Synthèse et mise en œuvre des systèmes

Bureau d'étude BE VHDL
M2 SME
module EIEAS3G1

Objet Technique étudié
(ou Système embarqué étudié) :
Pilote de barre franche pour voiliers

DOCUMENTATIONS

Pour plus d'infos :

Site web Thierry PERISSE (manuel de TP, cours VHDL, manuels logiciels, …)

http://thierryperisse.free.fr/index.php/be-vhdl/

Cours Eric PERONNIN

Composants logiques reconfigurables

Manuel utilisation Quartus II

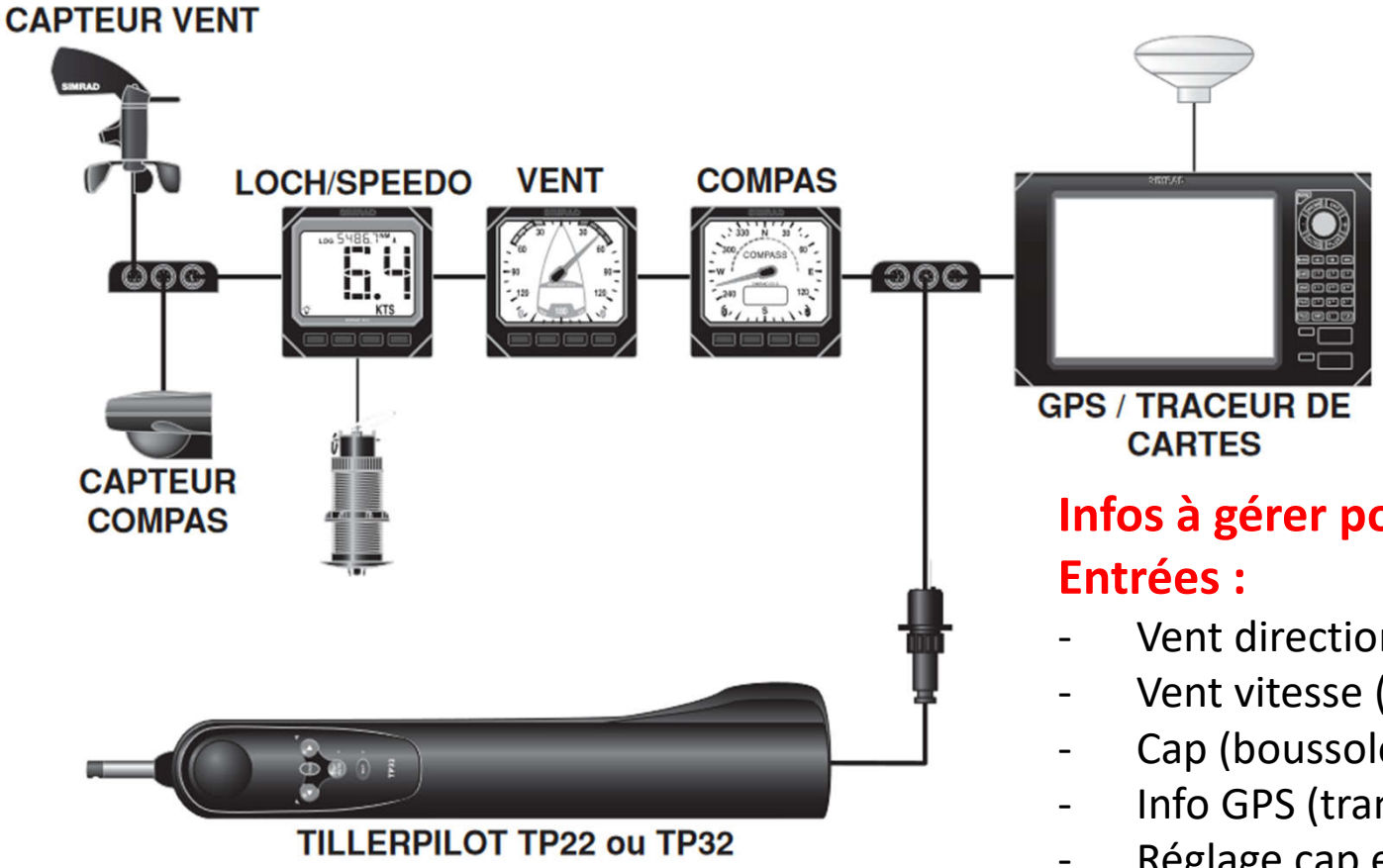
https://onedrive.live.com/download?cid=ADC7561192B36384&resid=ADC7561192B36384%213742&authkey=AKZEGnGR2R3JX_4&em=2

Guide d'utilisation MODELSIM

https://chief-violin-c20.notion.site/Guide-d-utilisation-ModelSim-f9f3175063f44eb699df340dafb405e1

OT : Pilote de barre franche

Exemple d'un Pilote de barre franche professionnel Tillerpilots de la société Simrad Ltd TP10, TP22, TP32. Manuel utilisation (en français)



Infos à gérer pour une navigation précise :

Entrées :

- Vent direction (girouette)
- Vent vitesse (anémomètre)
- Cap (boussole)
- Info GPS (trame NMEA)
- Réglage cap et modes de fcts 5 BPs
- Vitesse du bateau

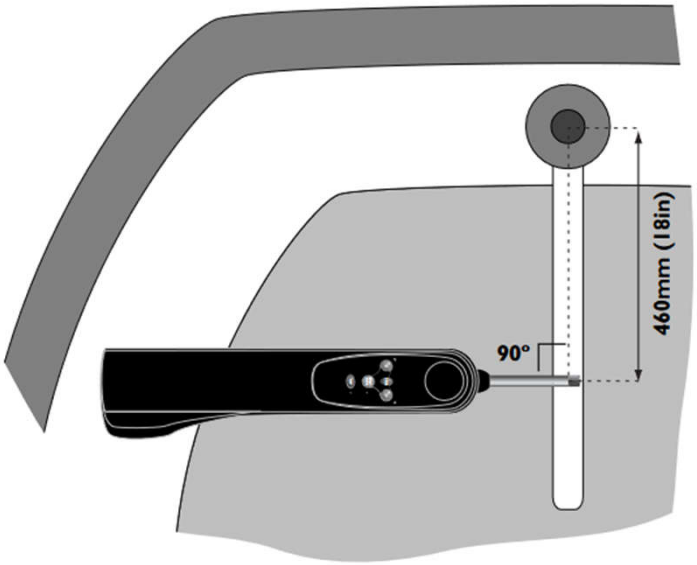
Sorties :

- Cmde barre (entrée et sortie Vérin)
- Cap (angle de barre) et Modes fcts (Leds et buzzer)

OT : Pilote de barre franche

Exemple du Tillerpilots de la société Simrad Ltd TP10, TP22, TP32.

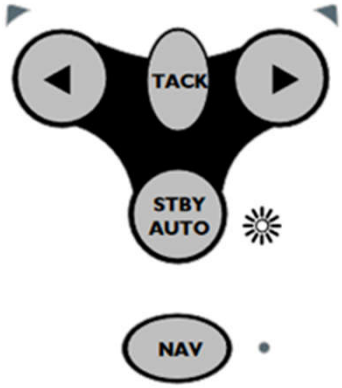
Mise en place du Tillerpilot :
L'angle entre le Tillerpilot et la barre franche doit être exactement de 90°



Les Tillerpilots ont été conçus pour allier le niveau le plus avancé de la technologie et des caractéristiques des pilotes automatiques à un mode opératoire simple et complet commandé par un clavier ergonomique à cinq touches

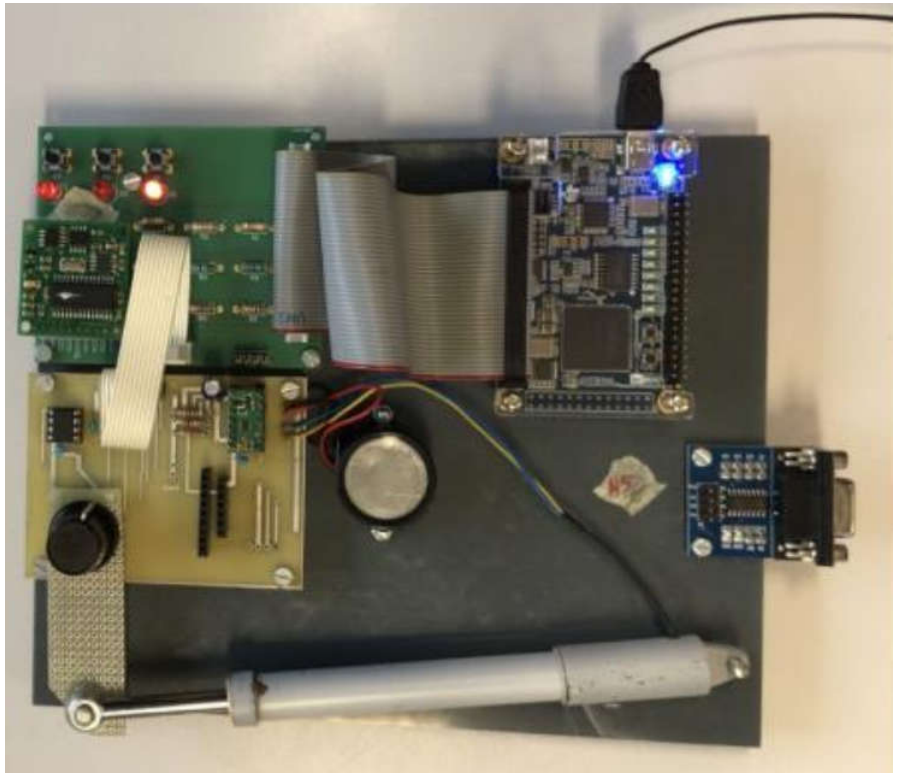
Il est possible d'effectuer des réglages de cap précis et d'utiliser toutes les fonctions de navigation
Plusieurs modes de fonctionnements : Veille, Pilote automatique, … (STBY/AUTO), Mode NAV : le Tillerpilot barre sur un cap précis (GPS, NMEA)

