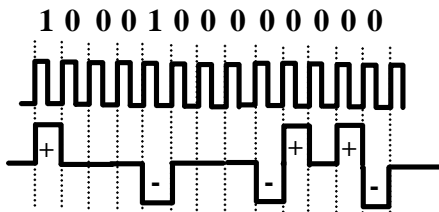


LES OUTILS WARP

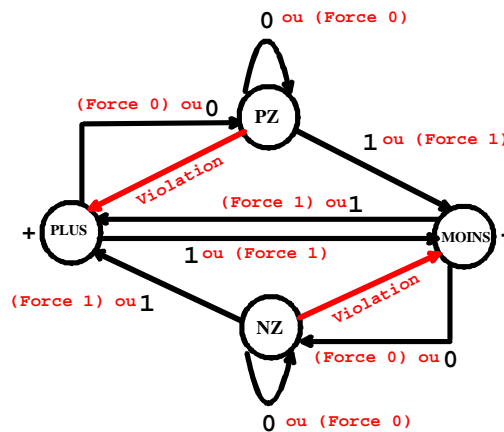
CANALTA		
ECA7	MODELE DE CANAL	SCA7
ECA6	DE TRANSMISSION	SCA6
ECA5	DEFAULT A.	SCA5
ECA4		SCA4
ECA3		SCA3
ECA2		SCA2
ECA1		SCA1
PG. 23/11/98		



```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
package cpt8z_pkg is
component cpt8z
port (CLK: in STD_LOGIC;
nrz_in: in STD_LOGIC;
reset: in STD_LOGIC;
EC: out STD_LOGIC_VECTOR (1 downto 0);
sept_zeros: out STD_LOGIC;
count: inout STD_LOGIC_VECTOR (2 downto 0));
end component;
end cpt8z_pkg;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
```



Stage MAFPEN VHDL Avril 1999.

Auteur : P.Guérangé.

Les outils Warp, synthèse VHDL.

(A) PRESENTATION.....	3
(A-1) LES OUTILS ETUDIES.	3
(A-2) LE TRAVAIL PROPOSE.	4
(B) INITIATION A GALAXY.....	4
(C) SYNTHESE GRAPHIQUE DE MACHINE D'ETATS.....	5
(C-1) PRESENTATION DE L'EXEMPLE TRAITE.	5
(C-2) ANALYSE DU CODE OBTENU.	5
(C-3) INCORPORATION DE LA MACHINE D'ETAT DANS UN PROJET PLUS AMPLE.....	7
(C-4) EXPLOITATION D'UN CODEUR INCREMENTAL.	8
(D) EXERCICES D'APPLICATION.....	10
(D-1) SYNTHESE DE LA MACHINE D'ETAT D'UN CODEUR A.M.I.	10
(D-2) AMELIORATION DU CODEUR AMI.....	11
(D-3) ETUDE D'UN CODEUR B8ZS.....	11
(D-3-1) PRESENTATION DU CODAGE B8ZS.....	11
(D-3-2) PRINCIPE DE FONCTIONNEMENT DU CODEUR.	12
(D-3-3) REPARTITION DES FONCTIONS EN MODULE VHDL.....	13
(E) UTILISATION DE WARP3, LA SYNTHESE SOUS VIEWDRAW.....	15
(E-1) PRESENTATION DU PROBLEME.	15
(E-2) AVANTAGES DE CETTE SYNTHESE	15
(E-3) INCONVENIENTS DE LA SYNTHESE PAR LE SCHEMA.....	17
(E-4) SYNTHESE D'UN DECODEUR DE HAMMING.....	17
(F) CREATION D'UN SYMBOLE SIMULABLE.....	18
(F-1) UTILITE DE LA CREATION DE SYMBOLE.....	18
(F-2) CREATION DU SYMBOLE DU DECODEUR DE HAMMING.....	18

(A) PRESENTATION.

L'objet de ce document est de compléter le cours sur le vhdl par la présentation des outils récents de synthèse et de simulation. A ce titre il ne sera pas fait de description du vhdl en tant que langage, sujet déjà abondamment traité dans les documents précédents.

(a-1) Les outils étudiés.

Les outils à notre disposition et objet de ce stage sont les suivants :

Un gestionnaire de projet : GALAXY, ce gestionnaire permet d'éditer puis de compiler des descriptions vhdl. D'organiser des descriptions hiérarchiques de projets complexes. La compilation fournit les fichiers de programmation des composants.

Un simulateur temporel : ACTIVE HDL SIM, ce simulateur est un simulateur temporel qui permet la simulation en tenant compte du temps de réponse des composants. L'objectif de la simulation est-il besoin de le rappeler est la validation des projets, avant programmation des composants.

Un générateur de machine d'états : ACTIVE HDL FSM, il permet de synthétiser automatiquement le code texte décrivant une machine d'état à partir d'un dessin de la machine.

La liaison avec viewlogic est possible et permet la synthèse directe de code vhdl à partir d'un schéma logique réalisé sous viewdraw. Cet outil est appelé WARP3.

La génération de composants simulables destinés à être intégré dans des schémas plus complexes comprenant d'autres composants logiques, compteurs discrets, mémoire, d'autres pils sera également traité.

(a-2) Le travail proposé.

Le plan du cours et le suivant :

Activité 1 : Traitement d'un fichier existant prise en main de galaxy. Compilation puis synthèse et simulation de ce fichier.

Activité 2 : Synthèse graphique de machine d'états.

Activité 3 : Incorporation et utilisation d'une machine d'états dans une bibliothèque.

Activité 4 : Exercices.

Activité 5 : Synthèse de code vhdl sous viewlogic avec warp3.

Activité 6 : Création de symbole simulable sous viewdraw.

(B) INITIATION A GALAXY.

Pour débiter prenons un exemple élémentaire qui va nous permettre de traverser toute la chaîne depuis la création, (la récupération), du texte jusqu'à la synthèse et la simulation du composant.

