Institut Universitaire de Technologie de CRETEIL-VITRY
Département de Génie Électrique et Informatique Industrielle

SIN1 - Système d'information numérique

TRAVAUX PRATIQUES
XILINX XC9572XL

Année universitaire 2013-2014
J. VILLEMEJANE - julien.villemejane@u-pec.fr

Consignes

- Tous les TP **doivent être préparés** avant la séance
- Faites **valider chacun des exercices** par l'enseignant
- Il vous est conseillé de faire un **compte-rendu** de chacun des TP
- Certains exercices ont déjà été vu en TD, reportez-vous à leur correction

Répartition des séances

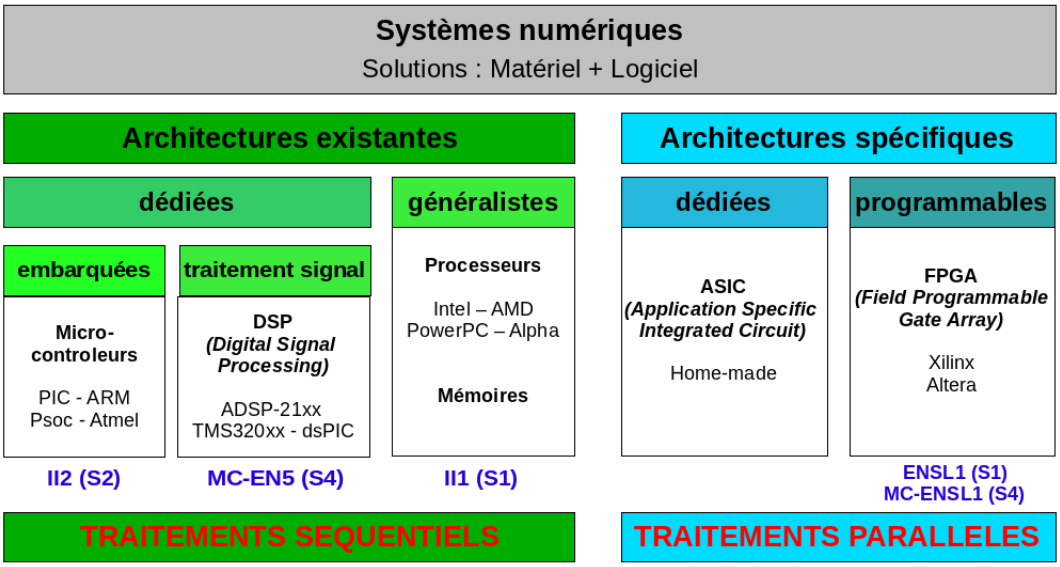
- **Séance 1** : Découverte de l'environnement (TP0) / Systèmes Combinatoires Simples (TP1)
- **Séance 2** : Systèmes Combinatoires Simples (TP1)
- **Séance 3** : Fonctions Combinatoires Standard (TP2)
- **Séance 4** : Synthèse de compteurs (TP3)
- **Séance 5** : Machine à états simple (TP4)
- **Séance 6** : Automate (TP6)
- **Séance 7** : TP Test - 1h30

Table des matières

Cours 1

Représentation comportementale

Dans le domaine des systèmes numériques, il existe deux grands domaines d'applications :
- les **processeurs** (et dérivés : microcontroleurs, DSP...) qui font du traitement **séquentiel**
- les **systèmes logiques** qui font du traitement **parallèle**



1. Synthèse structurelle et description comportementale

Il existe deux moyens pour synthètiser un système numérique logique.

Le premier fait appel aux techniques de synthèse classique ou encore appelée **synthèse structurelle**. A partir d'une table de vérité, on obtient les **équations logiques des sorties**, à partir des entrées, faisant intervenir des **opérateurs logiques** (voir cours sur la logique combinatoire).

Cette méthode permet d'obtenir la **structure finale** du système à concevoir. Chaque opérateur élémentaire possède sa solution technologique sous forme de **circuit intégré** (circuits CMOS : 4001 / porte XOR, 4081 / porte ET, ...).

Une fois la carte électronique réalisée, il n'est alors **plus possible** de modifier la fonction réalisée.

Une autre méthode consiste à utiliser un **composant logique programmable**, laissant ainsi la possibilité de modifier la fonction réalisée à souhait (mise à jour, correction de bugs...).

Il n'est alors plus nécessaire de connaitre la structure que devra avoir le système final, mais de pouvoir simplement décrire son comportement. On parle alors de **description comportementale** du système. Ceci nécessite l'utilisation de **langage de description de haut niveau**, tel que le VHDL ou le Verilog.

2. Composants programmables

Il existe trois grandes catégories de systèmes logiques programmables :
- les **CPLD** (Complex Programmable Logic Device)
- les **FPGA** (Field Programmable Gate Array)
- les **ASIC** (Application Specific Integrated Circuit)

Les CPLD seront présentés dans le TP0.

3. Les FPGA

Ces systèmes programmables sont initialement destinés au **prototypage de systèmes numériques** complexes. Ils sont une bonne alternative aux circuits spécifiques, les ASIC (Application Specific Integrated Circuit), pour des petites ou moyennes séries.

Il existe plusieurs grands fabricants : **ALTERA**, **ACTEL** (composants spécialisés) et **XILINX**. Pour les TP, nous utiliserons des FPGA de chez Xilinx.

3.1. Implantation

Chaque fabricant propose aussi des composants de taille variable : de **100.000** à **10.000.000 portes logiques**. Par comparaison, les portes standards commerciales possèdent entre 2 et 8 portes logiques pour une surface de silicium quasiment identique.

Quelque soit la technologie utilisée, **aucune porte logique** n'est réellement implantée. Il s'agit en fait de **blocs logiques programmables**, mais très *versatiles* (RAM), et d'une mer de connexions programmables. Chez Xilinx, ces blocs logiques sont appelés **CLB** (Common Logic Blocks).

4. Structure d'un FPGA - Xilinx

L'architecture, retenue par Xilinx, se présente sous forme de deux couches : une couche **circuit configurable** et un réseau de **mémoire SRAM**. La structure d'un FPGA est donnée dans la figure suivante. L'échelle est loin d'être réelle, les fonctions logiques n'occupant qu'environ 5% du circuit.

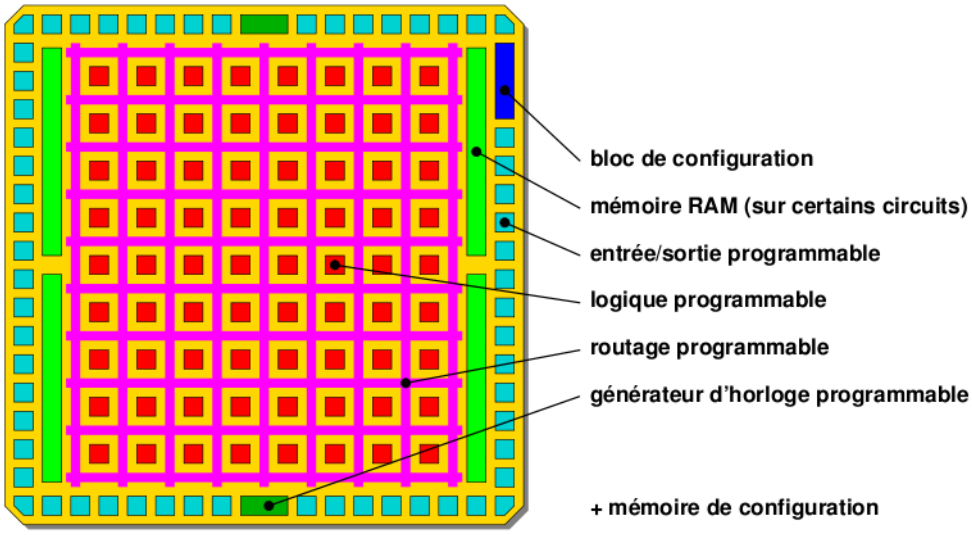


FIGURE 1 – Structure d'un FPGA

Les FPGA sont un rassemblement et une combinaison de différents blocs : **d'entrées/sorties** (IOB - Input Output Blocks), de **routage** (PSM - Programmable Switch Matrix), de **logique programmable** (CLB - Configurable Logic Blocks) et d'autres blocs plus spécifiques.