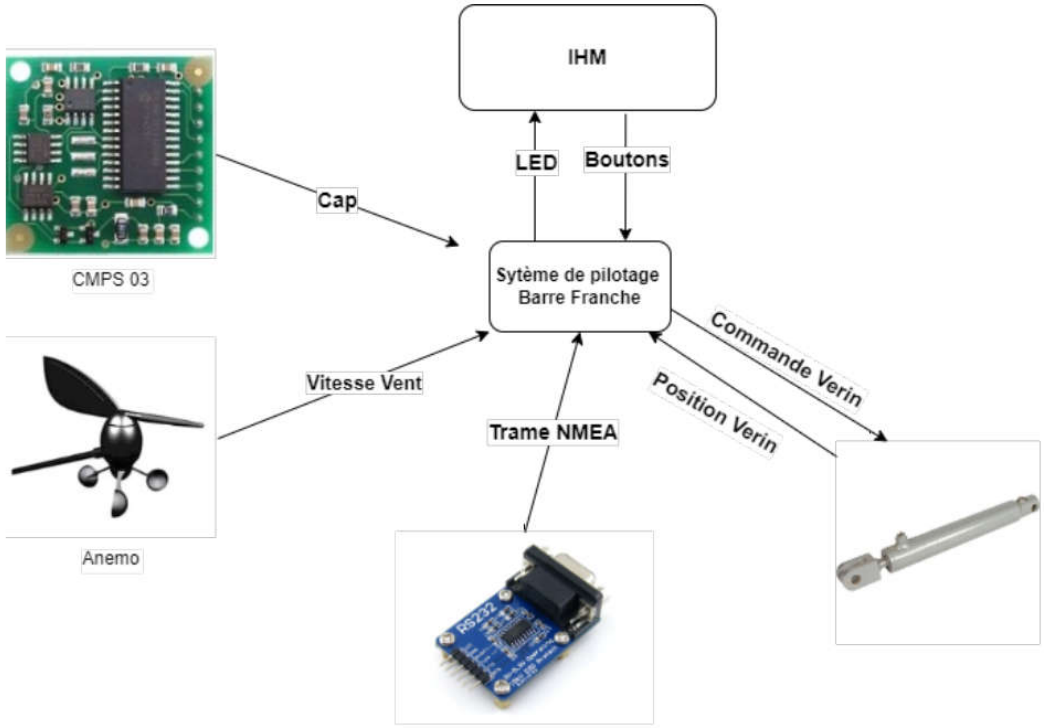
Réglages de CAP (1° ou 10°) (Babord ou Tribord) (appui court ou prolongé) (bip unique ou double bip) (éclat led (B ou T) ou clignotement led (B ou T)



OT : Pilote de barre franche

Présentation du système embarqué étudié que l'on va essayé de reproduire à l'aide de capteurs (anémomètre, girouette, boussole) et d'une maquette (fait maison) ou l'on retrouvera un vérin piloté par un PWM via un pont en H, l'angle de barre sera donné par un CAN via un potentiomètre, les informations NMEA seront données via le port série RS232.



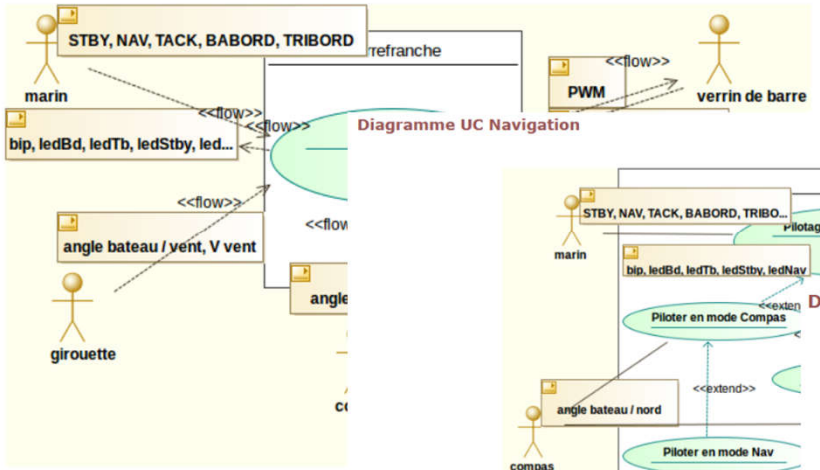


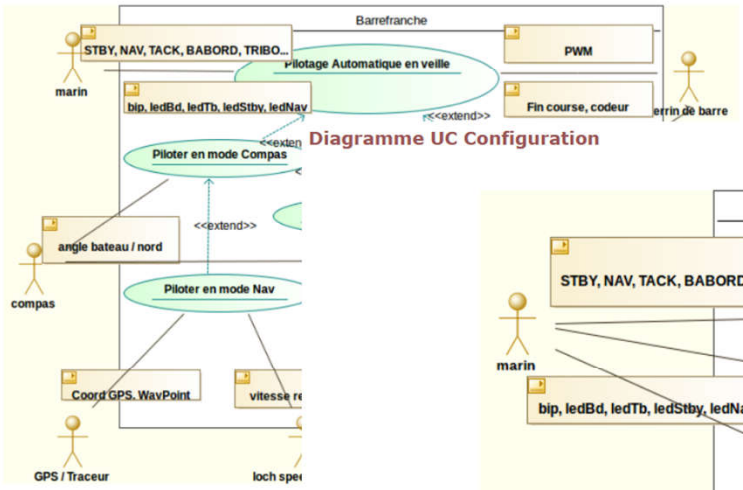
Modélisation UML de la barre franche

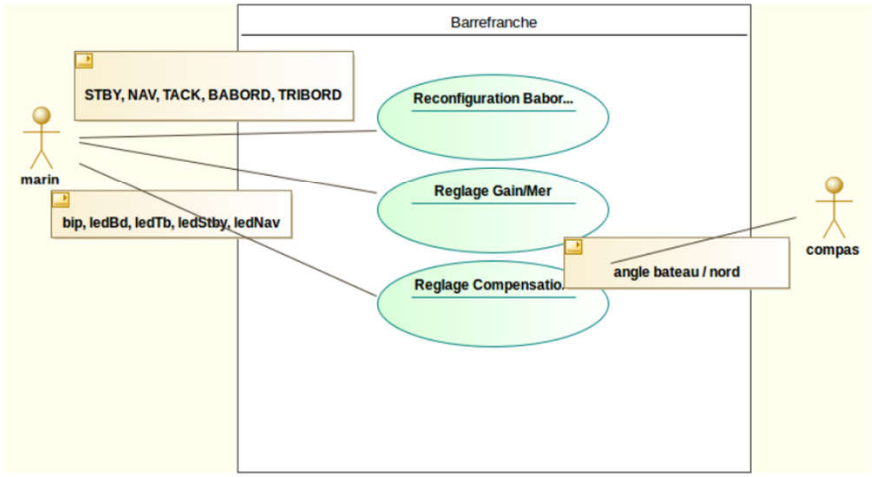
La modélisation UML est disponible sur le lien suivant :
https://homepages.laas.fr/berthou/UPS/M1ISME/UML2/WebModelPublisher/PiloteBarreFranche_v1_2/0.html