Le fichier à traiter est la description d'un compteur avec reset asynchrone. Ce fichier cnt4_ar.vhd est disponible dans le répertoire début des disquettes.

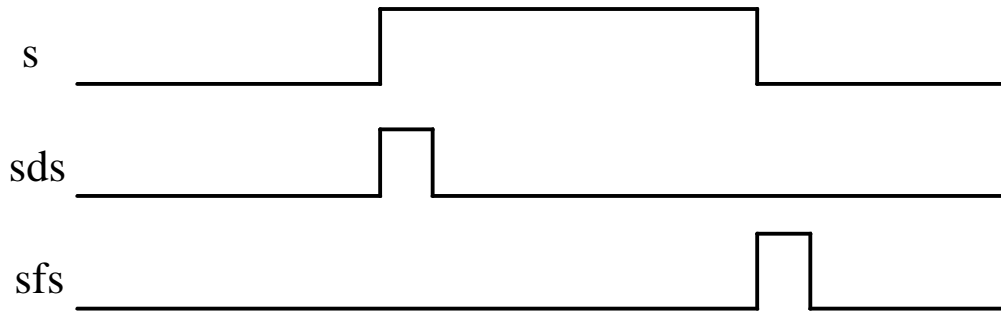
[ACTIVITE 1] A l'aide du document *Prise en main warp51*, réalisez l'ensemble des actions proposées pour arriver à la simulation du compteur avec HDL SIM.

Nous observons dans cette phase que les principales actions précédemment présentes dans warp sont toujours là, à savoir : écriture ou récupération de fichiers textes contenant du code vhdl, désignation du point de départ du projet, choix du composant, choix des options de compilation.

(C) SYNTHESE GRAPHIQUE DE MACHINE D'ETATS.

(c-1) Présentation de l'exemple traité.

Retrouvons la détection de début et de fin d'impulsion comme support de notre étude. Pour mémoire le projet consiste à détecter le début et la fin d'une impulsion selon le chronogramme ci-dessous :



La synthèse fait appel à une machine d'états que nous allons dessiner graphiquement.

[ACTIVITE 2] : A l'aide du document *Fsm*, réalisez la synthèse graphique de la machine d'états.

(c-2) Analyse du code obtenu.

Le code obtenu ressemble au code listé à la page suivante.

L'entité reprend les entrées sorties définies graphiquement. A ce stade le composant n'est pas incorporable comme partie intégrante d'une description plus vaste. Cela sera l'objet de la suite du travail.

Le type `Sreg0_type` est un type de variable énuméré donnant la liste des états de la machine . Le signal `Sreg0` gère la machine.

```

--
-- File: H:\wpat\tous_vhdl\stage_99\fsm\moore1.vhd
-- created: 04/02/99 16:40:46
-- from: 'H:\wpat\tous_vhdl\stage_99\fsm\moore1.asf'
-- by fsm2hdl - version: 2.0.1.45
--
library IEEE;
use IEEE.std_logic_1164.all;

entity moore1 is
    port (CLK: in STD_LOGIC;
          rst: in STD_LOGIC;
          s: in STD_LOGIC;
          sds: out STD_LOGIC;
          sfs: out STD_LOGIC);
end;

architecture moore1_arch of moore1 is

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S0, S1, S2, S3);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;

Sreg0_machine: process (CLK, rst)
begin
if rst='1' then
    Sreg0 <= S0;
elsif CLK'event and CLK = '1' then
    case Sreg0 is
        when S0 =>
            if s='0' then
                Sreg0 <= S0;
            elsif s='1' then
                Sreg0 <= S1;
            end if;
        when S1 =>
            Sreg0 <= S2;
        when S2 =>
            if s='1' then
                Sreg0 <= S2;
            elsif s='0' then
                Sreg0 <= S3;
            end if;
        when S3 =>
            Sreg0 <= S0;
        when others =>
            null;
    end case;
end if;
end process;

-- signal assignment statements for combinatorial outputs
sds_assignment:
sds <= '1' when (Sreg0 = S1) else
    '0' when (Sreg0 = S2) else
    '0';

sfs_assignment:
sfs <= '0' when (Sreg0 = S2) else
    '1' when (Sreg0 = S3) else
    '0';

end moore1_arch;

```

(c-3) Incorporation de la machine d'état dans un projet plus ample.

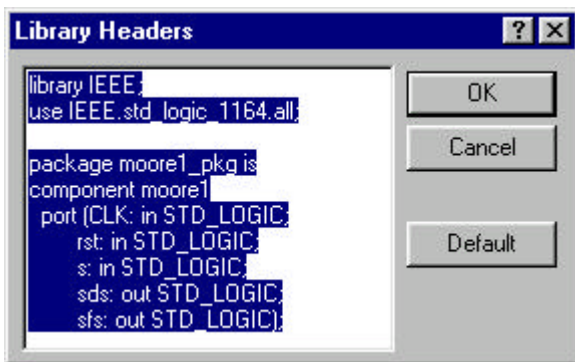
Pour intégrer notre description dans un projet plus ample il faut modifier le code de la machine ci-dessous pour y déclarer le paquetage correspondant.

Ce paquetage rend alors possible l'utilisation de cette description par un module de hiérarchie plus élevé.

Nous pouvons modifier directement le code texte issu de la traduction automatique mais cela rend cette modification caduque dès que nous intervenons sur le graphe de la machine d'état.

