ÉCOLE NATIONALE SUPÉRIEURE
de SCIENCES APPLIQUÉES et de TECHNOLOGIE

TECHNOPOLE ANTICIPA LANNION UNIVERSITÉ DE RENNES 1

De la description d'un système à la description des transistors en VHDL

D.Chillet

LOGICIEL et SYSTÈME INFORMATIQUE - ÉLECTRONIQUE et INFORMATIQUE INDUSTRIELLE - OPTRONIQUE

ENSSAT - Technopole Anticipa - BP 447 - 6 rue de Kérampont - 22305 LANNION Cedex - Siret 193 509 361 00278 - APE 9215
Téléphone (33) 02 96 46 50 30 - Télécopie (33) 02 96 37 01 99 - Email ...@enssat.fr - Minitel 36 15 ENSSAT - Serveur WWW http://www.enssat.fr/

Table des matières

0.1 Introduction 1
0.2 Présentation du système à concevoir - Conception de niveau N 1
0.2.1 Vue externe du système 1
0.2.2 Vue interne du système 2
0.2.3 Test du système 3
0.3 Conception de niveau $N-1$ 4
0.3.1 Vue interne structurelle du système 4
0.3.2 Vues externes des sous systèmes 6
0.3.3 Vues internes comportementales des sous systèmes 7
0.3.4 Test des sous systèmes 8
0.4 Conception de niveau $N-2$: la bascule D 11
0.4.1 Vue interne structurelle 11
0.4.2 Vue externe du composant de base 13
0.4.3 Vue interne comportementale du composant de base 13
0.4.4 Test du composant de base 13
0.5 Conception de niveau $N-3$ 14
0.5.1 Vues internes structurelles des composants de base 15
0.5.2 Vues externes des transistors 18
0.5.3 Vues internes des transistors PMos et NMos 19
0.5.4 Test des transistors PMos et NMos 20
0.6 Conclusion 22

De la description d'un système à la description des transistors en VHDL

0.1 Introduction

Ce document a pour objectif de montrer les différents niveaux d'abstraction qu'il est possible de formuler avec un langage de description matériel (en l'occurence VHDL).
Après la présentation du système cible, ce document abordera la modélisation des différents éléments qui le compose.
L'approche de conception sera volontairement très détaillé et, pour chaque étape, nous donnerons les informations suivantes, liées au niveau de conception i :

- une vue externe du système ou sous système sous forme graphique et dans le langage VHDL ;
- une vue interne. Pour cette vue, nous donnerons une description comportementale du système ou sous système.
- un composant de test permettant de valider le niveau de conception et de vérifier que la matérialisation répond bien aux exigences du cahier des charges.

Ensuite, nous passerons à l'étape suivante de la conception, le niveau de conception $i-1$, c'est à dire à la description structurelle du système ou sous système à partir de sous systèmes plus simple reliés entre eux par des signaux ;

0.2 Présentation du système à concevoir - Conception de niveau N

Le système que nous nous proposons de réaliser est un boitier effectuant une transformation parallèle série d'un nombre codé sur 8 Std_Logics.

0.2.1 Vue externe du système

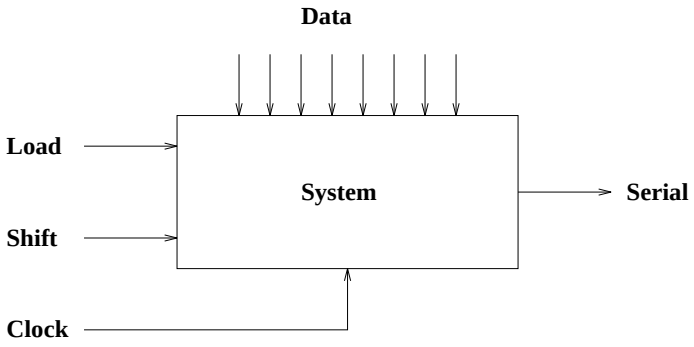


FIG. 0.1 – *Vue externe du système*

Ce système est composé de 3 entrées de type logique (type *Std_Logic*) et de N entrées de type logique, c'est à dire d'un tableau de Std_Logics (dans la description VHDL nous parlerons de vecteurs de

Std_Logics, type *Std_Logic_Vector*). La seule sortie de ce système correspond à la sortie de sérialisation du contenue du registre.
La description externe du système en VHDL est alors la suivante :

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY Systeme IS
        GENERIC (
                Nb : INTEGER );
        PORT (
                Clock : IN Std_Logic ;
                Load : IN Std_Logic ;
                Shift : IN Std_Logic ;
                Data : IN Std_Logic_Vector (Nb-1 DOWNTO 0) ;
                Serial : OUT Std_Logic );
END Systeme;
```

LIST. 1 - *Description de la vue externe de l'entité* Systeme

Notons, dans cette description, que le nombre d'entrées du port *Donnees* est un paramètre générique (Nb). Ce paramètre sera spécifié au moment de l'instanciation du composant.

0.2.2 Vue interne du système

Il s'agit de spécifier le fonctionnement de ce boitier en fonction des actions intervenants sur ses entrées. Pour réaliser cette description, nous allons utiliser un modèle comportemental et la notion de processus dans le langage VHDL.

Extraction des fonctionnalités du système

Le système est décomposé en deux fonctionnalités qui sont :

- une fonction de chargement de la valeur à transférer ;
- une fonction de décalage de la valeur.

A priori, on serait tenté de suivre cette décomposition pour réaliser la description, c'est à dire qu'une description à l'aide de 2 processus semble être une bonne réponse. Le problème auquel on est confronté dans ce cas précis est la manipulation du même signal par les 2 processus. Ce problème peut être résolu mais sa résolution engendre une description complexe. Donc dans ce cas précis, il est préférable de n'utiliser qu'un seul processus pour décrire les valeurs qui seront prises par le signal.

Description VHDL comportementale du système

Le fichier VHDL ci-dessous traduit directement ce que nous venons de dire.