Diagrammes de contexte / de navigation / de configuration

Diagramme de contexte

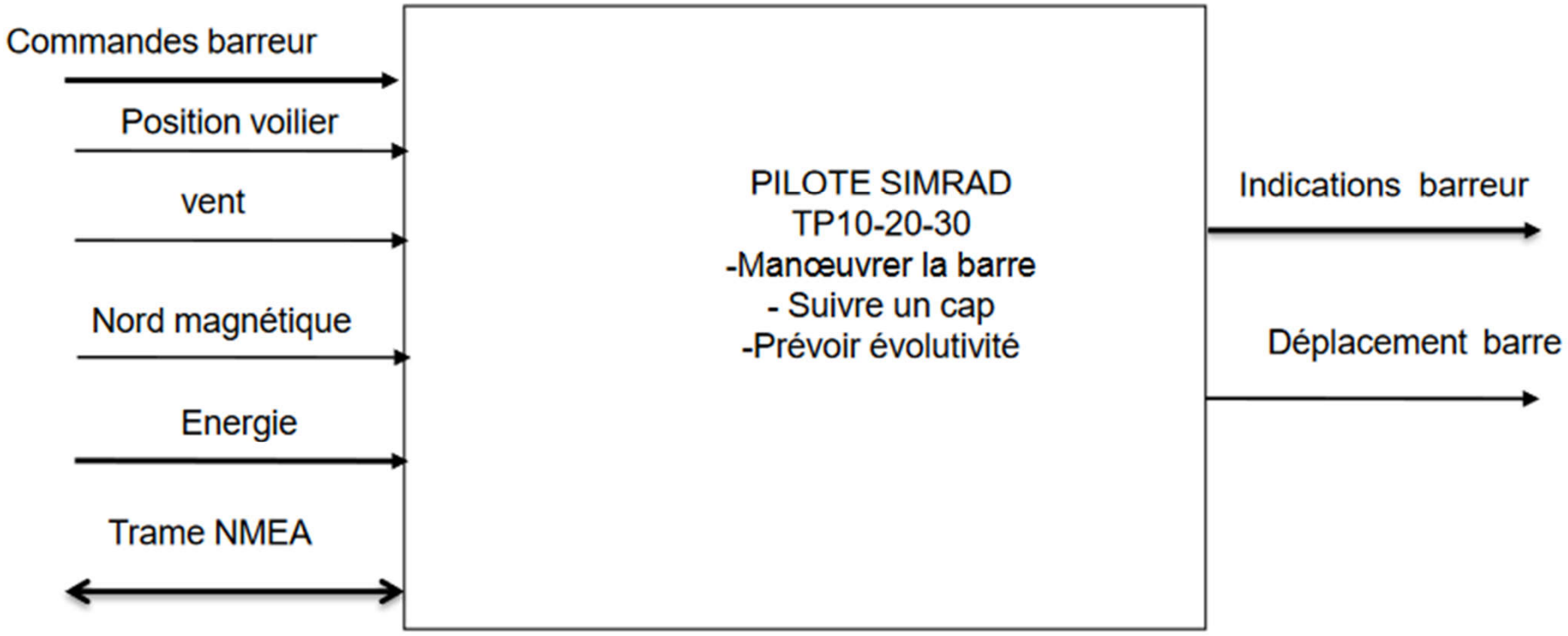






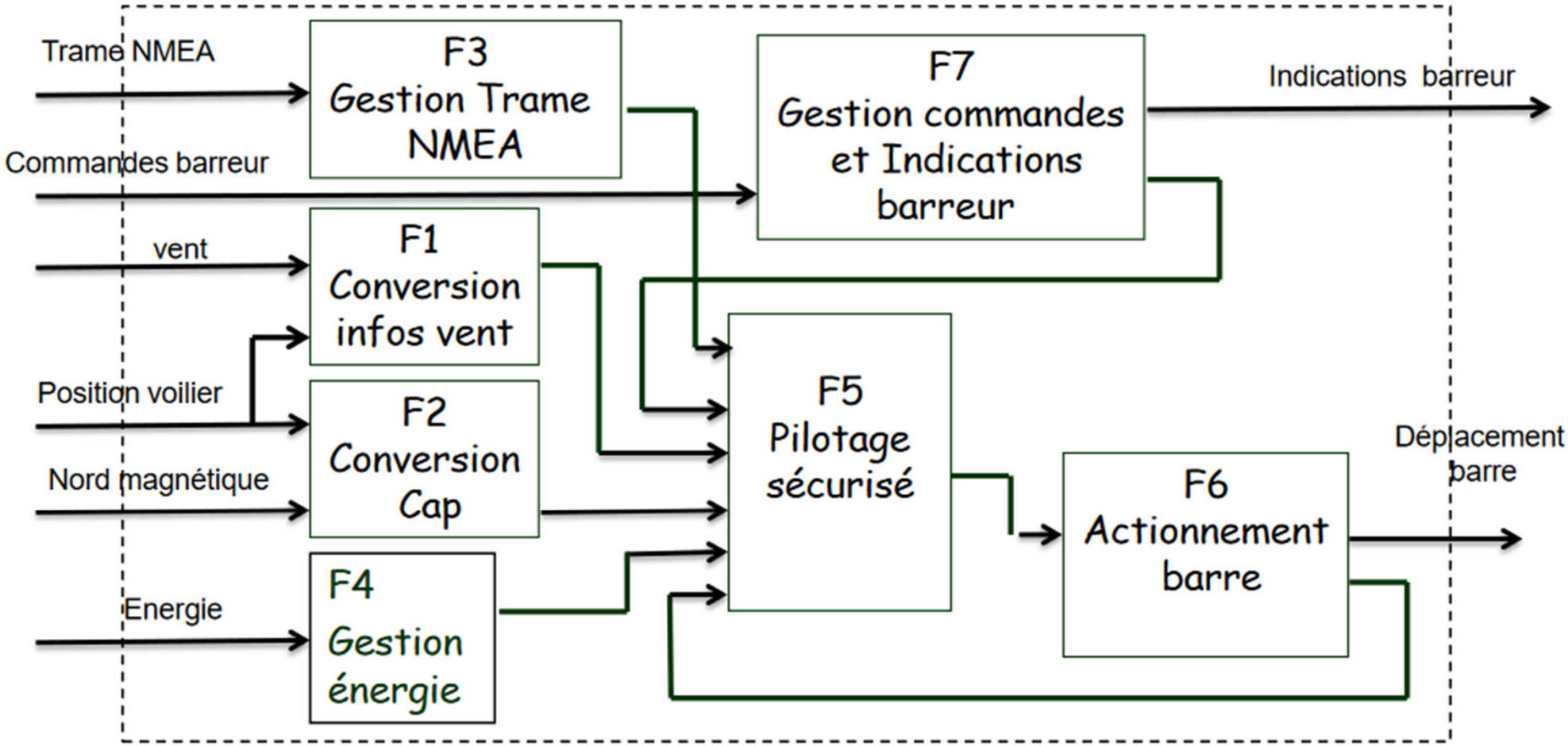
Exploration architecturale de l'OT

Boite noire : interaction avec l'extérieur (vent, marin, vérin, BPs, …)
Le travail consiste à décomposer l'OT en fonctions principales identifiées.



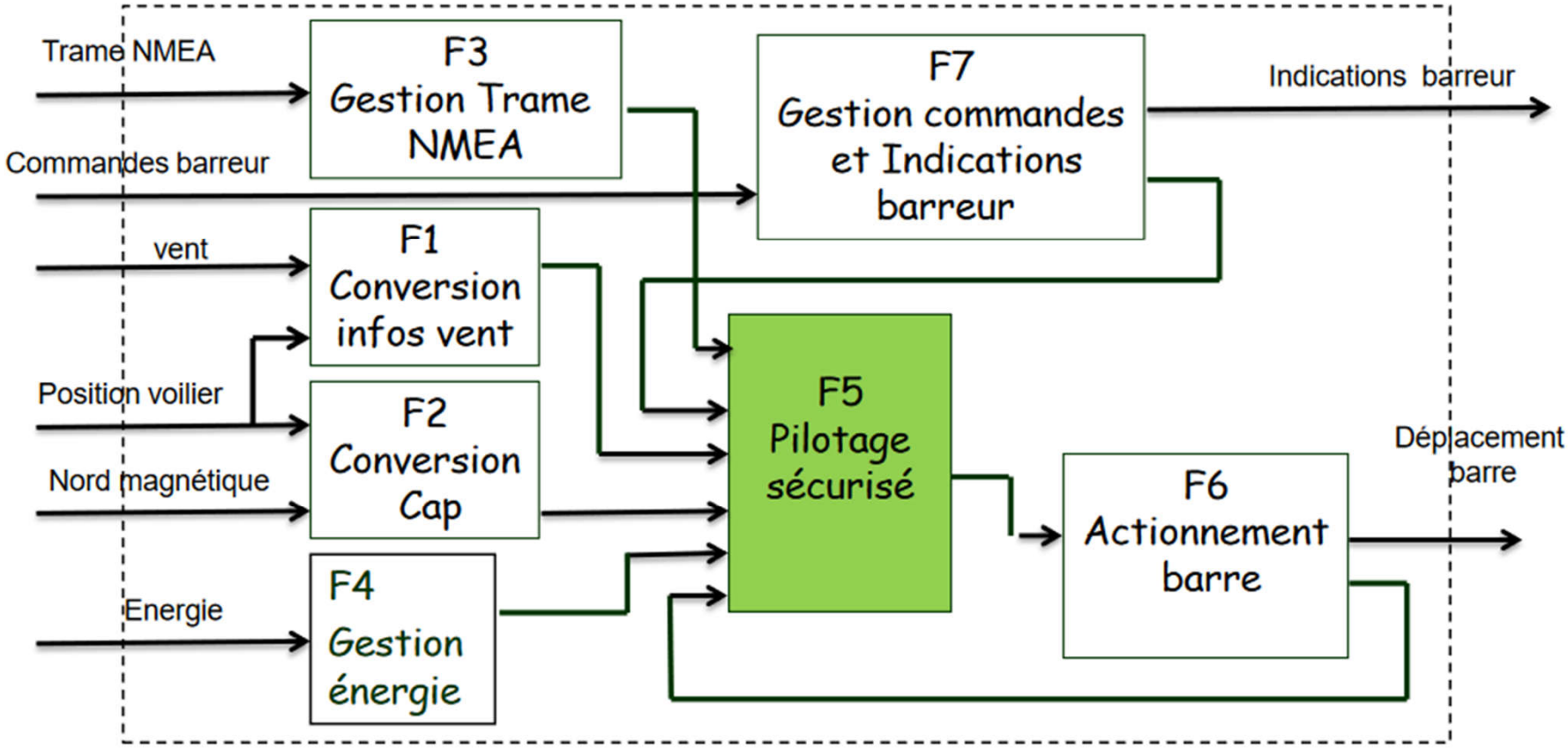
Exploration architecturale de l'OT

Décomposer (si nécessaire) les fonctions principales identifiées (F1 à F7) en fonctions secondaires plus simples pour lesquelles des solutions technologiques/physiques existent ou doivent être développées.



Exploration architecturale de l'OT

⇒F5: Pilotage
Fonctions à implémenter:
- gestion des modes de fonctionnement
- acquisition des paramètres de navigation
- pilotage automatique
- contrôle de la barre



Exploration architecturale de l'OT

⇒F5: Pilotage
Fonctions à implémenter:
- gestion des modes de fonctionnement
- acquisition des paramètres de navigation
- pilotage automatique
- contrôle de la barre

Type	Avantages	Inconvénients
Logique câblée (discret)	aucun	Quasi-obsolète
ASIC	Coût série si> 10^6	Temps de dev., coût proto Pas évolutif
FPGA	Temps de dev., coût proto	Coût série> µC
Logiciel	Temps de dev., coût proto	Performances de traitement limitées
Matériel/logiciel	Temps de dev., coût proto	minimes

Exploration architecturale de l'OT

⇒F5: Pilotage Solution ASIC:

1) Full Custom.