La solution consiste à incorporer les lignes de codes supplémentaires directement dans la synthèse graphique, via HDL FSM, de la machine d'état. Il suffit de taper ces lignes dans :

Synthesis -> Select -> Libraries



```
library IEEE;
use IEEE.std_logic_1164.all;

package moore1_pkg is
component moore1
port (CLK: in STD_LOGIC;
rst: in STD_LOGIC;
s: in STD_LOGIC;
sds: out STD_LOGIC;
sfs: out STD_LOGIC);
end component;
end moore1_pkg;
```

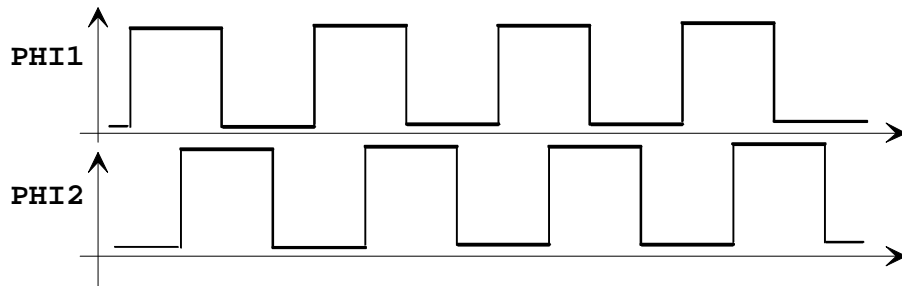
Nous allons exploiter cette technique dans le paragraphe suivant. L'appel du composant se fait simplement dans l'architecture en passant la liste des entrées sorties :

```
c1 : moore1 port map ( clk,raz,phil,deb_phil,fin_phil );
```

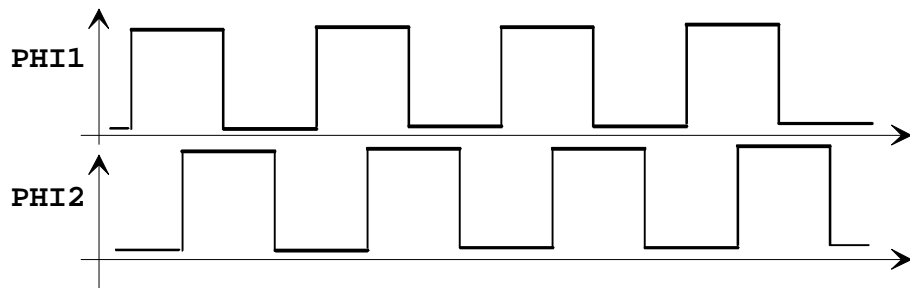
Nous voyons que la correspondance entre les entrées sorties est faite grâce à l'ordre des déclarations.

(c-4) Exploitation d'un codeur incrémental.

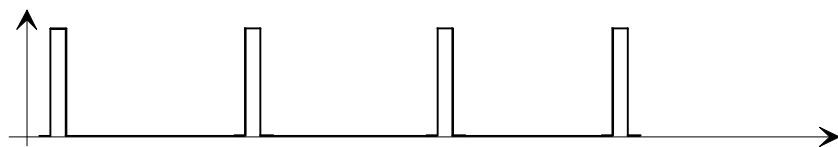
Un codeur incrémental est un codeur qui fournit deux trains d'impulsions PHI1 et PHI2 déphasés d'un quart de période. L'exploitation de ces trains permet de connaître le déplacement d'un mobile.



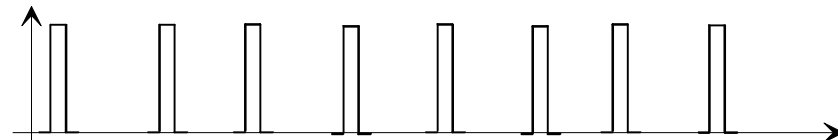
Selon la prise en compte des différents fronts des signaux on parle d'exploitation simple, double et quadruple.



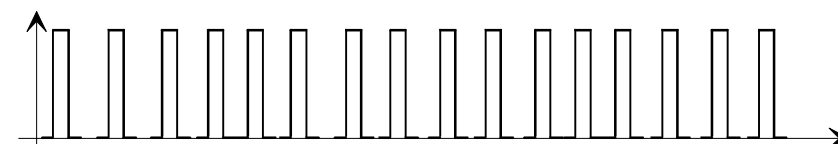
Exploitation simple



Exploitation double

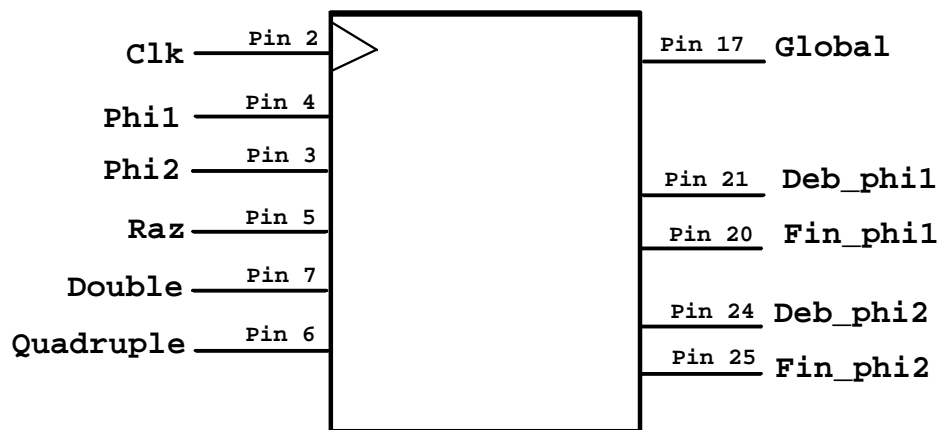


Exploitation quadruple



[ACTIVITE 3] : Réaliser la synthèse de l'exploitation simple, double ou quadruple d'un codeur incrémental. En utilisant la détection de début et de fin d'impulsion traitée en c-3, et conformément aux cahier des charges ci-dessous.

L'entité sera réalisée conformément au schéma ci contre. Les numéros de broche seront imposés de manière à respecter le routage de la maquette disponible. L'attribut pin_numbers le permet.



Le composant retenu pour la synthèse est un iSPGAL22V10 en boîtier PLCC 28 broches de Lattice.