Cours 2

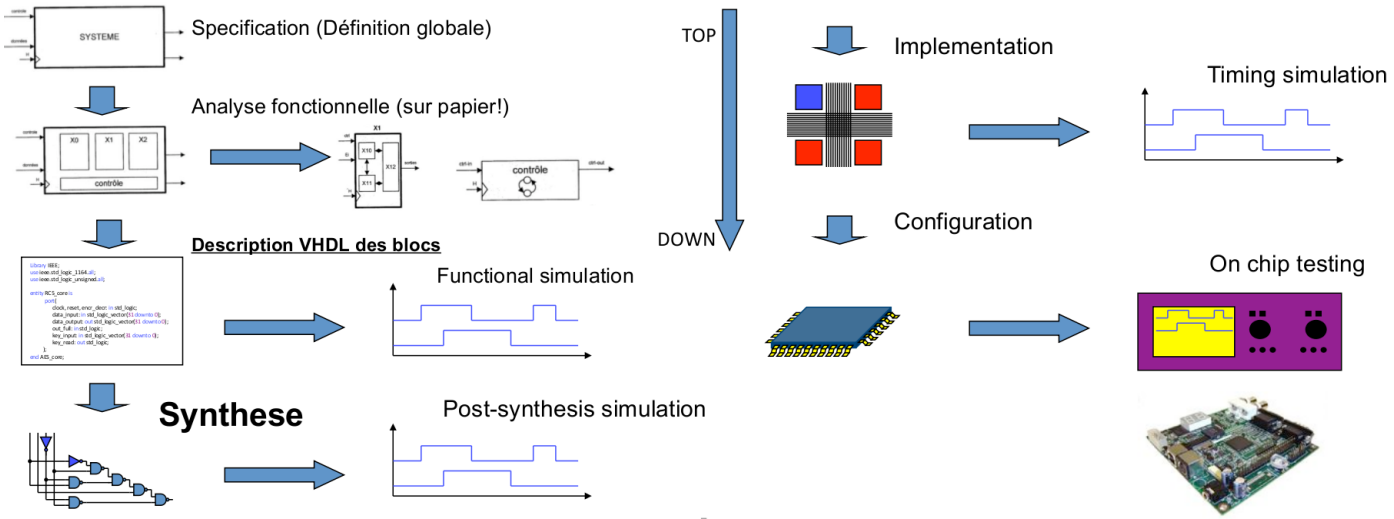
Développement d'un système numérique

Il existe 2 **langages de description** pour les composants logiques programmables :

- VHDL : Very High Speed integrated circuit Hardware Description Language ;
- Verilog.

Nous nous intéresserons par la suite au VHDL, qui est un langage de description normalisé (IEEE) et quasi-universel pour décrire des circuits intégrés.

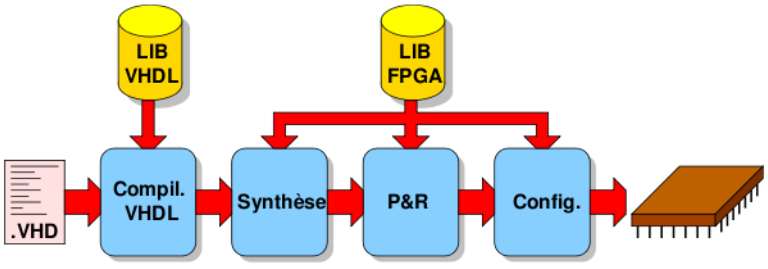
1. Phases de développement



L'**écriture du module VHDL** est la première chose à réaliser.

Ensuite, il est possible (et fortement conseillé) de **vérifier la syntaxe** de la description, en faisant appel à la fonction "*Check Syntax*" dans la partie "*Synthetize XST*". Cette étape est assez rapide et ne nécessite pas de connaître la cible, contrairement à l'étape d'après.

Vient ensuite la phase de synthèse ("*Synthetize XST*") puis de placement et de routage ("*Implement Design*"). Ces étapes nécessitent la connaissance, d'une part, de la cible (FPGA ou CPLD) et, d'autre part, de l'environnement du circuit (entrées/sorties associées aux autres composants de la maquette). Il est donc nécessaire, avant de réaliser ces étapes-là, de faire l'**assignation des broches** du composant avec la description fournie. Pour cela, vous pouvez vous aider de la fonction "*Assign Package Pins*" dans la partie "*User Constraints*" et de la documentation de la maquette fournie en début de ce document.



2. Structure d'un module VHDL

La description d'un système numérique par le biais du langage VHDL passe par 3 étapes différentes :

- la déclaration des ressources externes (bibliothèques) ;
- la description de l'**entité** du système, correspondant à la liste des entrées/sorties ;
- la description de l'**architecture** du système, correspondant à la définition des fonctionnalités du système.

L'ensemble est contenu dans un fichier source portant l'extension `*.vhd`.

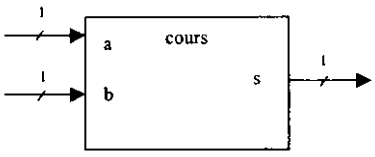
2.1. Déclaration des ressources externes

Cette phase est réalisée automatiquement pour les bibliothèques courantes. On retrouve en en-tête du fichier source `*.vhd` les instructions suivantes :

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

2.2. Entité

L'entité permet de spécifier les différents ports d'entrées/sorties du système. Pour chacun d'entre eux, il est indispensable de donner sa direction :



- **in** entrée simple
- **out** sortie simple
- **buffer** sortie rétroactive
- **inout** entrée-sortie bidirectionnelle (conflits possibles)

et son type (voir section suivante).

```
1  entity cours is
2     port(
3        a, b :    in STD_LOGIC;   -- commentaire
4        s :       out STD_LOGIC
5     );
6  end cours;
```

2.3. Architecture

Une architecture est reliée à une entité et permet de décrire le **fonctionnement du système**. Cette description peut être de deux types :

- description *comportementale* : le comportement du système est décrit (description la plus couramment utilisée en VHDL) ;
- description *structurelle* : la structure meme du système est décrite à base de portes logiques, bascules... (description réservée à des fonctions simples ou pré-calculées).

```
1  architecture Behavorial of cours is
2  -- declaration des signaux
3  begin
4     processus1;
5     processus2;
6     ...
7  end Behavorial;
```

3. Objets et types en VHDL

3.1. Objets

- **signal** objet physique, associé à des évènements
- **variable** intermédiaire de calcul, non physique
- **constant**

