MI01

Automne 2011

Shawky à utc.fr

POURQUOI MI01

- Puissance de calcul
  - Vidéo hologram CNN
  - Vidéo nanotronics

OBJECTIFS MI01

- Support de traitement de l'information
  - Electronique numérique
- Langage de modélisation matérielle (hardware)
  - VHDL (Hardware Description Language)
- Compréhension d'un microprocesseur récent
  - Pentium
- Autres microcontrôleurs
  - Applications embarquées
  - Rabbit 2100 (4 g)

PROGRAMME DE L'UV

- Introduction
- Rappel rapide algèbre de Boole
- Modélisation en VHDL (Hardware Description Language)
  - Modélisation combinatoire et séquentielle
  - Composants programmables (PLD, FPGA, etc.)
  - Des machines à états au micro-programmé
- Noyau d'un processeur
- Assembleur Pentium (famille x86)
- Aspects avancés Pentium 4, Mobile, etc.
  - Hyperpipeline, Superscalaire, MMX et SIMD2
  - Hyperthreading, multi-core

WHAT'S NEW?

- A11
  - Multicoeurs mobile
- A09 - A08
  - Cours orienté informatique embarquée
  - Pentium multi-core, hyper threading
- A07 - A04
  - Plus de TP
    - 6 TP en alternance → 10 TP sans alternance
    - + TP assembleur Pentium
    - TP rabbit
  - Cours
    - Pentium Mobile, double core

MI01 : PRATIQUE

- Site web
  - Moodle
- Initiative ePolycopié
  - Transparents cours, mais pas notes au tableau
  - Sujets de TD et de TP
    - A télécharger sur le site web avant le TD ou TP
    - Document TP en pdf, plus version papier en salle TP

DATES IMPORTANTES

- Cours : mardi 8H
- Début TD : semaine du lundi 19 septembre
- **Début TP : semaine du lundi 3 octobre**

EVALUATION

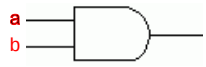
- **Enseignements :** C 32h TD 32h TP 20h
- **Évaluation :** Médian, final et notations TP
- **Attribution :** 0,3 x médian + 0,5 x final + 0,2 x TP > 10
  - 3 TP aléatoires corrigés sur 10

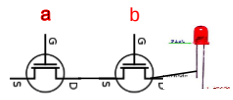
OBSESSION DU NUMÉRIQUE

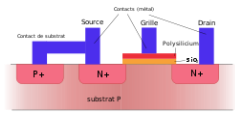
MAGIE DU NUMÉRIQUE

- Pourquoi à chaque fois qu'on parle du « numérique », on parle de « logique binaire»?
- Le binaire est magique !
  - Très résistant au « bruit électronique »
  - Facilement « composable »
    - On peut cascader les composants à souhait

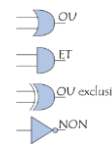
FONCTIONS LOGIQUES LES PLUS UTILISÉES

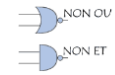


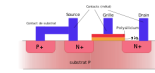


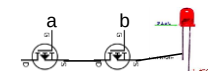


FONCTIONS LOGIQUES LES PLUS UTILISÉES









RAPPEL LOGIQUE BINAIRE

- http://www.commentcamarche.net/contents/logic/intro.php3

LOGIQUE COMBINATOIRE

- George Boole est le père fondateur de la logique moderne et son algèbre booléenne nous permet de résoudre les problèmes de logique combinatoire
- En abordant le concept de logique combinatoire avec **l'algèbre de Boole** comme outil mathématique nous étudions les principales combinaisons logiques souvent utilisées à des fins techniques
- Etudier principalement les fonctions logiques qui permettent de réaliser des fonctions arithmétiques
  - Fonctions logiques ET, OU, NON
  - Fonctions arithmétiques : addition, multiplication

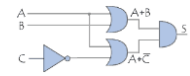
LES LOIS DE COMPOSITION

- Les lois de composition sont des règles logiques qui permettent de simplifier l'écriture des expressions algébriques.