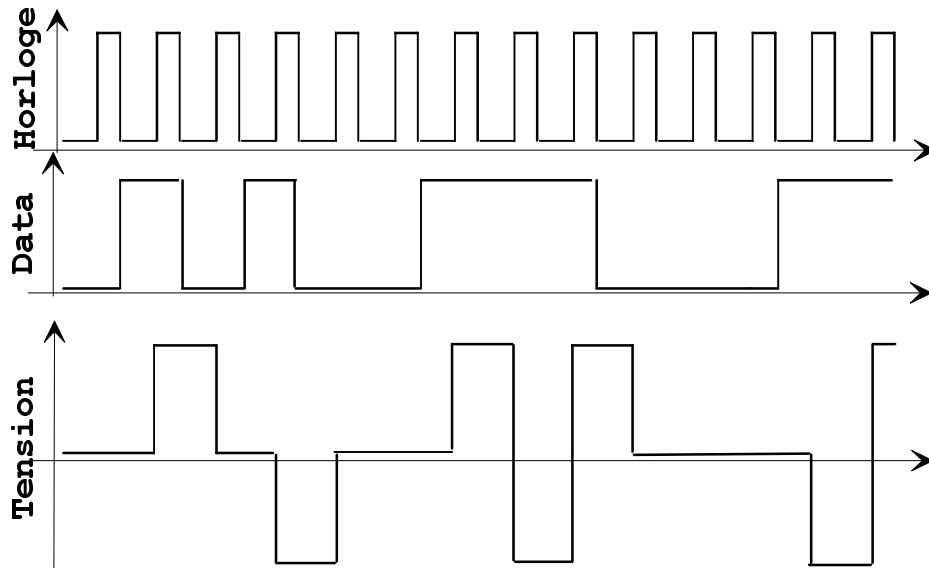
La programmation du composant et les essais seront faits avec les Kits Lattice iSP et les maquettes disponibles.

(D) EXERCICES D'APPLICATION.

[Activité 4] : Certains de ces exercices ont été traités en texte dans le cours précédent, nous allons ici mettre en œuvre les nouvelles techniques de synthèse pour les résoudre de nouveau.

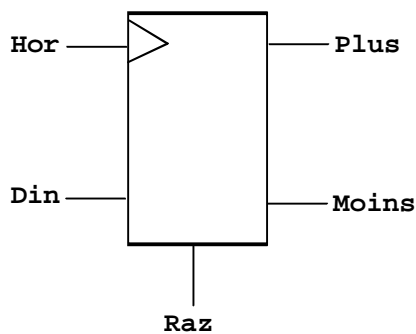
(d-1) Synthèse de la machine d'état d'un codeur A.M.I.

Ce codeur transforme le code binaire en un code à trois niveaux.

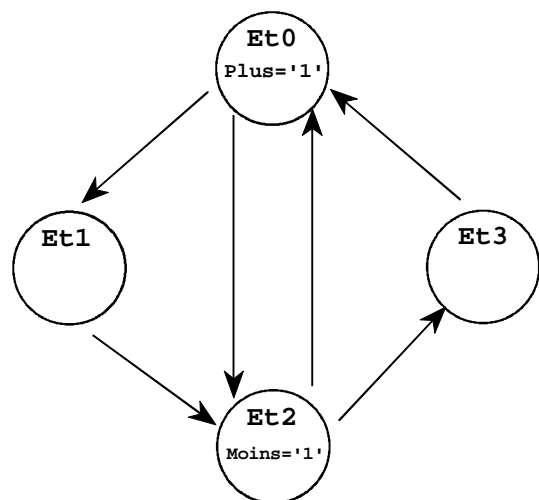


Un zéro correspond à une tension nulle. Les niveaux logiques un sont représentés par des impulsions qui durent une période de l'horloge et alternativement positives et négatives.

Réaliser la synthèse du codeur selon le bilan des entrées sorties ci-dessous, avec l'éditeur graphique de machines d'états.



La machine d'état ci-dessous représente le fonctionnement du codeur.

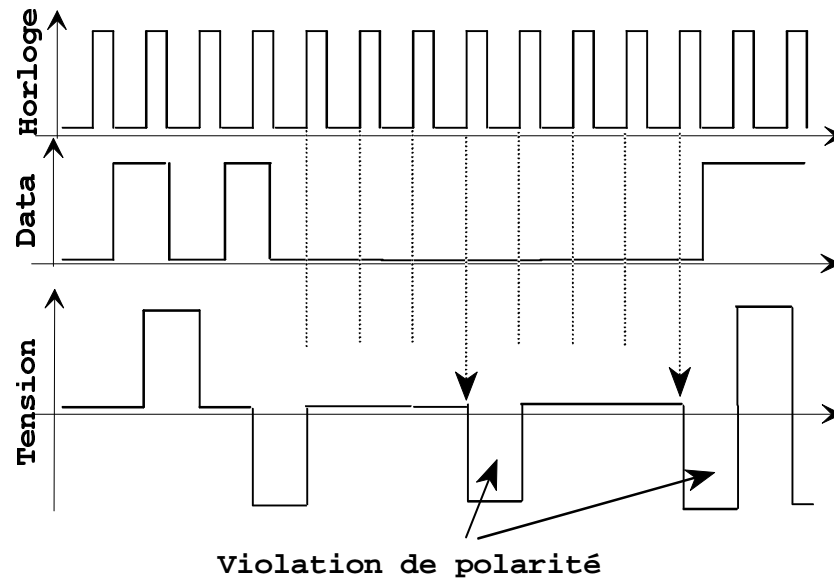


(d-2) Amélioration du codeur AMI.

Une manière d'améliorer le code précédent est dans le cas où il y aurait une longue suite de zéro à transmettre d'envoyer quand même une impulsion.

Cette impulsion est envoyée tous les trois zéros consécutifs, pour éviter de la confondre avec l'envoi d'un '1' on viole la règle d'alternance + -. Si le dernier '1' était codé plus alors on envoi un plus, si le dernier '1' était codé moins alors on envoi un moins.

Modifier la machine d'état précédente et faire la synthèse en respectant les nouvelles contraintes.



