UE VLSI
cours 4: Introduction aux HDL, le langage VHDL

Jean-Lou Desbarbieux
UPMC 2017

Sommaire

Objectif

Les **HDL** *Hardware Descryption Language* sont des éléments clef de la conception VLSI. Ils ont été conçus pour :

- Permettre la spécification non ambiguë d'un composant ;
- Simuler le plus tôt possible le comportement d'un composant ;
- Automatiser la conception ;
- Permettre les échanges entre les différents acteurs de la conception d'un système.

Initialement conçus pour la CAO électronique numérique, ils sont également maintenant capable de modéliser des fonctions analogiques voir mécaniques.

L'offre

Spice Berkley 1970.

Verilog 1984.

VHDL 1987.

VHDL-AMS 1999.

Verilog-AMS 2005.

SystemC 2005.

SystemC-AMS 2010.

Il existe d'autres HDL mais de moindre diffusion.

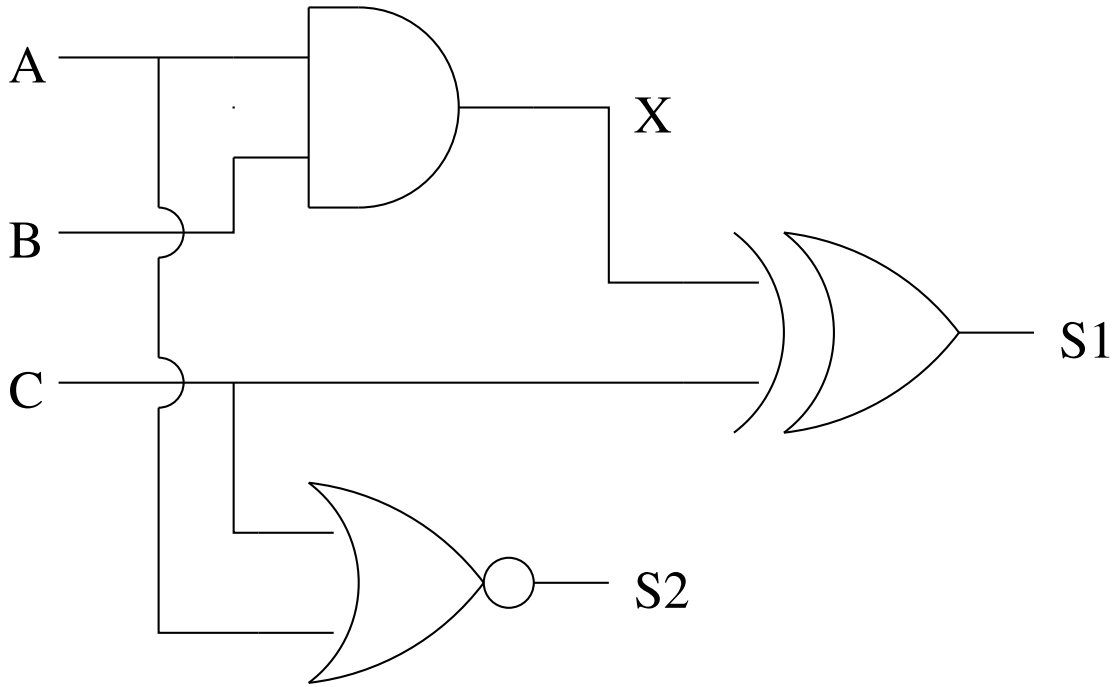
Système numérique

Un *HDL* doit permettre de représenter les valeurs électriques/numériques portées par un fil.

- Vrai ;
- Faux ;
- Haute impédance ;
- ...