Associativité (A.B).C est équivalent à A.(B.C) (A+B)+C est équivalent à A+(B+C)	**Idempotence** A.A est équivalent à A A + A est équivalent à A
Absorption A.(A+B) est équivalent à A A+A.B est équivalent à A	**Identité** 1.A est équivalent à A 0+A est équivalent à A
Commutativité A.B est équivalent à B.A A+B est équivalent à B+A	**Inversion** A./A est équivalent à 0 A+/A est équivalent à 1
Distributivité A+(B.C) est équivalent à (A+B).(A+C) A.(B+C) est équivalent à A.B+A.C	**Nullité** 0.A est équivalent à 0 1+A est équivalent à 1

COMPOSITION DE FONCTIONS

- Par exemple l'expression algébrique
  - (A+B).(A+/C)sera schématisée comme suit :



ADDITIONNEUR DE DEUX NOMBRES DE 1 BIT

- Pour une addition de deux nombres A et B de 1 bit, 4 combinaisons sont possibles, et le résultat occupe 2 bits : un bit pour la somme (S) et un pour la retenue (R).
- Voici la table de vérité de cette fonction :

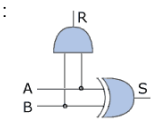
Entrée		Sortie	
A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

ADDITIONNEUR

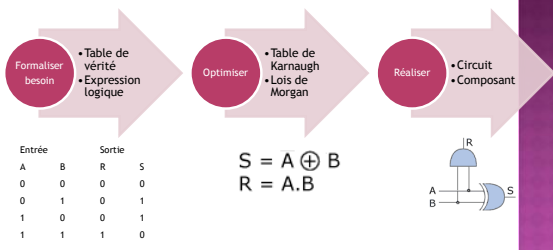
L'expression logique de cette fonction est donc :

$$S = \overline{A} \oplus B$$
$$R = A.B$$

Le circuit peut donc être représenté selon le schéma électrique suivant :



DU BESOIN À LA RÉALISATION



Entrée		Sortie	
A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$R = A.B$

VHDL

Introduction

Spécification d 'architecture

- Appelée aussi : Circuit
  - Centaines, milliers, voire millions de portes
- Définir (spécifier) ses fonctionnalités
- *Se fait en niveaux, pour mieux comprendre les fonctions*
  - Plus haut niveau : abstrait
  - Plus bas niveau : la fonction unitaire de base

SPÉCIFICATION D 'ARCHITECTURE

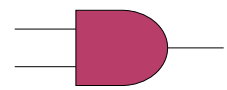
- Approche Top down
  - Commencer par spécifier le haut niveau
  - Rajouter les détails
    - (fonctions de bibliothèque)
  - Définir les cellules de base

SPÉCIFICATION D 'ARCHITECTURE

- Approche bottom-up
  - commencer d 'abord par les fonctions de base
  - constituer une bibliothèque
  - instancier et construire des macros blocs
  - Remonter jusqu 'au sommet

OUTIL DE SPÉCIFICATION D 'UNE ARCHITECTURE

• La spécification schématique assez limitée



- ré-utilisabilité
- extensibilité
- modularité
- maintenance (modification)

OUTIL DE SPÉCIFICATION D 'UNE ARCHITECTURE

- Premières représentations textuelles (années 80)
  - assez simples
  - s = a * b
    - s un nom libre désignant une sortie (à gauche de =)
    - a et b noms libres de signaux d 'entrée
    - * opérateur désignant l 'opération logique ' et '
  - Utilisées surtout pour la définition des fonctions de composants programmables
    - pour éviter de programmer en matrices de fusibles

PROBLÈME

- Autant de langages que de fabricants de composants
  - 10 - 12 langages différents et proches en même temps
- Nécessité d 'uniformiser et de standardiser