(d-3) Etude d'un codeur B8ZS.

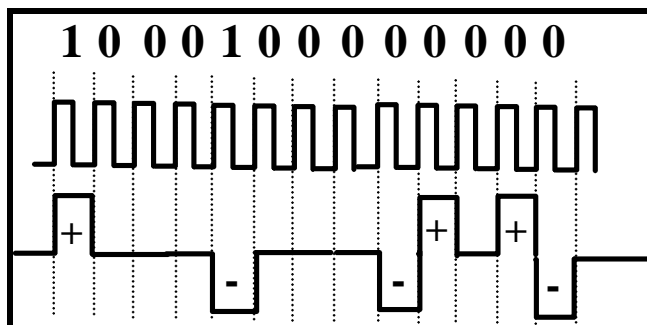
(d-3-1) Présentation du codage B8ZS.

Dans le codage AMI étudié ci-dessus les niveaux logiques '1' sont alternativement positif et négatif, et les niveaux '0' sont de tension nulle. L'horloge est reconstituée à l'aide des fronts montants et descendants présents sur le signal reçu.

Comme une suite de zéros ne contient aucun front, il peut y avoir une perte de synchronisation de l'horloge de réception. Le codage utilisé dans le standard T1 autorise jusqu'à 15 zéros successifs. Si la suite de zéros est trop longue alors le système perdra la synchronisation. Pour la transmission de la voix cela est acceptable mais cela ne l'est pas pour la transmission de données numériques. Une modification du codage AMI permet de pallier cet inconvénient sans restreindre l'utilisateur. C'est le codage B8ZS.

Un codage B8ZS est un code bipolaire avec une substitution de huit zéros consécutifs. Ce code est utilisé dans les réseaux et permet grâce à l'effacement des grandes plages de zéros consécutifs une meilleure reconstruction de l'horloge.

Les niveaux '1' sont codés de la même manière avec une alternance d'impulsions positives et négatives pendant que les zéros sont codés avec une tension nulle. Par contre dès qu'une chaîne de huit zéros apparaît alors elle est remplacée par un code particulier qui dépend de la polarité de la dernière impulsion émise.



Nous voyons dans le dessin ci-dessus que la chaîne de huit zéros a été remplacé par la séquence 000-+0+- car le dernier '1' était codé - ;

De même si le dernier '1' était codé + alors la suite de huit zéros serait codée 000+-0-+.

(d-3-2) Principe de fonctionnement du codeur.

Le codeur mémorise dans un registre à décalage les cinq derniers éléments à coder. Par ailleurs il compte le nombre de zéros consécutifs.

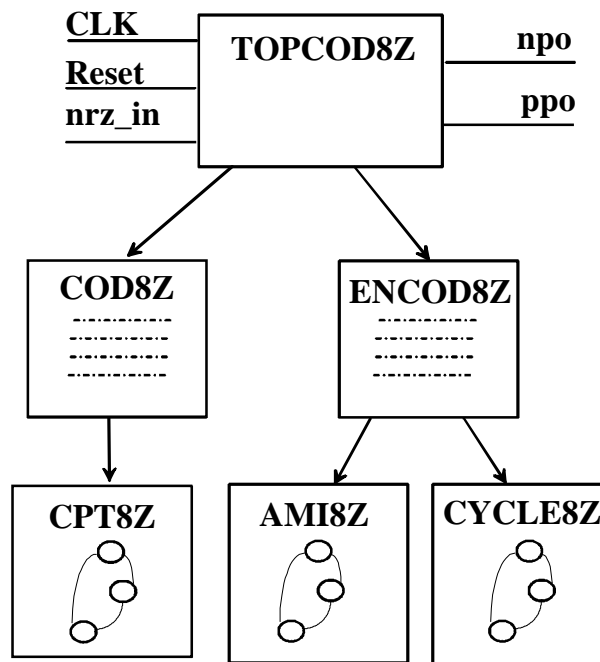
Lorsque l'on atteint huit zéros consécutifs alors le registre à décalage contient les cinq derniers zéros de la suite. Ces zéros sont alors remplacés à l'émission par la séquence B8ZS.

L'émission de la séquence se fait selon les deux machines d'états de la page suivante.

La première contrôle le fonctionnement de la seconde. Si huit zéros successifs sont détectés alors on quitte le mode normal de codage AMI pour générer la séquence de substitution. Le cycle complet pilote alors le codage AMI pour remplacer les cinq derniers zéros par la séquence de substitution. Une fois ce travail effectué on se replace en mode normal.

Un guide complet de réalisation est fourni. L'organisation de la synthèse est donnée ci-dessous.

Organisation générale de la synthèse



(d-3-3) Répartition des fonctions en module vhdl.

Voici la liste des fonctions des différents modules vhdl présentés dans l'organisation générale du projet.

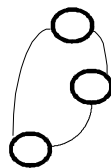
COD8Z	Comptage des zéros Décalage du flux d'entrée de cinq étages.
ENCOD8Z	Encodeur AMI avec ajout de la séquence B8ZS si nécessaire.
CPT8Z	Compteur de sept zéros successifs.
AMI8Z	Encodeur AMI amélioré.
CYCLE8Z	Supervision du codage AMI ou création séquence B8ZS.

Bien sur cette proposition ne constitue qu'une solution possible. L'ensemble des modules réalise le codeur B8ZS.

A chacun de ces modules correspond un fichier vhdl dont le TOPCOD8Z constitue le set top, point de départ de l'arborescence de la description.