3.2. Types

Types de base : **bit**, **bit_vector**, **integer**, **boolean**

Types IEEE : **std_logic**, **std_logic_vector**, **signed**, **unsigned**

Types définis par l'utilisateur :

- type énuméré, exemple : `type jour is (lu, ma, me, je, ve, sa, di);` (souvent utilisé dans les machines à état)

- sous-type : `subtype octet is bit_vector(0 to 7);`

3.3. Notations

bit : `'0'` ou `'1'` ; bit_vector : `"0100"` ; ASCII : `"Texte"` ; Décimal : `423` ; Hexadécimal : `x"1A"`

3.4. Opérateurs en VHDL

LOGIQUES : `and`, `nand`, `or`, `nor`, `xor`, `xnor`, `not`
DÉCALAGE : `sll`, `slr`, `sla`, `sra`, `rol`, `ror`
RELATIONNELS : `=`, `/=`, `<`, `>`, `<=`, `>=`
ARITHMÉTIQUES : `+`, `-`, `*`, `/`, `MOD`
CONCATENATION : **&**
AFFECTATION : **<=**

4. Instructions en VHDL

4.1. Hors processus

Ces instructions décrivent le plus souvent des éléments combinatoires (concurrentes).

4.1.1 Affectation conditionnelle

```
1  x <=  a when cond1 else
2        b when cond2 else
3        ...
4        z;
```

4.1.2 Affectation sélective

```
1  with expr select
2     x <=  a when val1,
3           b when val2,
4           ...
5           z when others;
```

4.2. Dans un processus

L'exécution des instructions s'effectue dans un ordre séquentiel (ordre d'écriture). Le processus est activé lorsqu'un des éléments de la liste de sensibilité est modifié.

La mise à jour des objets s'effectue simultanément à la fin du processus.

4.2.1 Syntaxe

Label : `process`(liste des signaux de sensibilité)
Nom des objets internes : types ; – si nécessaire
`begin`
...
`end process;`

4.2.2 Test : SI

```
1  if x="00" then
2      y <= '0';
3  elsif x="01" then
4      y <= '1';
5  end if;
```

4.2.3 Test : CAS

```
1  case x is
2      when "00" => y <= "00";
3      when "01" => y <= "10";
4      when others => y <= "11";
5  end case;
```

---

5. Tournures fréquentes en VHDL

5.1. Détection d'un front

```
1  if clk'event and clk='1' then ... ; -- front montant
2  if clk'event and clk='0' then ... ; -- front descendant
```

5.2. Remplissage d'un vecteur (bit_vector)

```
1  x <= (others => '0');    -- tous les bits a 0
2  x <= (others => '1');    -- tous les bits a 1
```

GEII - **SIN1**

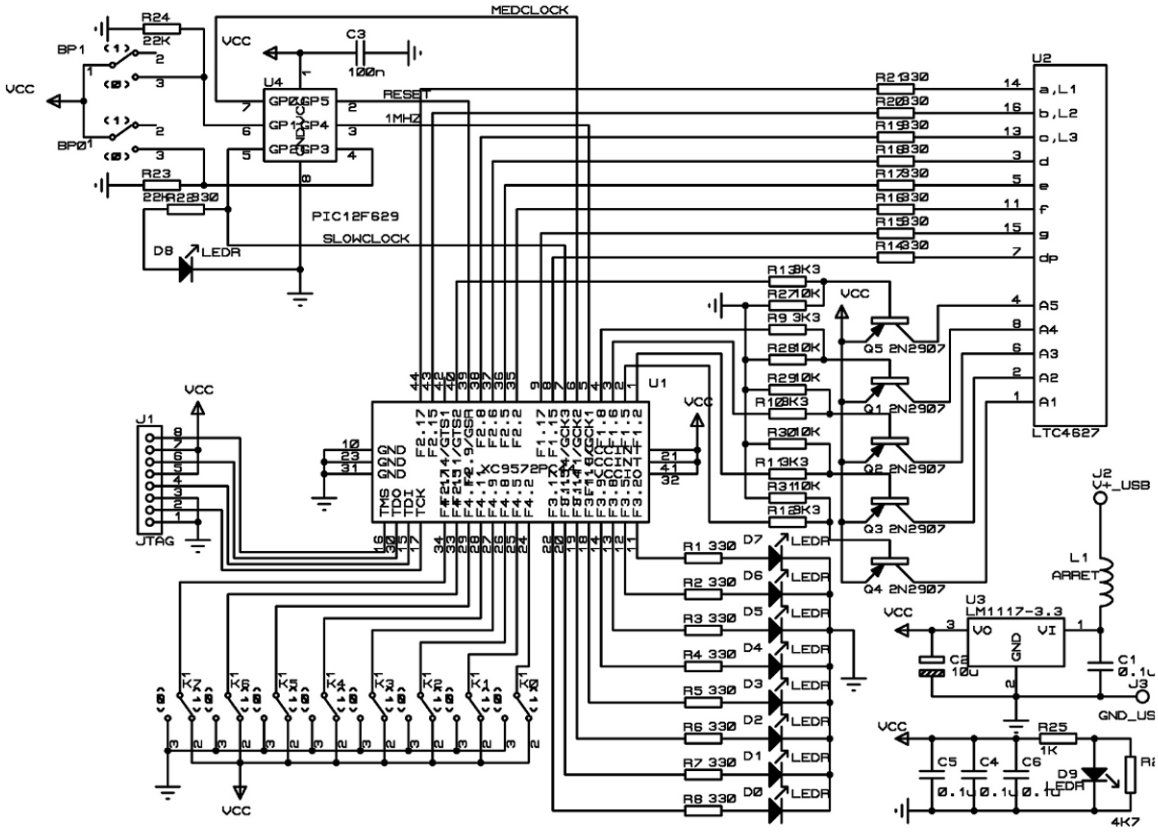
DÉPARTEMENT GÉNIE ÉLECTRIQUE & INFORMATIQUE INDUSTRIELLE — CRÉTEIL-VITRY

TP 0

Découverte de l'environnement et de la carte d'étude

1. Description de la carte d'étude

Tous les exercices de travaux pratiques seront réalisés sur le dispositif d'essai équipé d'un CPLD XC9572L, en boitier PLCC 44 broches. La maquette comprend aussi des interrupteurs et boutons-poussoirs, des diodes électroluminescentes, des afficheurs, un circuit générateur d'horloge.



1.1. Travail préparatoire

A partir du schéma précédent, vous devez répondre aux questions suivantes en **justifiant** vos réponses.
Ce travail est à faire **avant la première séance de TP**.

1. Le composant programmable (CPLD) est le boitier U1. Combien de broches possède-t-il ?
2. Quel composant est relié sur la **broche 34** ? A quoi sert-il ? La broche devra-t-elle être en entrée ou en sortie ? Dans quel sens ira le courant ?
3. A quoi sert le composant **U3** ? Que vaut alors **VCC** ?
4. A quoi sert la **LED D9** ?
5. Quel **niveau logique** faut-il appliquer sur la broche **broche 22** pour allumer la LED D0 ? En supposant que la tension de la diode est de 1.6 V lorsqu'elle est passante, quel est le courant qui traverse D0 lorsqu'elle est allumée ?

1.2. Entrées-sorties logiques

Les entrées logiques sont essentiellement réalisées par 8 interrupteurs dénommés K0 à K7.