NAISSANCE DU VHDL

- But : Normalisation générale des langages de spécification matérielle
- DoD met au point un langage
  - initialement pour le développement de circuits intégrés
  - VHDL : VHSIC HDL
    - Very High Speed Integrated Circuits Hardware Description Language
- Transfert de tous les droits sur le langage à IEEE society

RÉFÉRENCES

- Recherche catalogue BUTC avec 'VHDL'
- Liens Internet !
- ' VHDL '
  - Douglas L. Perry
  - Edition Mc Graw-Hill

VHDL

- Meilleure façon
  - aborder le langage par des exemples
  - revenir ensuite aux règles syntaxiques et sémantiques
- Aspects avancés
  - purement structurel ou comportemental
  - combinatoire et séquentiel
  - concurrence de fonctionnement
  - comprendre l 'opération de synthèse

VHDL

```
-- inverseur (ceci est un commentaire)
ENTITY inverseur IS
-- en majuscule les mots réservés (pas obligatoire)
PORT (e : IN BIT ; -- l 'entrée
      s : OUT BIT ) ; la sortie
END inverseur ;
```

Analyse

ENTITY : l 'interaction entre l 'entité et le monde extérieur
e,s des noms libres
IN BIT entrée de type BIT (1 seul fil)
OUT BIT sortie de type BIT (1 seul fil)

VHDL

```
ARCHITECTURE toto1 OF inverseur IS
BEGIN
  s <= NOT e ;
END toto1 ;
```

Analyse :

ARCHITECTURE description du fonctionnement interne de l 'entité
NOT opérateur logique d 'inversion
toto1 nom libre donné à l 'architecture
inverseur nom de l 'entité déjà déclarée avec ENTITY

VHDL

-- opérateur ET à 2 entrées

```
ENTITY et2 IS
PORT (a,b : IN BIT ;
        s : OUT BIT ) ;
END et2 ;
ARCHITECTURE titi1 of et2 IS
BEGIN
  s <= a AND b ;
END titi1 ;
```

-- ou à 2 entrées

```
ENTITY ou2 IS
PORT (a,b : IN BIT ;
        s : OUT BIT ) ;
END ou2 ;
ARCHITECTURE titi2 of ou2 IS
BEGIN
  s <= a OR b ;
END titi2 ;
```


VHDL

- Opérateurs de base compris par le VHDL standard
  - AND, NOT, OR, XOR
  - très limité => la clé de la portabilité du VHDL
  - La bibliothèque d 'opérateurs étendus est donc indispensable
    - écrite en utilisant les opérateurs de base
    - excellente portabilité

AFFECTATION CONDITIONNELLE

Exemple :

```
-- inverseur
ENTITY inverseur IS
PORT (e : IN BIT;
        s : OUT BIT);
END inverseur;

ARCHITECTURE bis OF inverseur IS
BEGIN
  s <= '1' WHEN (e='0') ELSE '0';
END bis;
```

WHEN affectation conditionnelle

simples guillemets pour l'affectation d'un niveau pour le type BIT '1'

possibilité d'un ELSE

Affectation sélective

Idem pour la porte ET

```
-- opérateur ET
ENTITY et2 IS
PORT (e1, e2 : IN BIT);
PORT (s : OUT BIT);
END et2;
ARCHITECTURE encore OF et2 IS
BEGIN
s <= '0' WHEN (e1 = '0' OR e2 = '0') ELSE '1' ;
END encore;
```


Règles syntaxiques et sémantiques

Affectation de signaux

* Affectation simple :

nom_de_signal <= expression_de_type_compatible

signifie une connexion entre 2 équipotentielles

* Affectation conditionnelle

```
nom_de_signal <= source_1 WHEN condition1 ELSE
                 source_2 WHEN condition2 ELSE
                 source_n ;
```

Affectation sélective

WITH expression SELECT
nom_signal <= source1 WHEN valeur1, *virgule*
source2 WHEN valeur2,
sourcen WHEN OTHERS ; *point virgule*

syntaxe inspirée de ADA (PASCAL), donc fortement typée (en contraste avec le C)

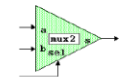
Comment synthétiser cette construction ?