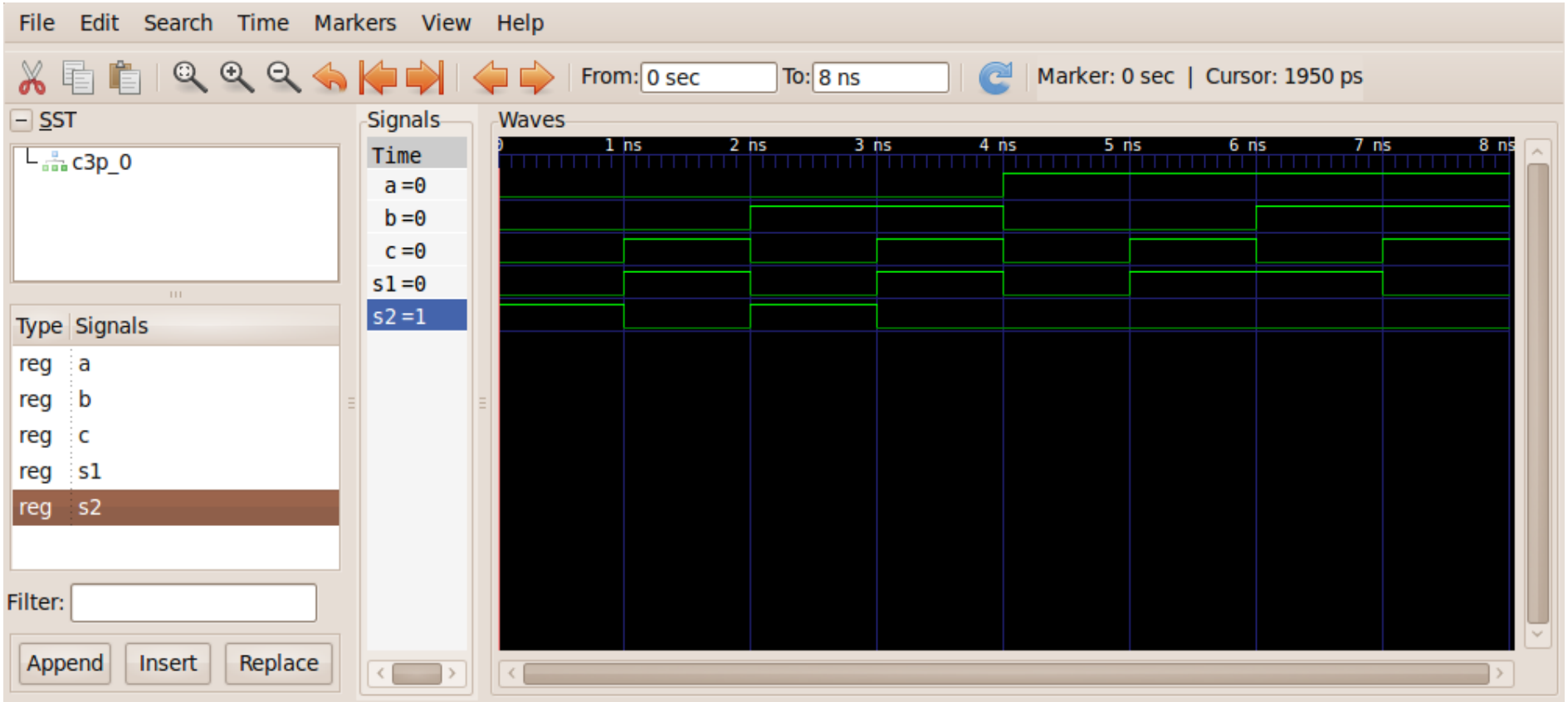
Système numérique

Un *HDL* doit permettre de décrire un système matériel qui est intrinsèquement concurrent (parallèle) et le simuler avec un ordinateur séquentiel.