Les sorties pourront être visualisées sur 8 diodes électroluminescentes (LED0 à LED7) ou bien 4 afficheurs 7 segments multiplexés.

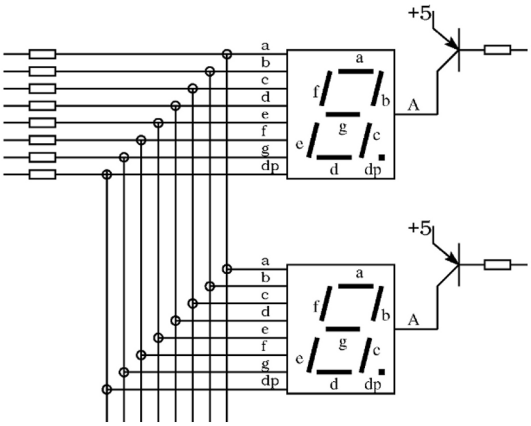
1.3. Horloges et remise à zéro

Il y a trois horloges de caractéristiques différentes produites par un circuit spécial :

- une horloge lente «CLK_SLOW», contrôlée par un bouton poussoir de même nom ; le bouton poussoir «CLK_SLOW» fonctionne ainsi : un appui bref (inférieur à une seconde) produit un front montant unique sur la ligne CLK_SLOW ; un appui long entraîne l'apparition d'un signal rectangulaire continu de période 0,5 seconde ; à la suite de quoi un nouvel appui arrête ce signal rectangulaire.
- une horloge moyenne «CLK_256», constante, de fréquence approximative 256 Hz ;
- une horloge rapide «CLK_1M», constante, de fréquence approximative 1 MHz.

Il y a également un bouton poussoir «RESET» permettant la remise à zéro des éléments séquentiels. Il produit une impulsion unique de durée approximative de 20 ms.

1.4. Afficheurs multiplexés



Le multiplexage de l'affichage permet de limiter le nombre de connexions pour ces 4 afficheurs. Mais son fonctionnement exige la mise en œuvre de composants séquentiels.

Pour activer un segment (cathode), on doit placer un 0 (zéro) sur la sortie correspondant au segment **seg**. Pour activer un afficheur, il faut également placer un 0 sur la sortie **an** désirée, (en raison de la présence d'un transistor amplificateur inverseur).

a...g, dp = cathodes des segments et du point décimal de tous les afficheurs ; an = commandes des anodes communes des afficheurs.

élément	broche	type
an(0)	p4	commande anode gauche
an(1)	p3	commande anode
an(2)	p1	commande anode
an(3)	p2	commande anode droite
an(4)	p40	commande anode points

élément	broche	type
g, seg(0)	p9	sortie
f, seg(1)	p35	sortie
e, seg(2)	p36	sortie
d, seg(3)	p37	sortie
c, seg(4)	p38	sortie
b, seg(5)	p43	sortie
a, seg(6)	p44	sortie
dp, seg(7)	p8	sortie

1.5. Les affectations des entrées et sorties

Du fait du pré-câblage de la maquette, certaines broches devront **obligatoirement** être utilisées comme entrées, et certaines autres comme sorties. Chaque broche du circuit correspond à un organe déterminé (poussoir, interrupteur, horloge, afficheur, led). Un **fichier de contraintes** correspondant au brochage du CPLD sur cette carte d'étude est fourni en annexe.

Les numéros de broches correspondent à la version en boîtier PLCC44 du circuit. Il est **impératif** que le type correspondant soit sélectionné dans l'environnement logiciel de développement : **XC9572XL**.

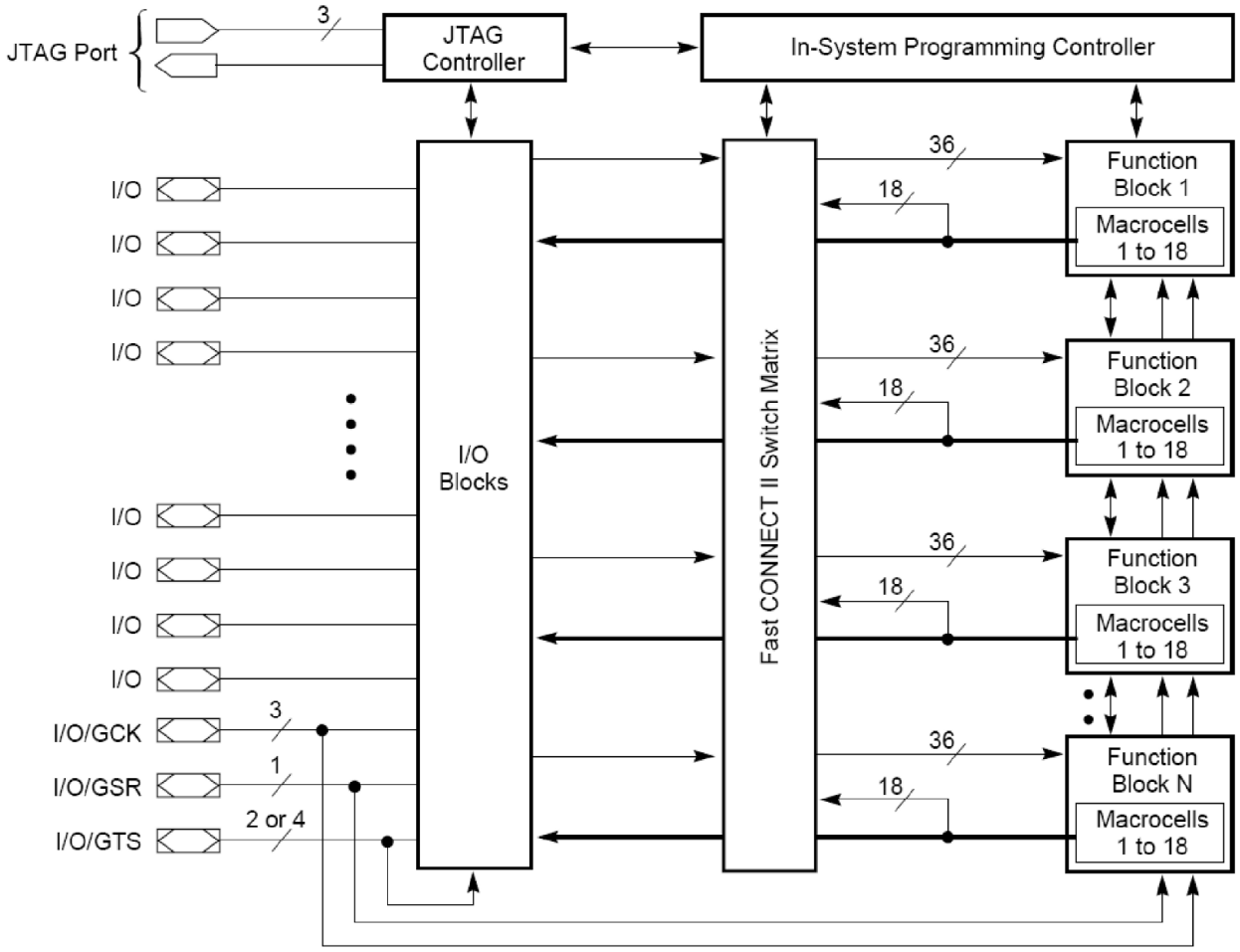
élément	broche	type	élément	broche	type
K(0)	p24	entrée	LED(0)	p22	sortie
K(1)	p25	entrée	LED(1)	p20	sortie
K(2)	p26	entrée	LED(2)	p19	sortie
K(3)	p27	entrée	LED(3)	p18	sortie
K(4)	p28	entrée	LED(4)	p14	sortie
K(5)	p29	entrée	LED(5)	p13	sortie
K(6)	p33	entrée	LED(6)	p12	sortie
K(7)	p34	entrée	LED(7)	p11	sortie

K=interrupteur ; LED=diode électroluminescente.