Avantages:

- faible prix unitaire en grande série
- fiabilité éprouvée
- répond exactement à la demande
- faible consommation

Désavantages:

- source potentielle de défauts de conception
- temps de développement
- modifications ultérieures difficiles

1) Prédiffusé (mer de portes ou de cellules)

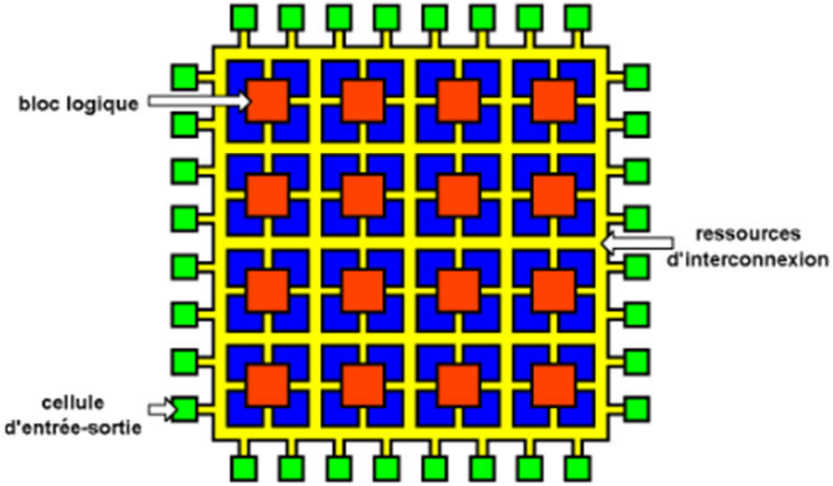
- Quelques améliorations/full custom (coût conception notamment)

Exploration architecturale de l'OT

⇒F5: Pilotage Solution FPGA:

1) Matériel pur (à base de logique câblée)
   - conception longue, un peu difficile à faire évoluer, coût série, …
   - mais nombreuses bibliothèques de composants virtuels (IP)

2) Matériel/logiciel (SOPC)
   - solution intéressante: on ne développe en hardware que le minimum (sur mesure)
   - temps de développement convenable
   - coût série mieux que ASIC mais moins bien que µC ou µP.



Exploration architecturale de l'OT

⇒F5: Pilotage

Solution logicielle:

Microprocesseur + mémoire + port E/S
- conception souple (que du soft)
- produit standard
- Mais plusieurs tâches à gérer en pseudo //: pb de temps d'exécution !!!
- complexité de la gestion des tâches (=> moniteur multitâches)
- Pb de robustesse du logiciel

Solution logicielle/matérielle:

µC: Microprocesseur + mémoire + périphériques (Timers, CAN, CNA, Ports, …)
- conception souple (que du soft)
- les contraintes fortes sont assumées par les périphériques matériels
- le logiciel gère les interactions et les ressources
- coût réduit car produit standard
- périphériques pas forcément bien adaptés aux besoins

Notion de co-design sur FPGA

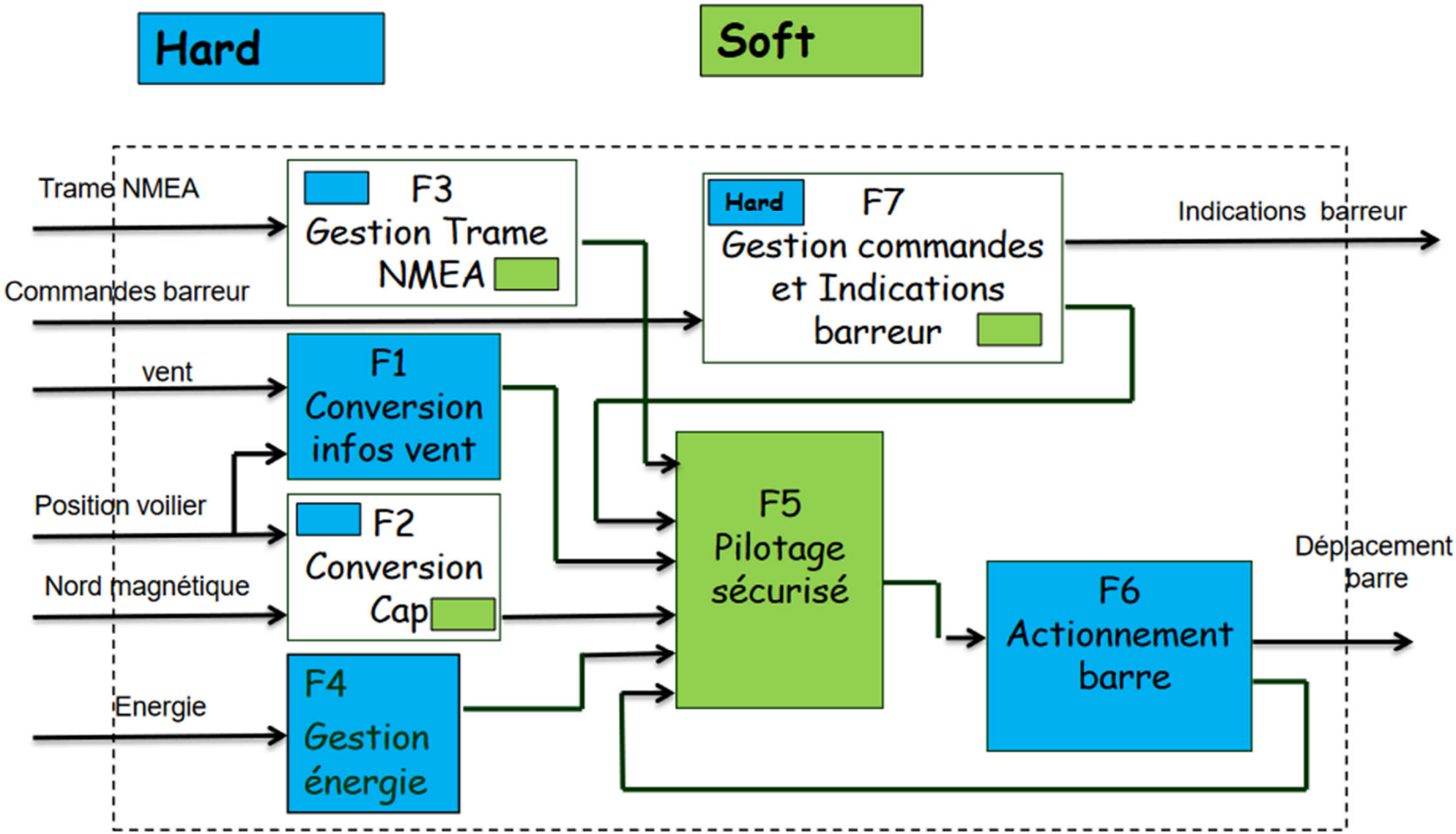
Solution préconisée pour la réalisation du BE :
Conception d'un SOPC / Conception conjointe logicielle/matérielle

Choix du FPGA avec processeurs embarqués :
FPGA Intel (Altera) avec processeur NIOS (Altera)
FPGA Xilinx avec processeur Microblaze (Xilinx)

Partitionnement Hard / Soft
Hard : Composants implantés en VHDL
Soft : Processeur softcore NIOS II implanté dans le FPGA
Bus Avalon : Interface de communication entre le hard et le soft utilisé par la famille des softcores NIOS d'Altéra

Remarques :
Les taches à faibles contraintes temps réel seront développées en C
Les autres taches seront supportées par composants numériques spécialisés.

Exemple de partitionnement Hard/Soft



Organisation du Bureaux d'études (1)

1) Présentation des objectifs et de l'organisation du bureau d'études

a. objectifs terminaux du BE :

- *analyse de spécifications et découpage fonctionnel du système choisi.*
- *Conception de circuits d'interfaces numériques en VHDL (conception, simulation, vérification sur maquette)*
- *Notion de Co-design et règles de conception*
- *interfaçage avec bus microprocesseur (NIOS + Altera **Avalon**)*
- *conception d'un SOPC et intégration D'IP (Intellectual Properties) propriétaires et fournisseurs tiers*
- *notions de simulation « Modelsim »*
- *validation du SOPC sur analyseur logique intégré et sur maquette*

b. Nombre de séances (environ 20x4h)

c. Répartition des activités (2 interfaces mini à concevoir par binôme + SOPC)

d. Règles communes de fonctionnement des BE

(absences, avancement du BE (github), évaluation (rapport « notion », ppt diapos, soutenance, vidéo (1mn ~), github « BE-VHDL-Ggb avec g n° du groupe et b n° du binôme », investissement BE, …)

Organisation du Bureaux d'études (2)

2) Présentation des circuits FPGA et tendances actuelles (à faire)

3) Présentation de l'exploration architecturale de la barre franche (donné)

Analyse des besoins et Spécification des interfaces (donné)

http://thierryperisse.free.fr/wp-content/uploads/2020/09/specification_interfaces_pilote_barre_franche.pdf

Décomposition fonctionnelle des interfaces (à faire)(BE)

4) Initiation au langage VHDL (à faire)

Exercices VHDL (TPs) et prise en main de l'environnement Quartus 9.0

(simulation + maquette Terasic DE2) :

Mémorisation, comptage/division, Registre à décalage, MAE, PWM, …

5) Développement et validation des interfaces (Outils logiciels et matériels)(à faire)

Quartus prime 18 lite édition + Modelsim + carte De0 Nano

6) Conception du SOPC et intégration des interfaces (à faire)

Quartus prime 18 lite édition + analyseur logique + carte De0 Nano

7) Développement du soft en langage C (à faire) NIOS II IDE 11 / Eclipse (environnement)

8) Intégration et validation du système (à faire) *NIOS II IDE 11 + DE0 Nano + Maquette*

Choix en BE VHDL (FPGA Altera)

Xilinx est l'inventeur du FPGA.
Xilinx et Altéra (Intel maintenant) sont leaders sur le marché des FPGA

Ils développent tous les deux des programmes universitaires intéressant :

- Soutien des projets de recherche
- Soutien pédagogique avec des donations :

  - Logiciels en version complète et couvrant toute la gamme des FPGA et CPLD Xilinx et Altera.