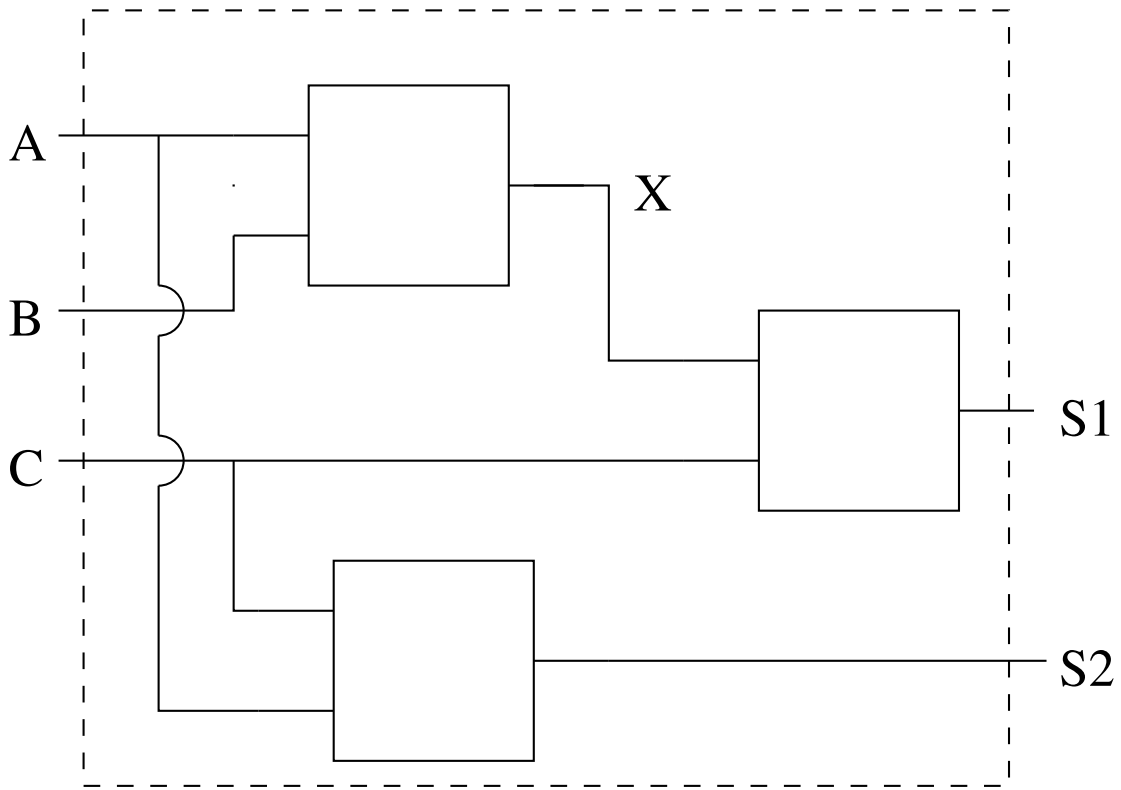
Le temps

Un *HDL* doit permettre de représenter le temps au moins de façon discrète.



Connexion de composants

Un *HDL* doit permettre de lister les composants d'un système et leurs connexions.



Structure d'un fichier VHDL

Le VHDL n'est pas sensible à la casse.

```
-- declaration des bibliotheques STD et WORK
-- sont connues par defaut
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- declaration de l interface
ENTITY et IS
PORT (
        i0, i1 : IN BIT;
        q : OUT BIT
);
END ENTITY;

-- description du composant
ARCHITECTURE nom_archi OF et IS
-- declaration des signaux
SIGNAL tmp : BIT;
BEGIN
tmp <= i0 AND i1;
q <= tmp;
END nom_archi;
```

Types entiers

Le type entier `integer` prédéfini dans la bibliothèque standard `STD` permet de stocker des nombres signés sur 32 bits.
Un sous type `subtype` permet de déclarer un type héritant des propriétés du type père.
Il existe 2 sous types associés à `INTEGER` : les entiers naturels et les entiers positifs. Leur déclaration dans la bibliothèque `STD` est la suivante :

```
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
```

Les attributs sont des caractéristiques de types ou d'objet qu'il est possible d'utiliser dans le modèle. Ils sont représentés de cette façon :
`<OBJET>'<ATTRIBUT>`
Il existe des attributs sur les types, sur les objets de type tableau et sur les signaux. Il est possible de créer ces propres attributs.