Le pictogramme indique l'origine du module vhdl :

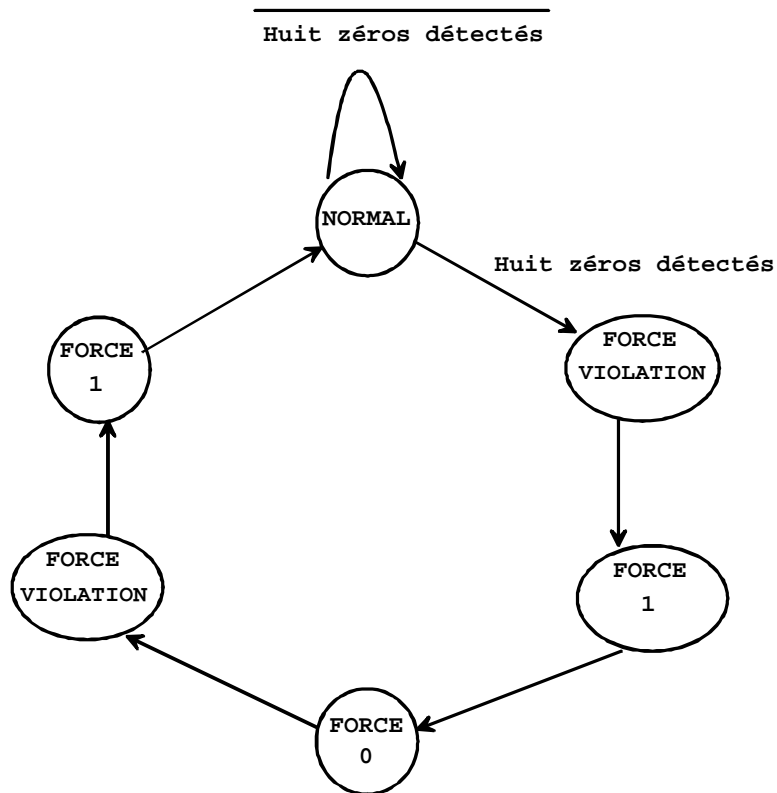
Machine d'état



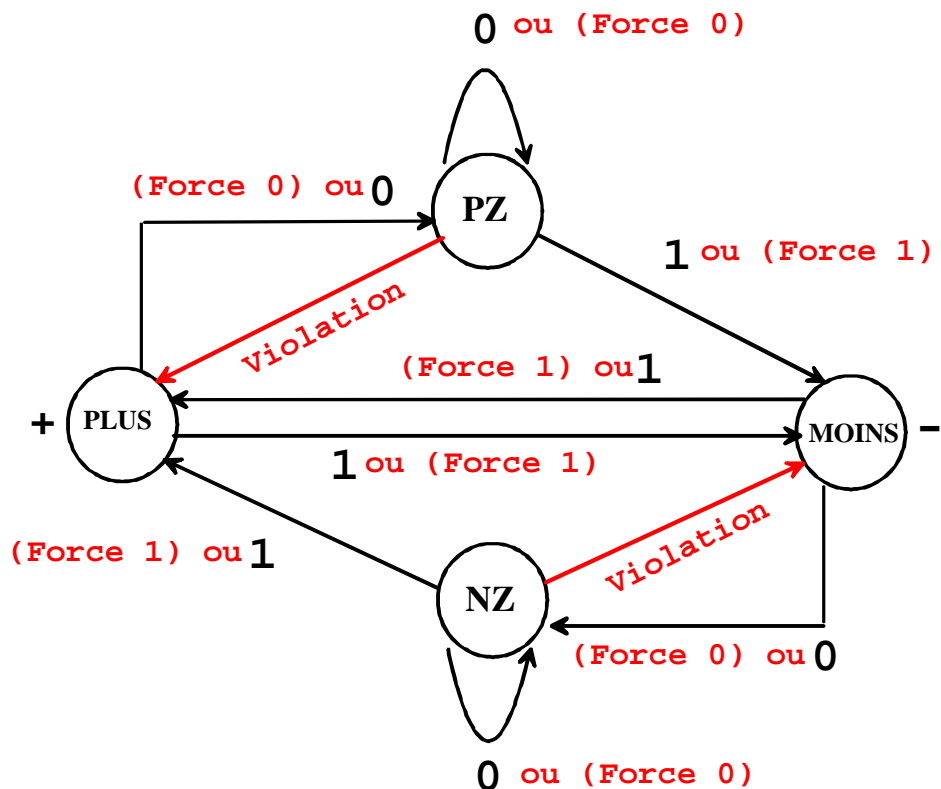
Texte vhdl

.....
.....
.....
.....

Machine d'état du mode : AMI ou substitution de séquence.



Codeur AMI asservi à la machine d'état précédente.



(E) UTILISATION DE WARP3, LA SYNTHÈSE SOUS VIEWDRAW.

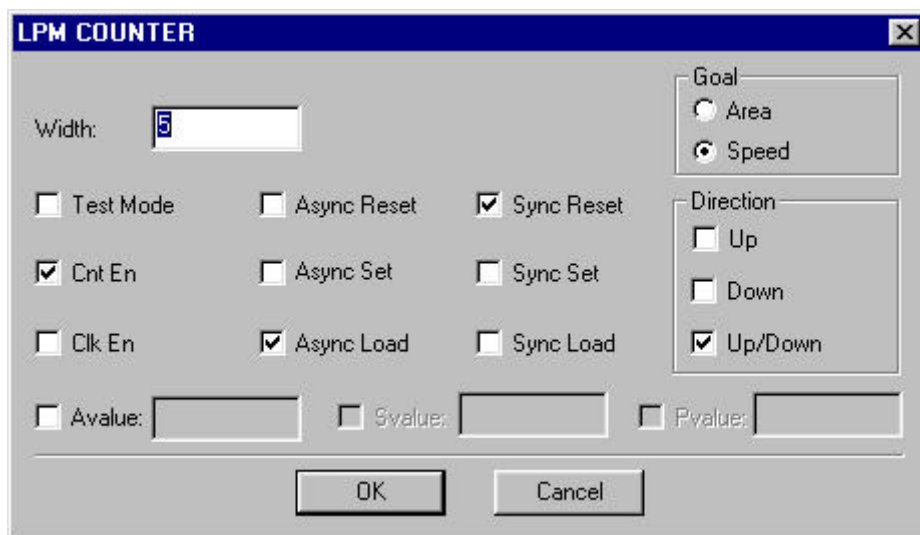
(e-1) Présentation du problème.