  - Cartes de développement gratuites ou à des tarifs préférentiel :

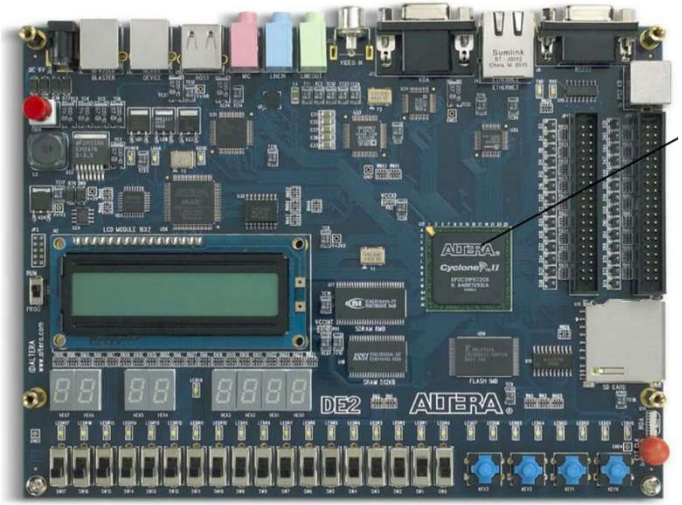
    Altera DE2 à base de FPGA Cyclone II

    Altera DE0 Nano à base de FPGA Cyclone IV

Cartes utilisées en TPs et BE :

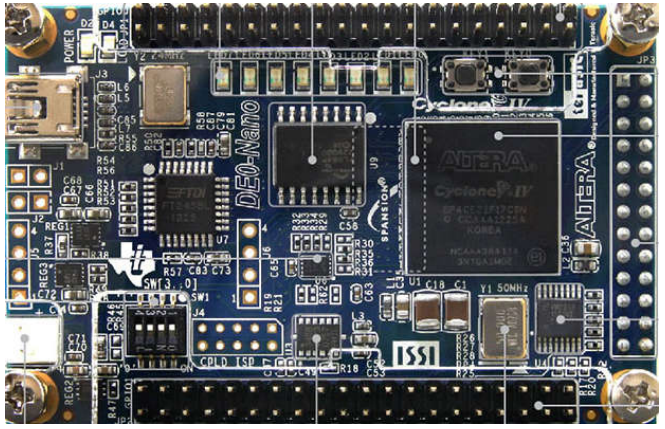
- **TERASIC DE2-C35 équipée d'un FPGA cyclone 2 (TPs + BE)**

- Altera cyclone 2 : ref : EP2C35F672C6N
- 33 216 éléments logiques
- 473 Kbits de SRAM embarquée
- 35 multiplieurs câblés (18x18 bits)
- 4 PLL
- 475 E/S
- 672 broches
- boîtier FBGA (Flip chip Ball Grid Array)



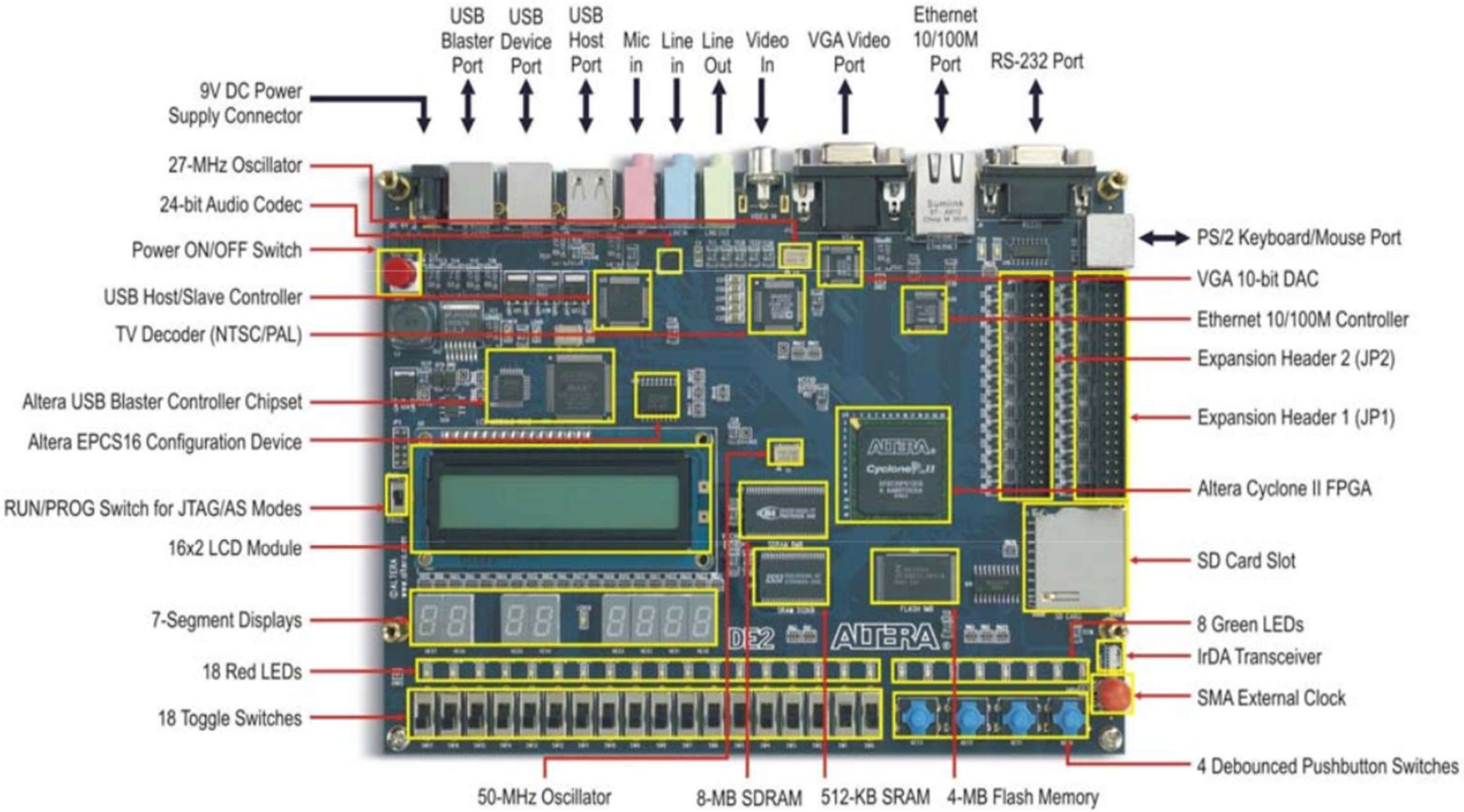
- **DE0 Nano équipée d'un cyclone 4 CE22 (BE)**

- Altera cyclone 4 : ref : EP4CE22F17C6N
- 22 320 éléments logiques
- 594 Kbits de SRAM embarquée
- 66 multiplieurs câblés (18x18 bits)
- 4 PLL
- 153 E/S
- 256 broches
- boîtier FBGA (Flip chip Ball Grid Array)



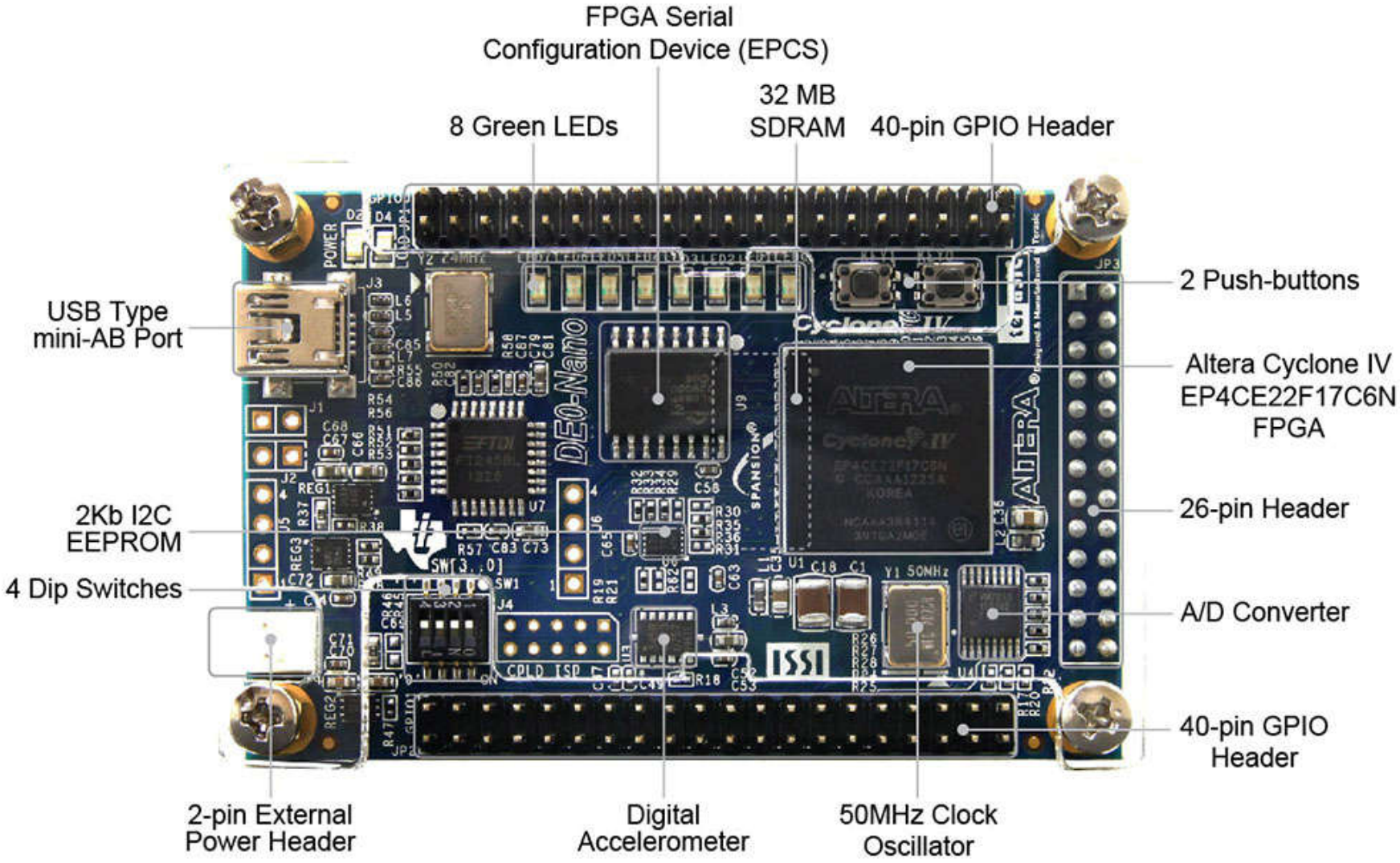
Cartes utilisées en TPs (BE) :