Types énumérés

Un type énuméré est un type défini par une énumération exhaustive :

```
type COULEURS is (ROUGE, JAUNE, BLEU, VERT, ORANGE);
```

Dans la bibliothèque STD on trouve notamment les déclarations de types suivantes :

```
type boolean is (FALSE, TRUE);
type bit is ('0', '1');
```

Nous allons trouver dans la bibliothèque IEEE la définition du fameux type STD_LOGIC (le standard industriel) :

```
TYPE std_ulogic IS ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care );
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic
```

Types tableaux

Les types tableau ou ARRAY sont des collections d'objets de même type, indexés par des entiers ou des énumérés.

```
TYPE vecteur IS ARRAY (0 TO 2) OF INTEGER;
```

La taille des types tableaux peut ne pas être définie statiquement :

```
TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_log
```

La taille doit être fixée lors de la déclaration des variables (ou signaux).
Le type STRING permet de définir les chaînes de caractères (bibliothèque STD).

```
TYPE string IS ARRAY (POSITIVE RANGE <>) OF character;
```

Processus

Les PROCESS définissent des sections de code séquentielles, notre circuit comportant 3 portes peut être décrit comme suit :

```
architecture behavior of C3P is

signal X : std_logic;

begin
        PROCESS (A, B)
        BEGIN
                X <= A AND B;
        END PROCESS;

        PROCESS (C, X)
        BEGIN
                S1 <= C XOR X;
        END PROCESS;

        PROCESS (C, A)
        BEGIN
                S2 <= C NOR A;
        END PROCESS;

end behavior;
```

Synchronisation

À défaut un PROCESS est répété indéfiniment, il existe plusieurs méthodes pour contrôler un processus :

- La liste de sensibilité : le PROCESS n'est réévalué que que si un des signaux définit dans la liste de sensibilité change de valeur ;

```
PROCESS (A, B) -- process sensibles aux signaux A et B
BEGIN
        X <= A AND B;
END PROCESS;
```

- L'instruction WAIT permet de suspendre l'exécution d'un processus :

```
PROCESS
BEGIN
        WAIT ON A, B; -- Attente evenement sur A et/ou B
        X <= A AND B;
END PROCESS;
```

- La forme WAIT ON est équivalente à une liste de sensibilité.
- La forme WAIT FOR permet de spécifier un délai d'attente.
- La forme WAIT UNTIL permet de spécifier une condition. Une liste de sensibilité constituée de tous les signaux
- L'instruction WAIT sans argument suspend définitivement le processus.

Variable versus signal

Une variable en VHDL se comporte comme toute variable d'un langage séquentiel, son affectation est instantanée et est réalisée par l'opérateur :=.
Un signal permet d'exprimer la concurrence du matériel en VHDL, son affectation est différée et est réalisée par l'opérateur <=.

```
ARCHITECTURE behav OF c3p IS

BEGIN
        PROCESS (A, B, C)
        VARIABLE X : std_logic;
        BEGIN
                X := A AND B;
                S1 <= C XOR X;
        END PROCESS;

        PROCESS (C, A)
        BEGIN
                S2 <= C NOR A;
        END PROCESS;

END behav;
```

Flot de donnée

Du fait du caractère différé de l'affection des signaux, celle-ci peut être réalisée hors de tout processus :

```
ARCHITECTURE dataflow OF c3p IS

SIGNAL X : STD_LOGIC;

BEGIN
        X <= A AND B;
        S1 <= C XOR X;
        S2 <= C NOR A;

END dataflow;
```

Instructions séquentielles (1/3)

- Instruction conditionnelle :

```
IF A = '1' THEN
        X := B;
ELSE
        X := '0';
END IF;
```