élément	broche	fonction principale	caractéristique
CLK_1M	p5	horloge	fréquence 1MHz
CLK_256	p6	horloge	fréquence 256Hz
CLK_SLOW	p7	horloge	pas à pas ou 2Hz
RESET	p39	remise à zéro	impulsion

2. Architecture sommaire des CPLD XC9500

L'architecture interne des CPLD XC9500 est représentée sur la figure ci-dessous.



2.1. Blocs d'entrées/sorties

Les blocs d'entrées/sorties réalisent l'interface entre la logique interne et les broches accessibles à l'utilisateur. Chaque *IOB* comprend un buffer d'entrée, un amplificateur de sortie, un multiplexeur de sélection et une commande de mise à la masse.

En configurant des broches d'entrée/sortie bien choisies comme des masses additionnelles, les signaux parasites induits par les commutations des différentes sorties peuvent être réduits.

2.2. Blocs fonctionnels - FB

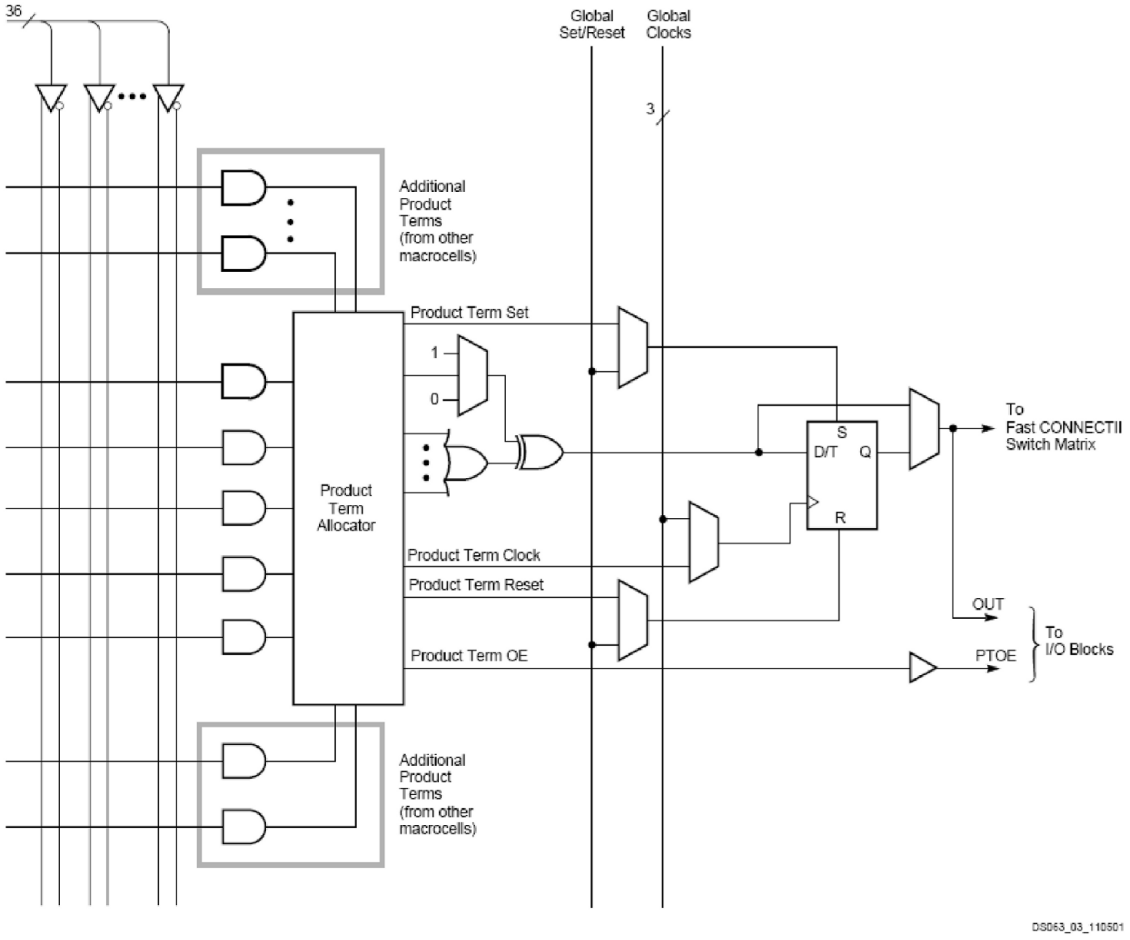
Les CPLD de chez Xilinx se décomposent en plusieurs **blocs fonctionnels** (*Function Block*, FB) et en blocs d'entrées/sorties (*I/O Blocks*, IOB) totalement interconnectés.

Chaque FB se comporte comme un circuit logique programmable à **36 entrées** et **18 macrocellules** (macrocells) engendrant 18 sorties ainsi que 18 termes de validation (PTOE) (voir figure ci-après).

A l'intérieur d'un bloc fonctionnel, la logique est réalisé par une **somme de produits**. Les 36 entrées sont combinées à leur complément et forment les 72 entrées d'un réseau de portes et qui engendrent 90 termes de produit. Chaque terme de produit peut être redirigé sur n'importe quelle macrocellule par l'**allocateur de terme produit** (*Product Term Allocator*, PTA).

2.3. Macrocellules

Chaque macrocellule peut être individuellement configurée comme une fonction combinatoire ou séquentielle (un registre).



Cinq termes produits directs provenant du réseau de portes *ET* sont disponibles comme entrées des portes *OU* et *OU-EXC* pour les fonctions combinatoires, ou pour produire des contrôles (*Clock*, *Reset*, *OE)*. C'est le PTA qui sélectionne comment les 5 termes directs sont utilisés.

Le registre (synchrone) de la macrocellule peut être configuré soit comme un type D, soit comme un type T, ou bien désactivé pour les fonctions combinatoires. Chaque registre accepte des commandes *Set* et *Reset* asynchrones.

Durant l'établissement de l'alimentation, tous les registres sont initialisés dans un état défini par l'utilisateur (0 si non spécifié).

3. Premiers programmes en VHDL

3.1. Réalisation d'une porte logique

1. Réaliser une **porte logique «ET» à 2 entrées** (interrupteurs). La visualisation de l'état de sortie se fera sur une LED. Quelles sont les ressources utilisées pour réaliser cette fonction ?
2. Réaliser une **porte logique «OU» à 4 entrées** (interrupteurs). La visualisation se fera là encore sur une LED. Le nombre de ressources utilisées a-t-il changé ?

3.2. Décodeur 2 vers 4

Dans un soucis de gain de place, la plupart des systèmes numériques n'ont qu'un seul bus de données où doit transiter l'ensemble des informations entre les différents organes de ce système (microprocesseur, mémoires, boitiers d'entrées/-sorties...). Il est alors indispensable d'adresser spécifiquement les circuits devant être reliés entre eux. On parle alors d'**adressage**. Pour cela, chaque circuit est équipé d'une entrée de sélection ($\overline{CE}$), actives au niveau «bas».