- **TERASIC DE2-C35 équipée d'un FPGA cyclone 2 (TPs + BE)**



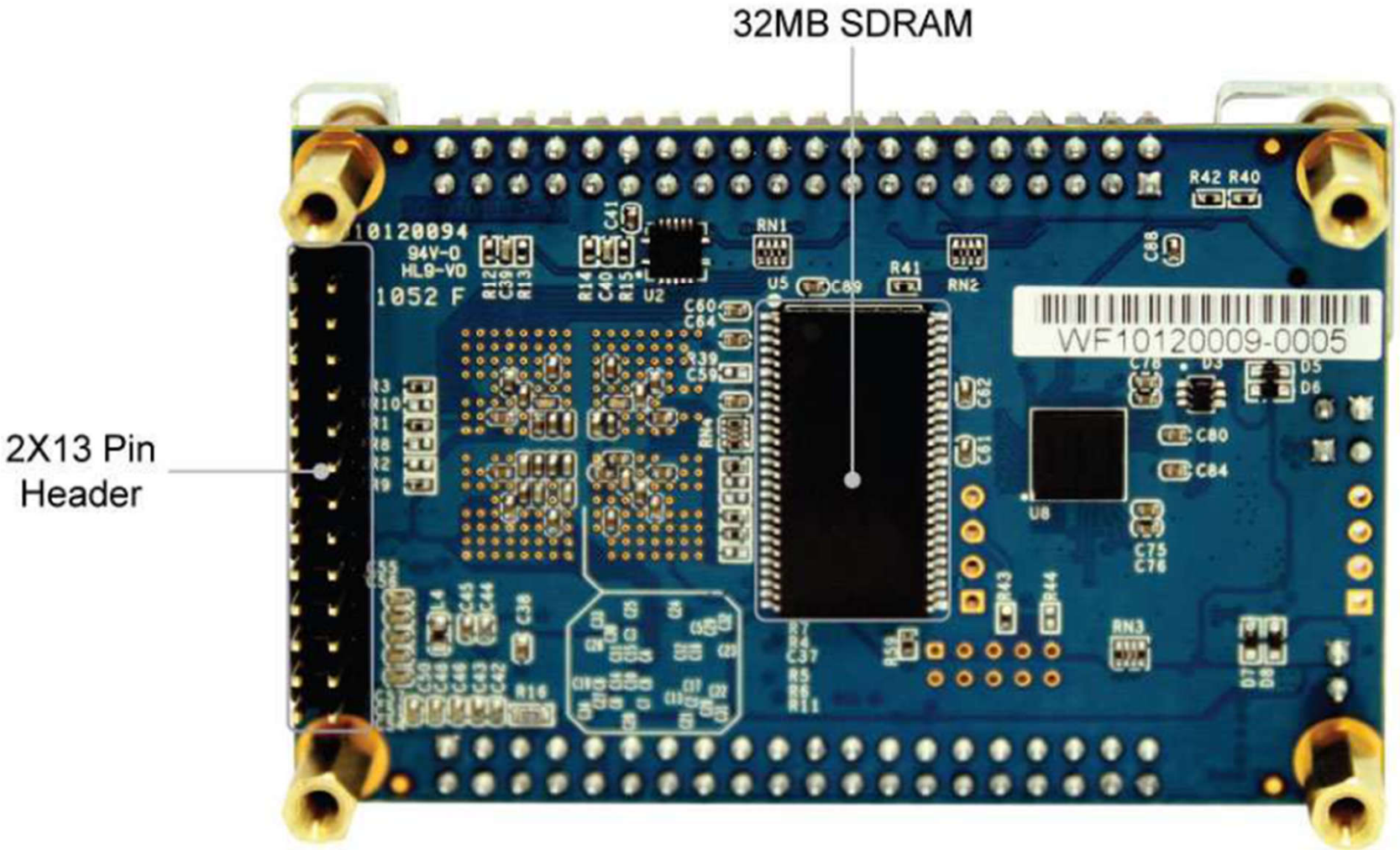
Cartes utilisées en BE :

- **DE0 Nano équipée d'un cyclone 4 CE22 (vue de dessus Top view) Emplacements des connecteurs et des composants clés.**



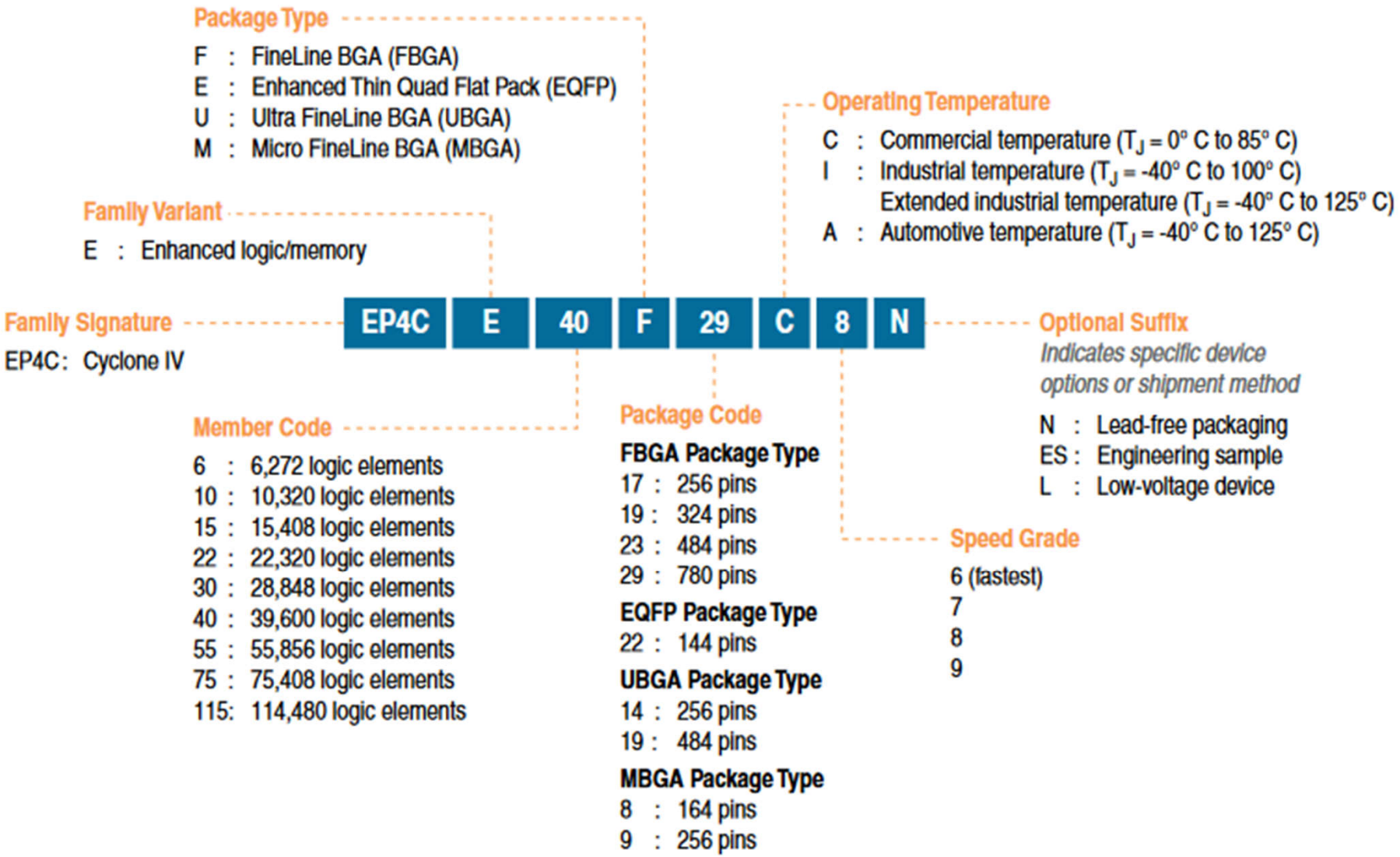
Cartes utilisées en BE :

- **DE0 Nano équipée d'un cyclone 4 CE22 (vue de dessous Bottom view) Emplacements des connecteurs et des composants clés.**