Multiplexeur ?

Exemple 1a: Multiplexeur 2 voies

Définition d'entité

```
ENTITY mux2 IS
 PORT (a,b,sel: IN std_logic; s: OUT std_logic);
END;
```



Description comportementale

```
ARCHITECTURE comportement OF mux2 IS
BEGIN
 s <= a WHEN sel='0' ELSE b WHEN sel='1' ELSE 'X';
END comportement;
```


Catégories de "conteneurs"

Classes de conteneurs *on utilisera le terme «conteneur» plus global que variable...*

I- signaux :

Un signal représente une valeur physique (équipotentielle) échangée entre les blocs d'une fonction

SIGNAL nom1, nom2 : *type* ;

nom1 <= valeur_compatible_avec_le_type *la valeur peut-être le résultat d'une expression, suivant les techniques d'affectation de la section précédente*

Variables

conteneurs qui servent à stocker un résultat intermédiaire, utile surtout pour un algorithme séquentiel mais qui ne sera pas forcément synthétisé physiquement.

Utilisées uniquement à l'intérieur d'un process, une procédure ou une fonction.

VARIABLE nom1, nom2 : type ;

nom1 := valeur_compatible_avec_le_type ; *la valeur peut-être le résultat d'une expression, ou la valeur renvoyée d'une fonction*

Différence entre VARIABLE et SIGNAL

pas nécessairement synthétisé

forcément synthétisé => existe physiquement

** possibilité d'affecter la valeur d'une VARIABLE à SIGNAL et vice-versa, si de même type*

Constantes

III- Constantes

CONSTANT nom1 : type;

ex : '0', '1', «abcde»

Types de "conteneurs"

le nombre de type est assez limité, et ils sont orientés électronique numérique.

Extension analogique (VHDL-A) comporte d'autres types convenables à l'électronique analogique.

I- type entier

Valeurs sur 32 bits

-2^{31} à $2^{31} - 1$ *tous les synthétiseurs n'acceptent pas les valeurs négatives pour un conteneurs. Pour un SIGNAL sur un bus composé de plusieurs lignes, il attribue automatiquement le complément à 2*

Exemple

SIGNAL nom : integer ;

ou VARIABLE nom : integer ;

Types de "conteneurs"

Remarque : Définition de plage

SIGNAL valeur : integer RANGE 0 TO 9 ;

II - Type énuméré

TYPE feu IS (vert, orange, rouge);

-- sous type créé

SIGNAL feu1 : feu ;

III - Type bit

2 valeurs possibles : '0', '1'

SIGNAL | VARIABLE nom : BIT ;

IV- Type vecteur

SIGNAL bus_d : BIT_VECTOR (0 to 7) ; *définit un vecteur de taille 8 bits*

Le vecteur peut-être manipulé en entier, ou fil par fil

Ex : SIGNAL busd_micro : BIT_VECTOR (7 DOWNTO 0)

SIGNAL busd_mem : BIT_VECTOR (7 DOWNTO 0)

busd_mem <= busd_micro ; ***taille identique !***

Résumons les points couverts

- Modélisation d'une fonction combinatoire

  Pour l'instant, aucune notion de :

  - temps
  - ni d'ordre de fonctionnement
- Différentes catégories de «conteneurs»
  - SIGNAL
  - VARIABLE
  - CONSTANT
- Différents types pour chaque catégorie (pas toutes)
  - ENTIER RANGE 0 TO 255 => déduire le nombre de bits
  - BIT
  - VECTEUR
- Affectations conditionnelles ou sélectives