On souhaite réaliser un décodeur d'adressage pour un système comportant deux lignes d'adresse A0 et A1 et quatre dispositifs an0 à an3 en sortie. La sortie an0 est active pour un code d'entrée valant 0 (décimal), an1 pour 1, an2 pour 2 et an3 pour 3.

1. Faire le **schéma synoptique** du système.
2. Donner la **table de vérité** de chacune des sorties an0 à an3 en fonction des entrées A0 et A1.
3. Spécifier alors ce circuit décodeur au moyen d'**équations logiques** habituelles.
4. Reformuler ensuite le problème en utilisant des **affectations conditionnelles**.
5. Puis en utilisant une **affectation sélective**.

GEII - **SIN1**

IUT CRÉTEIL-VITRY — DÉPARTEMENT GÉNIE ÉLECTRIQUE & INFORMATIQUE INDUSTRIELLE

TP 1

Systèmes Combinatoires Simples

Objectifs

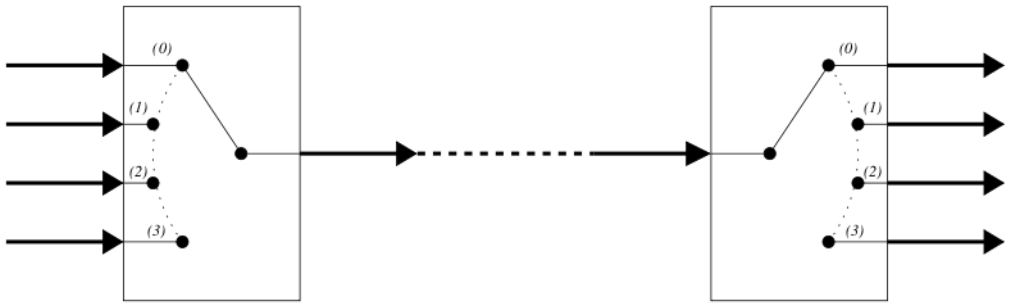
- Découvrir la carte d'étude basée sur un *CPLD XC9575XL*
- Découvrir l'environnement de développement VHDL de *Xilinx* (ISE-WebPack)
- Réaliser des fonctions combinatoires simples à l'aide d'un composant logique programmable

1. Multiplexeur

1. Réaliser un multiplexeur de 4 lignes (4 interrupteurs) vers 1 ligne (1 LED). Utiliser 2 interrupteurs pour réaliser la sélection des voies.
2. Rajouter une fonction de validation générale (OE).

2. Multiplexeur et démultiplexeur

Réaliser un **multiplexeur de 4 lignes** (4 interrupteurs) vers 1 ligne puis un **démultiplexeur 1 ligne vers 4 lignes** (4 LED). Utiliser 2 interrupteurs pour réaliser la sélection des voies du multiplexeur (AS) et 2 interrupteurs pour la sélection de la destination en sortie du démultiplexeur (AD).



Multiplexeur Démultiplexeur

3. Additionneur à 4 bits

On souhaite additionner deux nombres *a* et *b* codés sur 4 bits et retourner le résultat *r* sur 4 bits.

1. Faire le **schéma synoptique** du système.
2. Réaliser l'**additionneur 4 bits**, les deux mots à additionner seront formés par les interrupteurs et les sorties seront visualisées sur des diodes.
3. Réaliser l'additionneur complet 4 bits (avec la retenue).

GEII - **SIN1**

DÉPARTEMENT GÉNIE ÉLECTRIQUE & INFORMATIQUE INDUSTRIELLE — CRÉTEIL-VITRY

TP 2

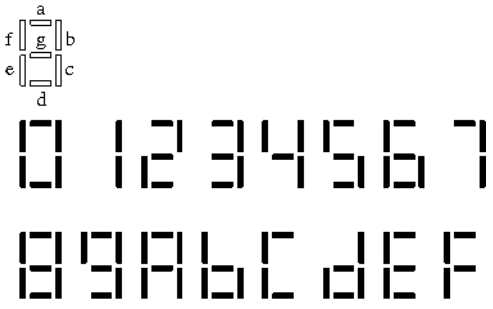
Logique Combinatoire (suite)

Objectifs

- Réaliser des fonctions combinatoires un peu moins simples à l'aide d'un composant logique programmable

1. Décodeur à sept segments hexadécimal

1. Faire le **schéma synoptique** du système.
2. En utilisant les interrupteurs $K_0..K_3$ comme entrées et les 7 segments d'un afficheur comme sortie, réaliser un **décodeur binaire/hexadécimal** selon le code donné par la figure ci-dessous.
3. Limiter la plage codée à 0-9 (décodeur décimal), l'afficheur étant éteint ailleurs.



Attention : les afficheurs sont multiplexés et actifs à l'**état bas** ! !

2. Additionneur à 4 bits avec afficheur

A partir des deux précédents exercices, il est possible de réaliser un additionneur complet avec affichage du résultat (la retenue sera affichée sur une LED).

Réaliser l'**additionneur complet 4 bits à affichage** à l'aide d'un seul fichier *.vhd. Un seul afficheur sera utilisé et la retenue sera affichée sur une LED annexe.

3. Comparateur binaire

Un comparateur binaire est un dispositif qui accepte deux mots binaires de 4 bits sous forme parallèle A et B et qui possède des sorties binaires indiquant la relation d'ordre entre ces mots : égalité, supériorité ou infériorité.

1. Combien de signaux de sortie sont-ils nécessaires ?
2. Faire le **schéma synoptique** du système.
3. Coder un comparateur binaire à mots de 4 bits. Les sorties seront visualisées sur des diodes.

GEII - **SIN1**

DÉPARTEMENT GÉNIE ÉLECTRIQUE & INFORMATIQUE INDUSTRIELLE — CRÉTEIL-VITRY

TP 3

Logique Séquentielle Elémentaire

Objectifs

- Réaliser des systèmes séquentiels simples
- Réaliser un premier système séquentiel synchrone

1. Compteur synchrone 4 bits

1. Faire le **schéma synoptique** du système.
2. Quelle est la dynamique de ce compteur ?
3. Réaliser un compteur sur 4 bits par description comportementale.
4. Rajouter une entrée de remise à zéro asynchrone, puis synchrone.