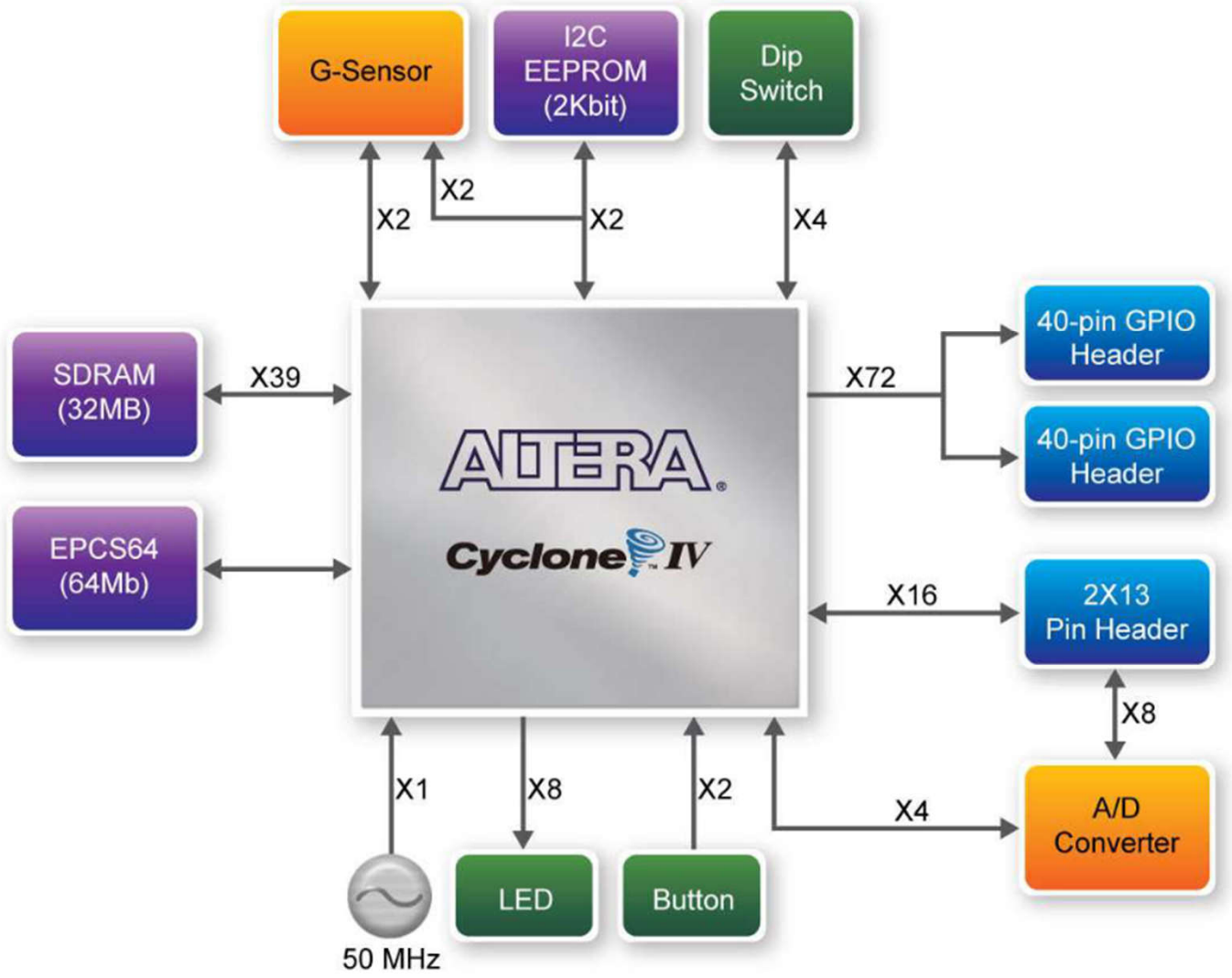
Cartes utilisées en TPs (BE) :

- **Carte DE0 nano équipé d'un Cyclone IV E** EP4CE22F17C6N



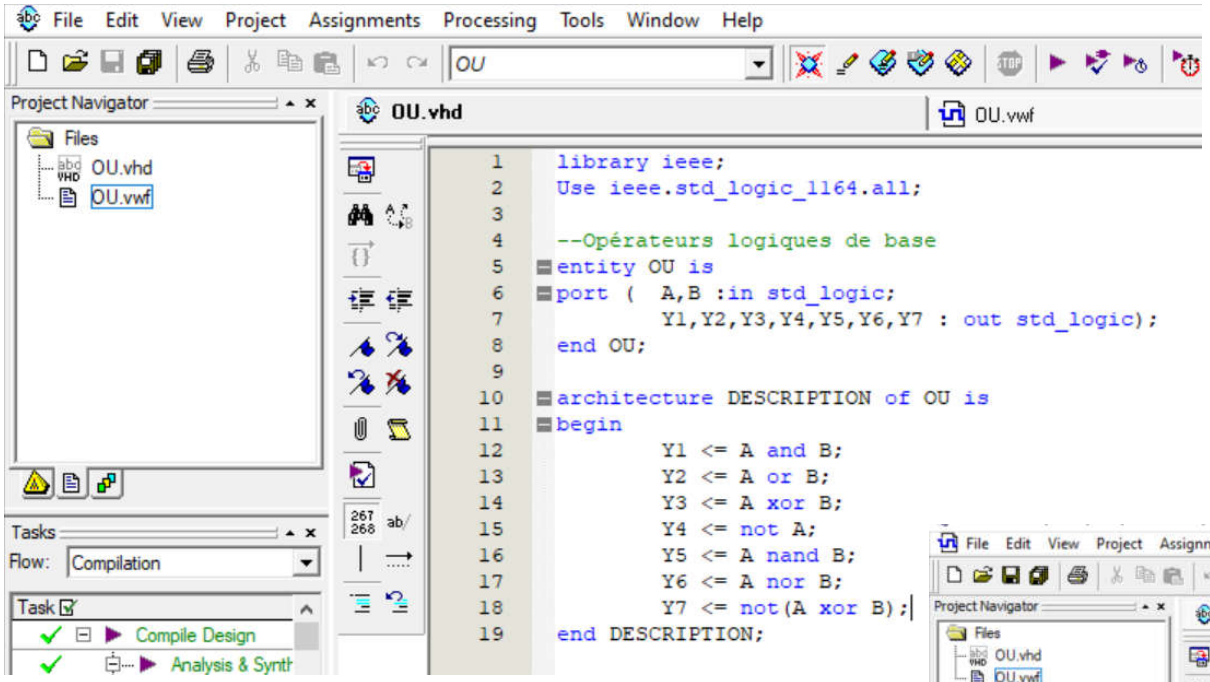
Cartes utilisées en BE :

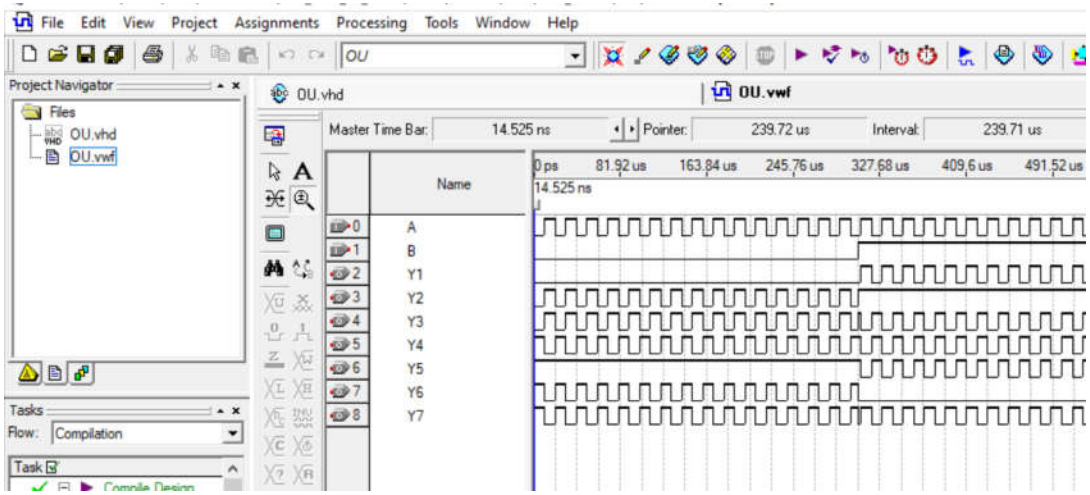
- **DE0 Nano équipée d'un cyclone 4 CE22 (Schéma bloc)**



Outils logiciels TPs et Bes :

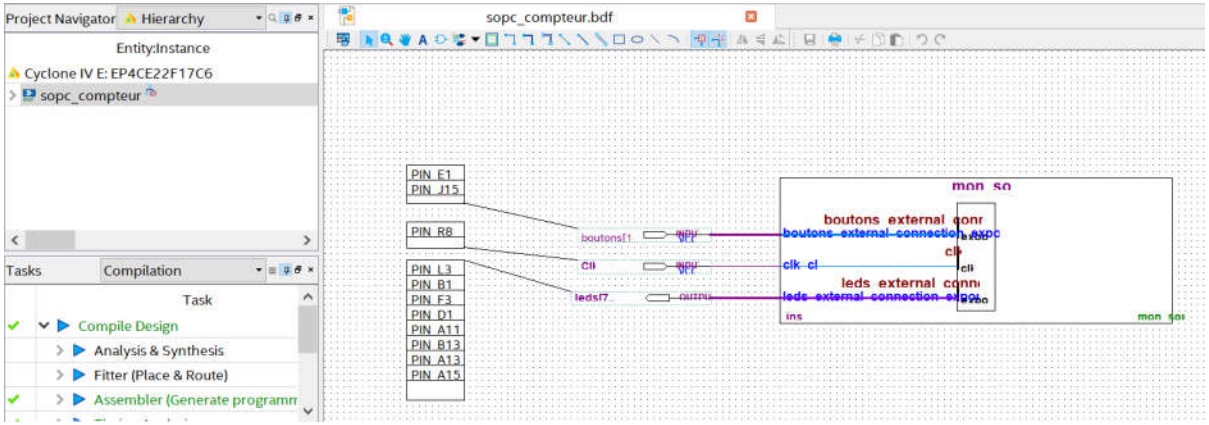
TP : Quartus II vers 9 avec simulateur intégré