- Instruction sélective :

```
PROCESS (C, A)
VARIABLE CetA : std_logic_vector(1 DOWNTO 0);
BEGIN
        CetA := C & A;
        CASE CetA IS
                WHEN "00" => S2 <= '1';
                WHEN OTHERS => S2 <= '0';
        END CASE;
END PROCESS;
```

Instructions séquentielles (2/3) les boucles

- Boucle générale :

```
LOOP
        clock <= NOT clock;
        WAIT FOR 5 ns;
        clock <= NOT clock;
        WAIT FOR 5 ns;
END LOOP;
```

- Boucle while :

```
VARIABLE cpt : UNSIGNED(3 DOWNTO 0) :=0;
...
WHILE cpt < X"F" LOOP
        cpt := cpt + 1;
END LOOP;
```

- Boucle for :

```
FOR cpt IN 1 TO 7 LOOP
        ...
END LOOP;
```

Instructions séquentielles (3/3)

Il est possible avec les assertions de vérifier des règles lors de la simulation d'un système :

```
ASSERT vs1 = S1 REPORT "Erreur sur S1" SEVERITY ERROR;
```

Le niveau de sévérité est aussi mentionné dans le message produit dans la console. Le niveau de sévérité par défaut est `NOTE`.

```
type severity_level is (NOTE, WARNING, ERROR, FAILURE);
```

L'instruction `REPORT` permet d'afficher un message dans la console indépendemment de toute condition d'assertion. Le niveau de sévérité est `NOTE`.

Flot de donnée

Du fait du caractère différé de l'affection des signaux, celle-ci peut être réalisée hors de tout processus :

```
ARCHITECTURE dataflow OF c3p IS

SIGNAL X : STD_LOGIC;

BEGIN
        X <= A AND B;
        S1 <= C XOR X;
        S2 <= C NOR A;

END dataflow;
```

Pas de variables (utilisation uniquement dans les `PROCESS`), seulement des signaux.

Toutes les affectations s'effectuent de façon concurrente.

Instructions concurrentes

- Instruction conditionnelle :

```
X <= B WHEN A = '1' ELSE '0';
```

- Instruction sélective :

```
CetA := C & A;
WITH CetA SELECT
        S2 <= '1' WHEN "00",
                  '0' WHEN OTHERS;
```

Gestion du temps

Lors d'une simulation la gestion du temps repose sur un échéancier, des événements sont placés sur cet échéancier et peuvent provoquer l'évaluation de processus.
Deux primitives VHDL permettent d'ajouter des événements à cet échéancier :

`WAIT` suspend l'exécution d'un processus jusqu'à une date ;

`AFTER` poste dans le futur la modification d'un signal.

```
X <= A AND B AFTER 2 ns;
```

Gestion du temps

La simulation d'un fichier VHDL se réalise en trois temps :

- l'analyse peut s'apparenter à la compilation (`gcc -c`) et produit avec GHDL un `.o` et ajoute l'entité produite au fichier listant le contenu d'une bibliothèque (`work-obj93.cf` par défaut).
- l'élaboration réalise le lien entre les entités et produit un exécutable.
- la simulation (exécution du fichier avec GHDL) produit des affichages sur le terminal (`REPORT ...`) et peut générer un fichier de trace contenant la valeur des signaux aux différents temps. Cette trace peut être consultée graphiquement avec *gtkwave*.

Génération des stimuli

Pour tester un composant il est nécessaire d'écrire un programme VHDL contenant au moins un processus dont le rôle est de :

- Affecter des valeurs aux entrées du circuit à tester ;
- Faire progresser le temps ;
- Vérifier la validité des sorties produites par le circuit en cours de test.

Voir exemple du test exhaustif du circuit à 3 portes.

GHDL

GHDL est un compilateur VHDL au code source ouvert (*open source*) développé par Tristan Gingold, il s'appuie sur gcc et supporte la norme VHDL 93.

`http://ghdl.free.fr`

Voir le fichier `Makefile` pour son utilisation.