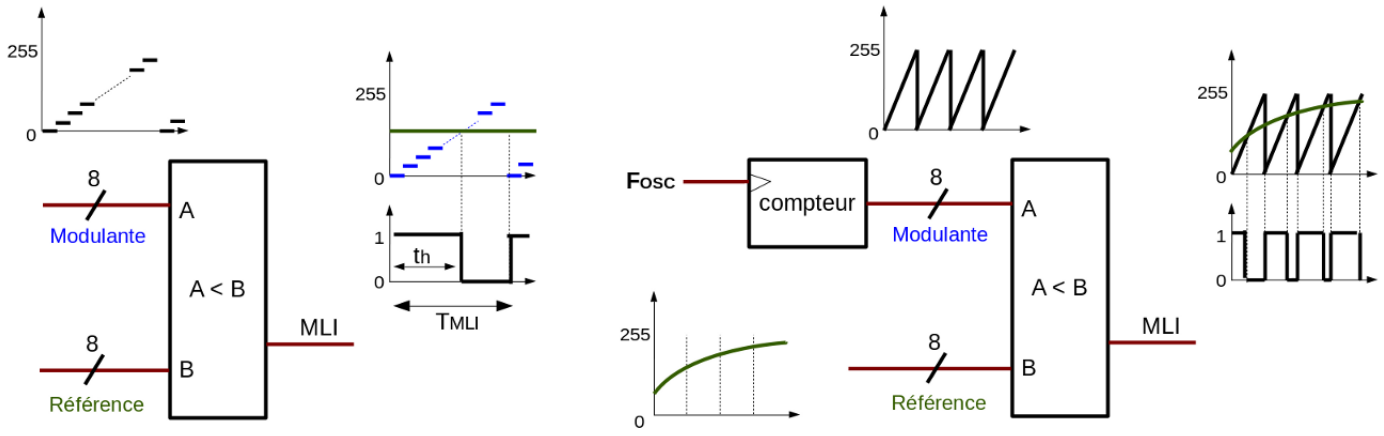
Les sorties seront d'abord visualisées sur des LED puis sur un afficheur 7 segments (voir TP précédent).

2. Compteur décimal

1. Décrire un compteur synchrone avec remise à zéro asynchrone comptant de 0 à 9.
2. Combien de bascules sont nécessaires à la réalisation de ce compteur ?
3. Rajouter une entrée et modifier le code source afin de transformer ce compteur en compteur/décompteur.

3. Gradateur de LED

La luminosité d'un éclairage (dans notre cas, une LED) peut être réglée par l'application d'un signal rectangulaire périodique de fréquence assez élevée et de rapport cyclique non constant. On parle alors de gradateur.



Le rapport cyclique d'un signal rectangulaire périodique est défini comme le rapport entre le temps pendant lequel le signal est à l'état haut et la période. On exprime souvent ce rapport en pourcentage. Ainsi :

- la LED est éteinte pour une rapport cyclique de 0% ;
- la luminosité est maximale pour une rapport cyclique de 100%.

Cette application peut être simplement réalisée à l'aide d'un compteur à 4 bits (synchrone) et d'un comparateur arithmétique à 4 bits. Les interrupteurs déterminent le rapport cyclique. L'horloge à choisir pour le compteur est CLK_1M.

1. Faire le **schéma synoptique** du système complet.
2. Réaliser le fichier source permettant de décrire ce gradateur de LED.

GEII - **SIN1**

DÉPARTEMENT GÉNIE ÉLECTRIQUE & INFORMATIQUE INDUSTRIELLE — CRÉTEIL-VITRY

TP 4

Logique Séquentielle Synchrone

Objectifs

- Réaliser des systèmes séquentiels synchrones
- Différencier la synthèse structurelle et la synthèse comportementale

1. Registre à décalage

1. Faire le **schéma synoptique** du système.
2. Réaliser un **registre à décalage simple sur 6 bits**, avec une entrée série.
3. Le préchargement est maintenant en parallèle et réalisé à l'aide de 6 interrupteurs. Modifier le code source précédent. L'ordre de préchargement sera donné par un interrupteur indépendant.
4. Modifier le code source pour obtenir un **registre à recirculation**.

Les sorties seront visualisées sur des LED et l'horloge sera fournie par le générateur manuel CLK_SLOW.

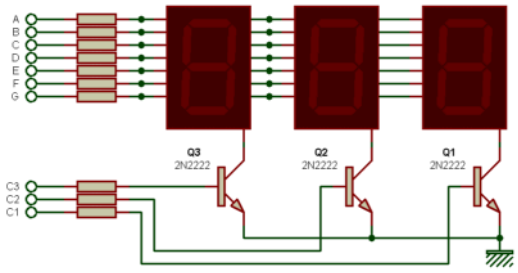
2. Affichage multiplexé

On souhaite à présent réaliser un afficheur à 4 chiffres (tel que ceux utilisés dans les horloges).

Les données à afficher doivent s'étendre sur 4x4 = 16 bits. Chaque groupe de 4 bits doit être dirigé, à tour de rôle, vers l'afficheur qui lui correspond.

Les données à envoyer sur les afficheurs transitent par un **même bus** (7 fils dans le cas d'un afficheur 7 segments - de `a` à `g`) pour l'ensemble de ces afficheurs. Ainsi le chiffre à afficher est envoyé sur tous les afficheurs en même temps. Sans une gestion particulière de ces afficheurs, il n'est pas possible d'écrire des nombres.

C'est pourquoi, en parallèle ce ces entrées de données, il existe une **entrée de validation** pour chacun des afficheurs (`an0` à `an3`). Ces entrées sont souvent commandés via des transistors pour permettre un gain de courant. Le schéma ci-dessous donne le câblage d'un tel système (3 afficheurs 7 segments dans ce cas-là).



Compte tenu du multiplexage de l'affichage, un *séquenceur* est nécessaire pour contrôler les opérations. La fréquence de balayage des afficheurs doit être suffisante pour que l'oeil humain ne perçoivent pas le clignotement. L'horloge CLK_256 est suffisante.

1. Ecrire le **séquenceur** sous forme d'un processus unique.
2. Ajouter le **décodeur 7 segments** en dehors de ce processus (voir TP précédents).
3. Pour réaliser la source de données en entrée, réaliser un **compteur binaire** commandé par CLK_SLOW (16 bits).

GEII - **SIN1**

DÉPARTEMENT GÉNIE ÉLECTRIQUE & INFORMATIQUE INDUSTRIELLE — CRÉTEIL-VITRY

TP 5

Premières machines à état

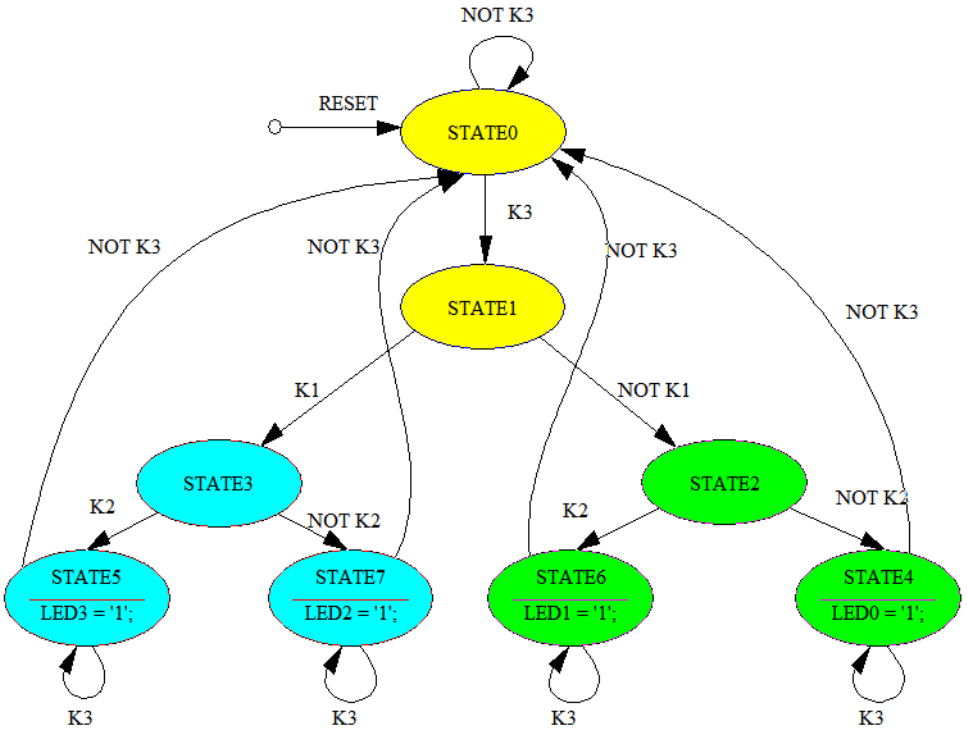
Objectifs

- Décrire une machine à état en VHDL

1. Première machine à états

Réaliser la **machine à états** définie par le diagramme d'états ci-dessous, avec les informations suivantes :

