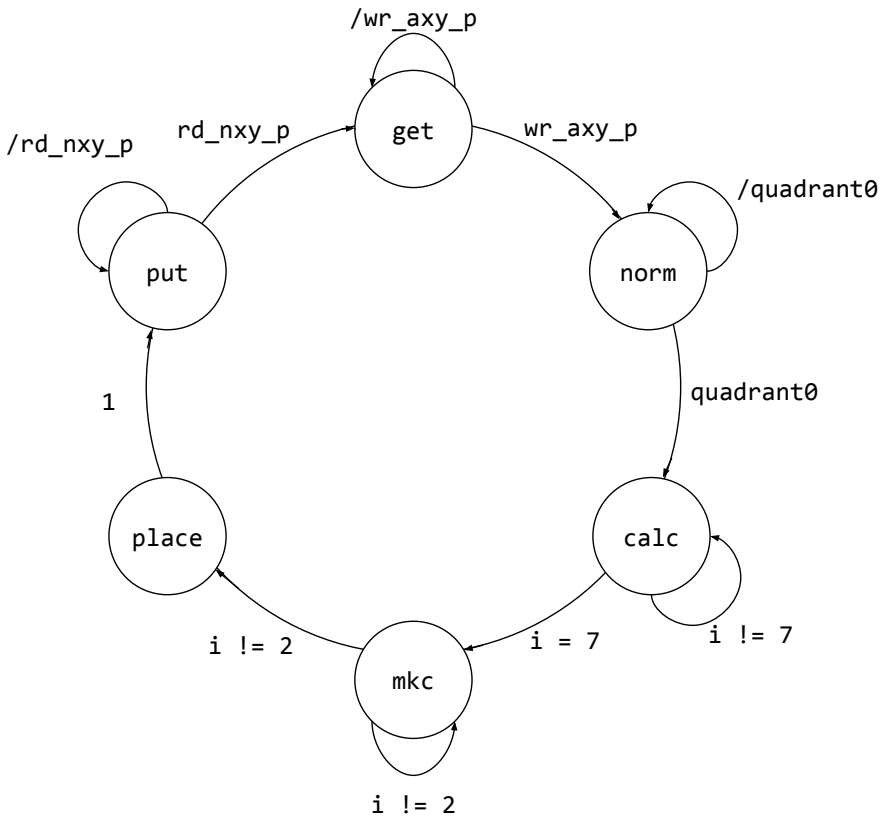

Interface de cordic



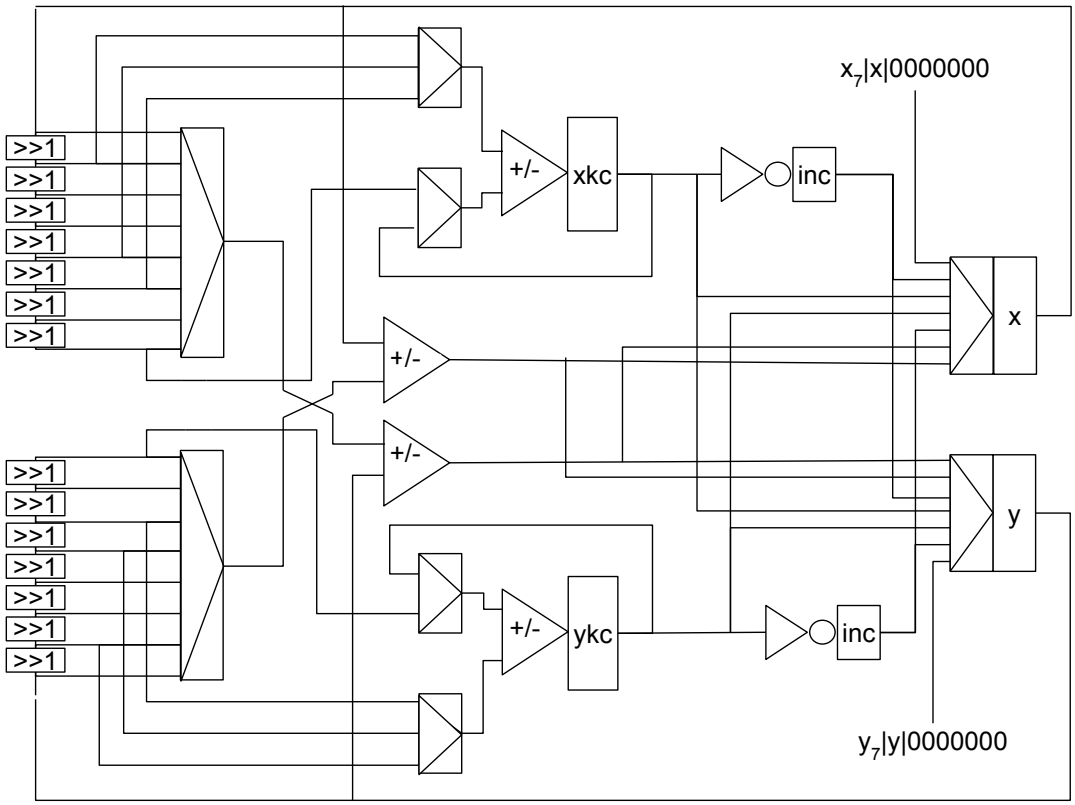
Décomposition en partie contrôle et données



automate



Chemin de données



Travaux Pratiques - PGCD

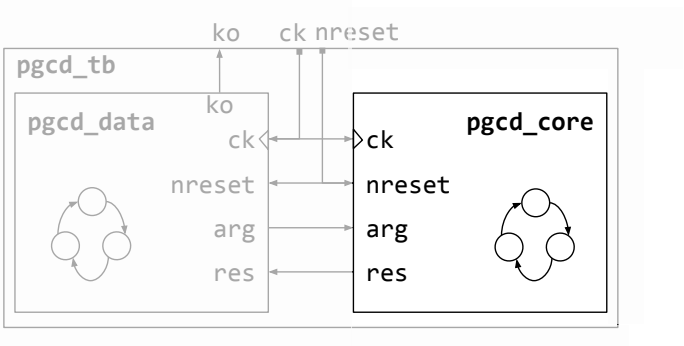
Le but est de réaliser un ASIC complet (peut-être avec les plots)

Vous partez :

- d'un fichier VHDL imcomplet

Vous devez pour le moment :

- compléter et valider le modèle VHDL du PGCD
- synthétiser pour obtenir une netlist sur SXLIB

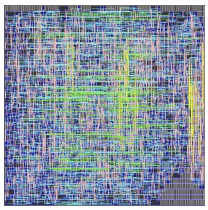


Travaux Pratiques - Cordic

Vous allez travailler sur Cordic, le code qui vous est donné fonctionne, mais vous devez le faire évoluer, Il y a plusieurs possibilités de difficultés croissantes :

- Créer un test bench comme pour PGCD (vous n'êtes pas obligé de faire beaucoup de tests, car on ne va pas fabriquer le circuit…)
- Réduire le nombre d'entrées-sorties
  - entrer x, y et a séquentiellement
  - sortir nx et ny séquentiellement
- Augmenter le débit en créant un pipeline à deux ou trois étages, p. ex.
  - lecture et normalisation
  - calcul
  - placement + multiplication et écriture

Vous devez synthétiser Cordic sur SXLIB.



Pour le placement-routage on pourrait aussi décrire le chemin de données avec DPLIB mais nous ne le ferons pas…