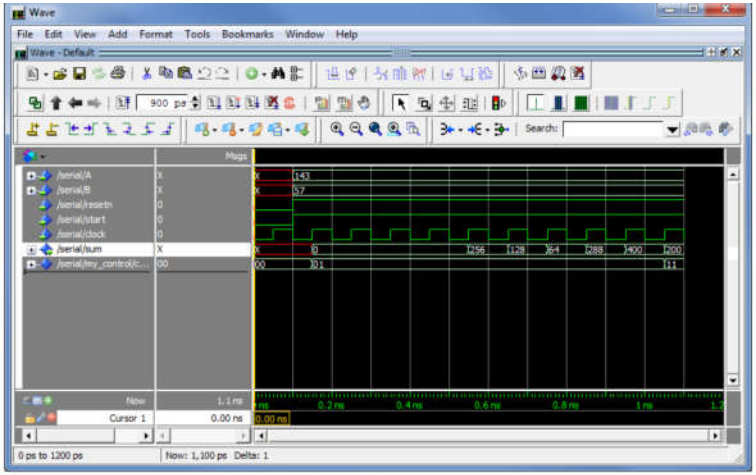


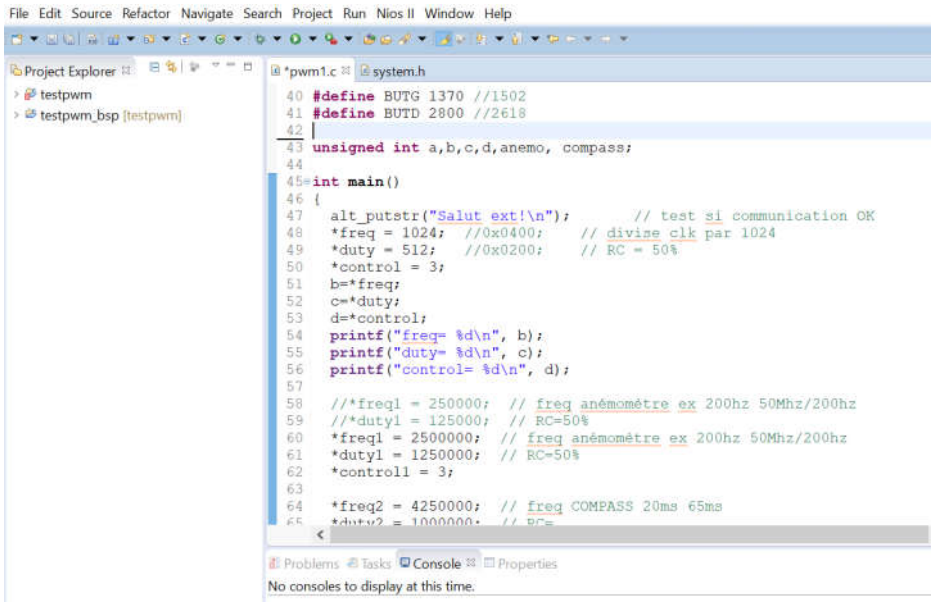


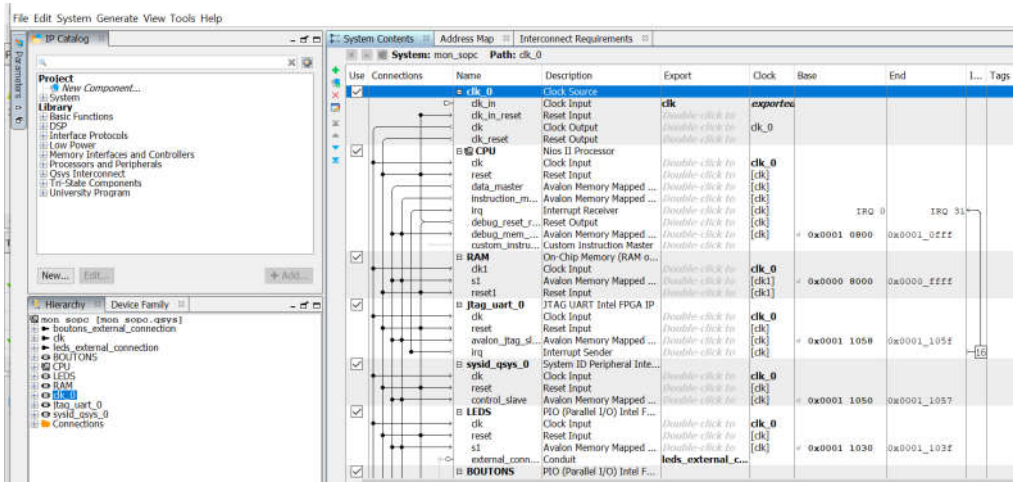
Outils logiciels TPs et Bes :

BE : Quartus II vers 18 lite édition + Plateform designer + Modelsim









Conception des SOC/SOPC (Platform Designer)

Platform Designer permet, entre autres, de concevoir des microcontrôleurs spécifiques à une application. Ces microcontrôleurs comportent donc une partie processeur (softcore NIOS II) à laquelle on associe des périphériques (PIO, Timers, UART, USB, composants propriétaires, ...) et de la mémoire. Cette dernière peut être embarquée dans le FPGA (on parle alors de RAM/ROM On Chip) ou à l'extérieur du composant.

La partie microprocesseur proprement dite est le softcore NIOS II de INTEL, processeur de 32 bits qui se décline en trois versions : économique, standard, rapide. La version économique, la moins puissante, utilise le moins de ressources du FPGA. Bien sûr il est possible d'intégrer d'autres types de processeurs pour peu qu'on dispose de leurs modèles (VHDL, Verilog, ...).

Introduction au VHDL

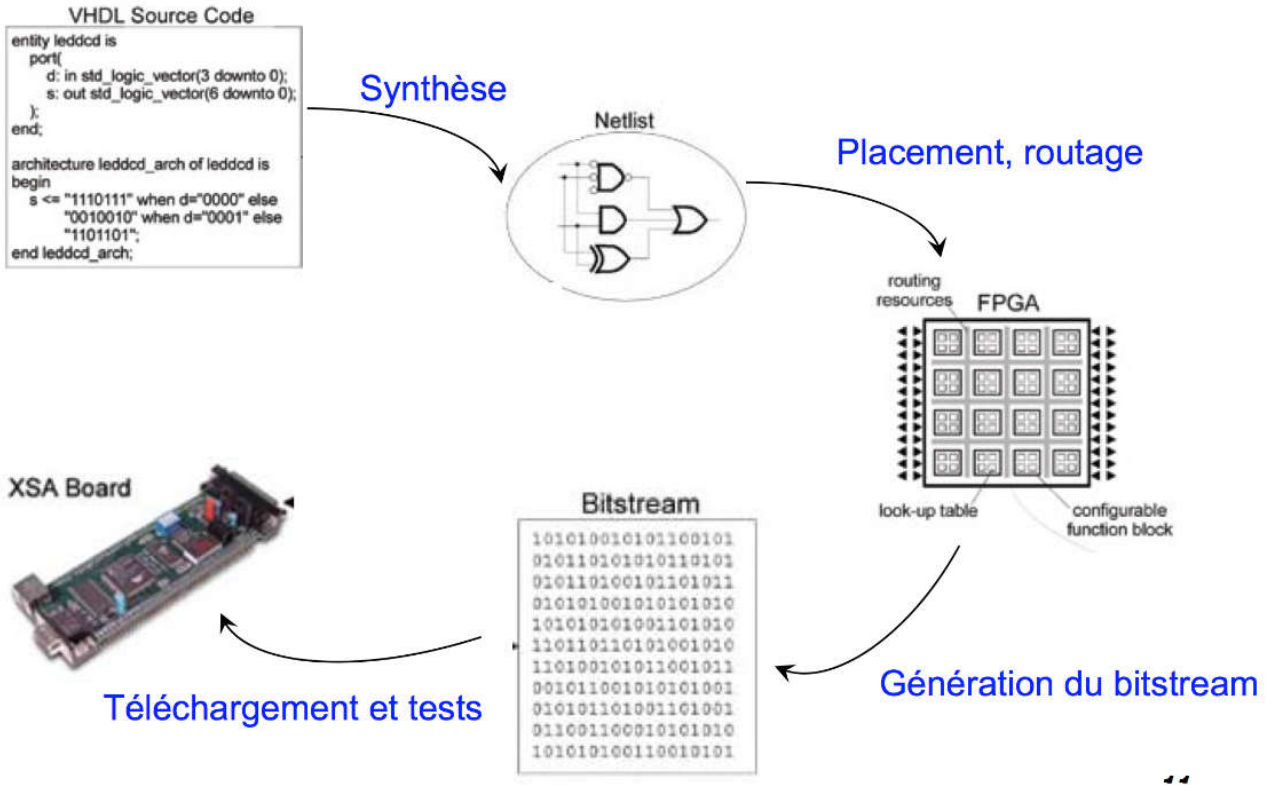
Very High Speed Integrated Circuit **H**ardware **D**escription Language

(Language de description matériel)

C'est un standard de L'IEEE (Institute of Electrical and Electronics Engineers). C'est un langage de description de haut niveau pour la simulation et la synthèse de circuits électroniques numériques.

Etapes de développements :

Saisie du texte VHDL
Analyse syntaxique
Simulation fonctionnelle
Choix du composant
Placement et Routage
Simulation temporelle
Programmation du composant
Vérification du fonctionnement



Introduction au VHDL

Quelques règles pour la conception d'un système complexe:

1) Privilégier une approche « Top-Down »
- identifier les fonctions principales et éventuellement les décomposer pour aboutir à des fonctions plus simples à concevoir
- identifier les entrées/sorties et les signaux d'interconnexion entre les différentes fonctions
- si les fonctions sont utilisées plusieurs fois, privilégier une approche « composant » sinon on peut associer à une fonction un « process »
2) Labelliser les fonctions, signaux et variables de manière claire afin de faciliter la phase de mise au point.
Exemple :
- s_fp1_nom signal *pour un signal généré par la fonction fp1 et actif à 1*
- s_fp1_nom signal_n *pour un signal généré par la fonction fp1 et actif à 0*
- v_fp1_nom variable *pour une variable interne à la fonction fp1*
3) prévoir un signal de « remise à zéro » qui remet le circuit qui a été conçu dans une configuration connue (surtout si le circuit a un fonctionnement séquentiel)
4) valider chaque fonction séparément avant intégration dans le niveau supérieur (tests unitaires)
5) synchroniser les signaux d'entrée asynchrones/système pour éviter les aléas de fonctionnement
6) Enfin (le plus difficile !) penser « circuit » et surtout pas « programme informatique » sinon c'est l'échec assuré !