La synthèse de code vhdl peut être obtenue à partir d'un schéma réalisé sous viewdraw. Il suffit pour cela d'effectuer ce schéma avec les composants de la bibliothèque lpmlocal fournie par cypress. La transcription du code vhdl est alors faite automatiquement. La compilation et la synthèse sont alors traitées avec galaxy sur le fichier <*>.vhd obtenu.

(e-2) Avantages de cette synthèse .

Pour l'électronicien qui est habitué à penser 'schéma' et est à l'aise avec les circuits logiques combinatoires et séquentiels pas de dépaysement. Grâce au logiciel il va pouvoir associer les fonctions logiques comme pour un schéma traditionnel. La traduction en code vhdl est alors automatique.

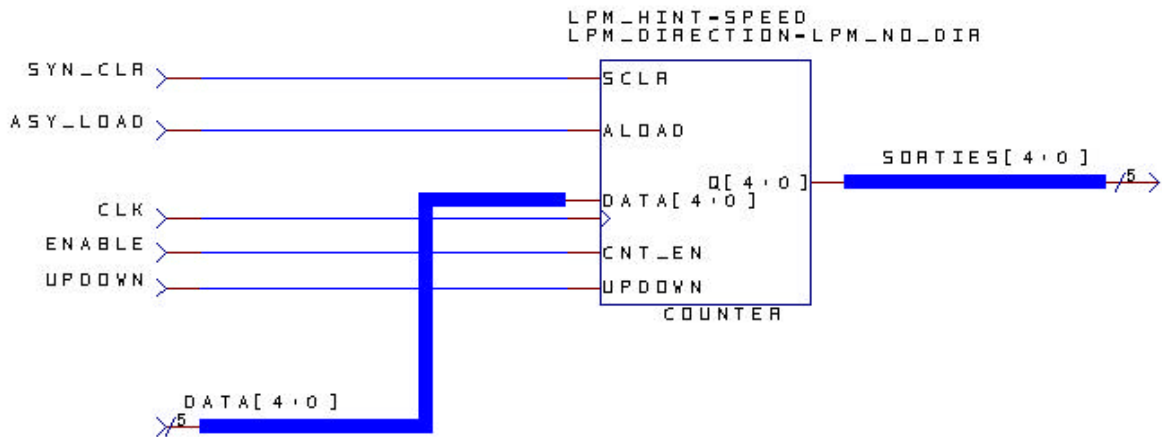
Les composants disponibles dans la bibliothèque n'ont pas de limite de taille, par exemple un compteur peut avoir le nombre d'étages que l'on souhaite, avec chargement synchrone ou non, reset synchrone ou non,... cette grande souplesse est due au fait que les descriptions vhdl de la bibliothèque lpmlocal sont générales et utilisent intensivement la description generic et le passage de paramètres de dimensionnement.



Le compteur générique, ici 5 bascules.

Le document uguide au format pdf décrit l'organisation et le fonctionnement des éléments de la bibliothèque LPM.

Exemple d'un schéma utilisant le compteur précédent :



Un compteur particulier !

Observons dans le schéma ci-dessus l'entrée et la sortie directe des signaux sous la forme d'un bus.

Voici l'architecture correspondante.

```

architecture archcompteur of compteur is
    signal zero: std_logic := '0';
    signal one: std_logic := '1';

begin
    v11i20: mcounter
        generic map(lpm_width => 5,
                    lpm_hint => speed)
        port map(aload => asy_load,
                 clock => clk,
                 cnt_en => enable,
                 data(4) => data4,
                 data(3) => data3,
                 data(2) => data2,
                 data(1) => data1,
                 data(0) => data0,
                 q(4) => sorties4,
                 q(3) => sorties3,
                 q(2) => sorties2,
                 q(1) => sorties1,
                 q(0) => sorties0,
                 sclr => syn_clr,
                 updown => updown,
                 testout => open,
                 testin => zero,
                 testenab => zero,
                 sload => zero,
                 sset => zero,
                 aclr => zero,
                 aset => zero,
                 clk_en => one);
end archcompteur;

```

Architecture de notre description schématique.

L'association avec des machines d'états est possible ainsi que la description hiérarchisée des projets. Voir à ce sujet la synthèse de l'unité arithmétique et logique proposée dans le tutoriel de cypress warp3.

(e-3) Inconvénients de la synthèse par le schéma.

Parallèle : Pour ne pas perturber les habitudes des automaticiens les automates programmables ont été programmés avec des langages à contact. Ces langages étaient identiques aux schémas électriques ils apportaient la souplesse de la programmation dans un environnement familier à l'utilisateur.

Rapidement pour exploiter les possibilités nouvelles des automates de plus en plus performants les langages plus informatiques sont apparus. Les tâches de plus en plus complexes qui leurs étaient assignées ne pouvaient pas être réalisées en langage à contact.

Un automatisme complexe utilise les deux possibilités de programmation en coopération mutuelle au mieux des avantages de chacune.

Peut être vivons-nous le même palier dans la conception de système logique directement avec le vhdl utilisé en tant que langage complet et autonome.

(e-4) Synthèse d'un décodeur de hamming.

Le codeur de hamming permet de détecter et de corriger une erreur dans un mot de sept bits de long.

[ACTIVITE 5] Réaliser la synthèse du décodeur de hamming avec l'aide du document uwarp3.dsf, cette synthèse utilise la saisie sous la forme d'un schéma viewdraw.

(F) CREATION D'UN SYMBOLE SIMULABLE.

(f-1) Utilité de la création de symbole.

Les composants programmables ne sont pas utilisés seuls. Il faut pouvoir intégrer ces composants dans des schémas plus complexes. Ces schémas pourront être simulés pour en tester le bon fonctionnement.