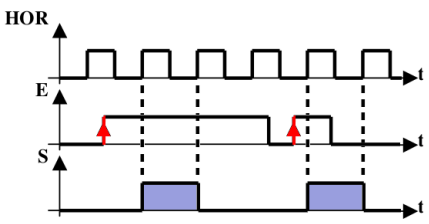
- l'horloge est CLK_SLOW, le bouton RESET (p39) doit être utilisé ;
- l'état de la machine est mémorisé par un signal *etat* (STD_LOGIC_VECTOR) ;
- la numérotation des états est conforme au diagramme ;
- l'état est visualisé sur l'un des afficheurs 7 segments ;
- les LEDs et les interrupteurs suivent la nomenclature du diagramme.



Vérifier le bon fonctionnement de cette machine avec une horloge pas à pas.

2. Détecteur de front montant

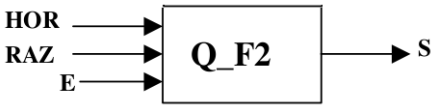
On considère un signal E (représenté par l'interrupteur K0) susceptible de changer d'état n'importe quand. On désire construire une machine d'état synchronisée par une horloge (CLK_SLOW) qui fournisse une (et une seule) impulsion sur sa sortie de détection S après apparition d'un front montant sur E.



1. Faire le **diagramme d'état** de la machine décrite ci-dessus.
2. Décrire en VHDL la machine de Moore correspondante.

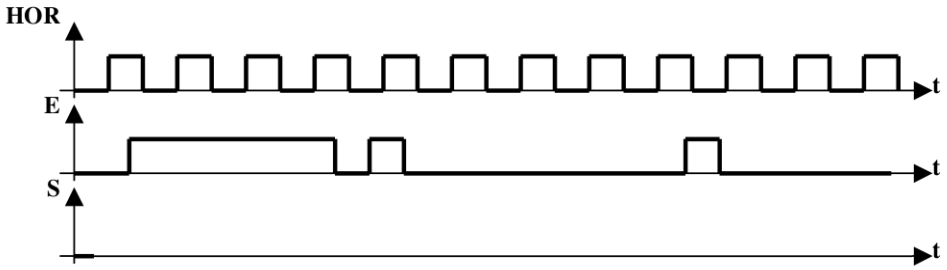
3. Séquenceur

On désire réaliser une fonction dont la sortie *S* recopie l'état logique présent sur son entrée *E* si celle-ci est restée stable après **2 coups d'horloge successifs**.



L'entrée *RAZ* a été rajoutée afin que l'état initial du système soit défini. De façon plus générale, un système séquentiel doit **toujours être initialisé**, de plus cette initialisation doit être asynchrone.

1. Compléter le **chronogramme** ci-dessous en fonction du comportement décrit précédemment.
2. Faire le **diagramme d'états** du système.
3. Décrire cette machine à état en VHDL.



GEII - **SIN1**

TP 6

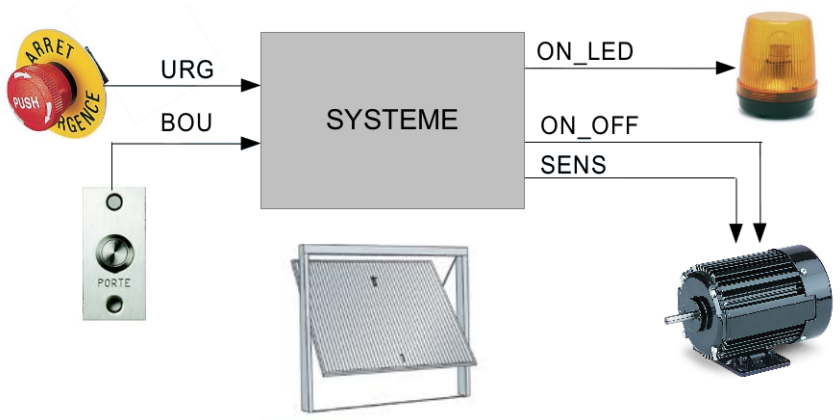
Systèmes numériques

Objectifs

- Concevoir des systèmes numériques simples

1. Automatisme de porte de garage

Une porte de garage motorisée comporte un moteur électrique réversible à 2 sens de marche correspondant à la montée ou à la descente.



La commande est assurée par un bouton poussoir unique, *BOU* qui commande la marche : ouverture lorsque la porte est fermée, fermeture lorsque la porte est ouverte.

Deux capteurs de fin de course haut et bas, FH et FB, permettent d'arrêter automatiquement le mécanisme.

Pour représenter l'état du système, on utilisera un vecteur CMD à 2 composantes :

- CMD(0) représente l'activité : marche si CMD(0) = '1', arrêt sinon ;
- CMD(1) représente le sens de marche : montée si CMD(1) = '1', descente sinon.

Lorsqu'une commande est en cours d'exécution, le bouton *BOU* est inopérant.

L'horloge du système est CLK_1M (p5).

Le bouton *BOU* est CLK_SLOW (p7).

La montée ou la descente seront visualisées par 2 diodes $MONTEE$ et $DESCENTE$.

1. Etude du **système sans sécurité** :
   (a) Faire le **diagramme d'état** (3 états A, M et D sont suffisants).
   (b) Décrire ce système en VHDL.

2. Etude d'une **sécurité simple** :
   Un détecteur d'urgence, *URG*, permet d'arrêter immédiatement le fonctionnement, quel que soit l'état du système, à la position où il se trouve ; dans ce cas, après élimination du défaut ($URG =' 0'$), un appui sur *BOU* permet de reprendre le mouvement dans le sens précédent.
   (a) Modifier le **diagramme d'état** précédent.
   (b) Décrire ce nouveau système en prenant un interrupteur (p24) pour *URG*.

2. Distributeur de boisson

On désire simuler une machine d'état devant équiper un appareil distributeur de boisson. Au départ cette machine est dans un état de repos qu'elle indique en allumant une LED $D1$.

Le prix d'une boisson (une seule sorte) est de 1,50 euro. Cette somme peut être fournie au moyen de 2 sortes de pièces : 0,5 euro ou 1 euro.

La détection d'une pièce introduite dans la machine est simulée par 2 interrupteurs (un pour chaque sorte de pièce). La machine ne rend pas la monnaie si l'utilisateur dépasse la somme exigée. Dès que la somme voulue a été atteinte ou dépassée, la LED $D1$ s'allume pour indiquer la libération d'une bouteille et le retour à l'état de repos.

1. Tracer le diagramme d'état puis le minimiser.
2. Créer cette machine.
3. Reprendre le problème pour une boisson coûtant 2 euros.

Pour réaliser l'horloge indispensable à la machine d'état, on utilisera le générateur manuel (CLK_SLOW).