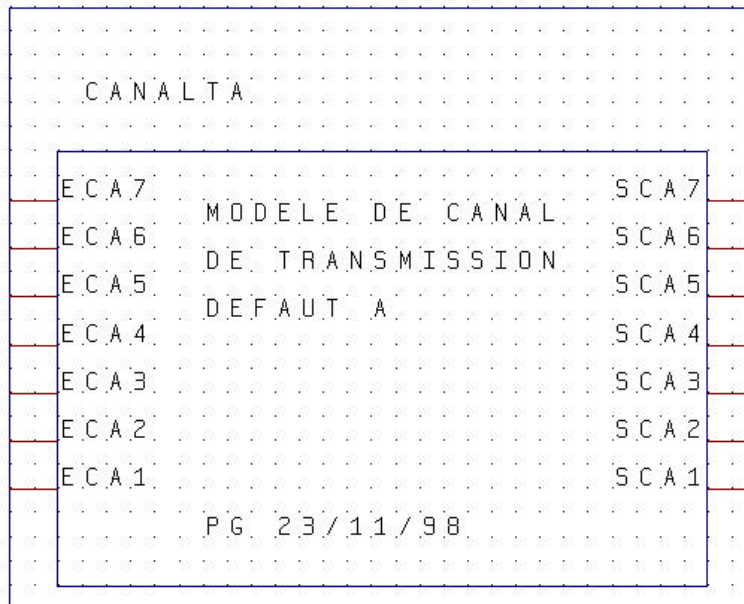
L'environnement viewlogic permet ce travail. Nous allons procéder à la création d'un symbole pour notre décodeur de hamming, puis simuler le schéma obtenu.

(f-2) Création du symbole du décodeur de Hamming.

[ACTIVITE 6] Réaliser le symbole du décodeur de hamming avec l'aide du document *Création d'un symbole utilisable dans un schéma.*

Une fois le symbole créé, réaliser la simulation avec viewsim du décodeur de hamming.

Nous pouvons associer ce décodeur avec un canal de transmission provoquant des erreurs.



Canal de transmission avec défaillance.

Le schéma interne permet de déduire les combinaisons d'entrées qui donnent une erreur lors de la transmission.

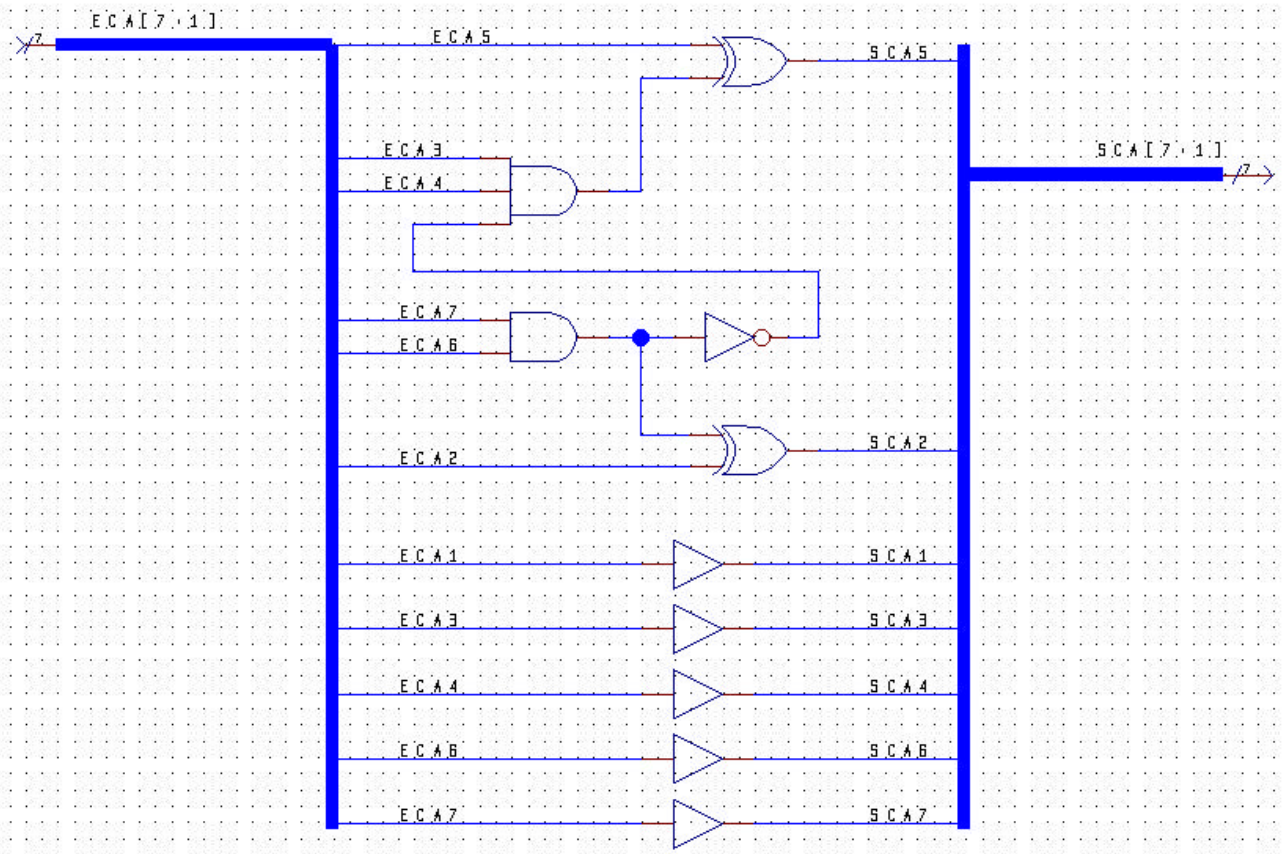


Schéma interne du 'canal'.

Ce modèle a été réalisé avec warp3 puis intégré dans un symbole simulable.

Nous nous trouvons en présence d'un exemple de simulation avec plusieurs composants programmables associés ensemble. Il est possible d'y ajouter des composants simulables avec viewsim comme des mémoires, ou d'autres composants logiques.