```
ARCHITECTURE Comportementale OF Systeme IS

        SIGNAL Valeur : Std_Logic_Vector (Nb -1 DOWNTO 0)  := (OTHERS => 'Z');

BEGIN
        ProcessLoadShift : PROCESS
                VARIABLE Valeur : Std_Logic_Vector (Nb -1 DOWNTO 0)  := (OTHERS => 'Z');
        BEGIN
                WAIT ON Clock;
                IF (Load = '1' AND Clock='1' AND Clock'event) THEN
```

```
                Valeur := Data;
            END IF;
            IF (Shift = '1' AND Clock='1' AND Clock'event) THEN
                Valeur(Nb-2 DOWNTO 0) := Valeur(Nb-1 DOWNTO 1) ;
                Valeur(Nb-1) := '0';
            END IF;
            Serial <= Valeur(0) AFTER 10 ns;
    END PROCESS ProcessLoadShift;

END Comportementale;
```

LIST. 2 - *Description de l'architecture comportementale du* System

0.2.3 Test du système

Il s'agit de s'assurer que le système décrit permet bien de réaliser la fonctionnalité désirée. Pour cela nous allons effectuer une simulation du système. Cette simulation est assurée par un simulateur VHDL qui gèrera les événements des signaux d'entrées et affichera les niveaux (ou valeurs) des signaux de sortie.
Pour réaliser cette simulation, nous décrivons un composant de test. Pour tout composant à tester, nous procèdons de la même manière, c'est à dire que nous décrivons un composant (une entité) n'ayant ni *generic* ni *port*s d'entrées. Ce composant réalise une instanciation du composant à tester puis gère, sous la forme d'un processus, les niveaux (ou valeurs) des signaux d'entrées. L'avantage du composant de test par rapport à une simulation par fichier script du simulateur, est qu'il peut être repris sur n'importe quelle plate forme VHDL, ce qui n'est pas le cas du fichier script qui dépend étroitement du simulateur choisit.
Nous donnons ci-dessous la vue externe du fichier de test ainsi que son architecture alliant à la fois le comportemental et le structurel.

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestSysteme IS
END TestSysteme;

ARCHITECTURE test OF TestSysteme IS

    COMPONENT Systeme
        GENERIC (
            Nb : INTEGER );
        PORT (
            Clock : IN Std_Logic ;
            Load : IN Std_Logic ;
            Shift : IN Std_Logic ;
            Data : IN Std_Logic_Vector (Nb-1 DOWNTO 0) ;
            Serial : OUT Std_Logic );
    END COMPONENT;

    SIGNAL s_clock : Std_Logic := '0';
    SIGNAL s_load : Std_Logic := '0' ;
    SIGNAL s_shift : Std_Logic := '0' ;
    SIGNAL s_serial : Std_Logic := 'Z';
    SIGNAL s_D : Std_Logic_Vector (15 DOWNTO 0) := "0101010011001100";

    FOR Syst : Systeme USE ENTITY work.Systeme(Comportementale);
--  FOR Syst : Systeme USE ENTITY work.Systeme(Structurelle);
```

```
BEGIN
        Syst : Systeme
                GENERIC MAP (16)
                PORT MAP (s_clock, s_load, s_shift, s_D, s_serial);

        ProcessSimulation : PROCESS
                BEGIN
                        WAIT FOR 120 ns;
                        s_load  <= '1' ;

                        WAIT FOR 100 ns;        -- Attention, il ne faut pas de front
                        s_load  <= '0' ;        -- montant d'horloge entre le passage
                        s_shift  <= '1' ;        -- a 0 de load et le passage a 1 de shift
                                                -- Sinon on charge un zero dans le registre
                                                -- a decalage
                        WAIT FOR 1 ms;
                        s_shift <= '0';

        END PROCESS ProcessSimulation;

        ProcessHorloge : PROCESS
                BEGIN
                        s_clock <= '0' ;
                        WAIT FOR 50 ns;
                        s_clock <= '1' ;
                        WAIT FOR 50 ns;
        END PROCESS ProcessHorloge;

END test;
```

LIST. 3 - *Description de l'entité de test* TestSysteme

0.3 Conception de niveau $N-1$

Il s'agit, maintenant que nous avons réalisé la description comportementale du système, de nous intéresser à la matérialisation de celui-ci.
Comme nous l'avons dit précédemment, nous devons assurer le chargement de la valeur à décaler ainsi que le décalage de celle-ci. La première fonctionnalité nous amène à mettre en place des bascules afin de mémoriser la valeur à transmettre. Ensuite, un certain nombre de blocs logiques (portes logiques élémentaires) nous sont nécessaires pour assurer à la fois le chargement et le décalage (en fonction de l'état des signaux *Load* et *Shift*).

0.3.1 Vue interne structurelle du système

Le schéma que nous proposons est donné sur la figure 0.2.

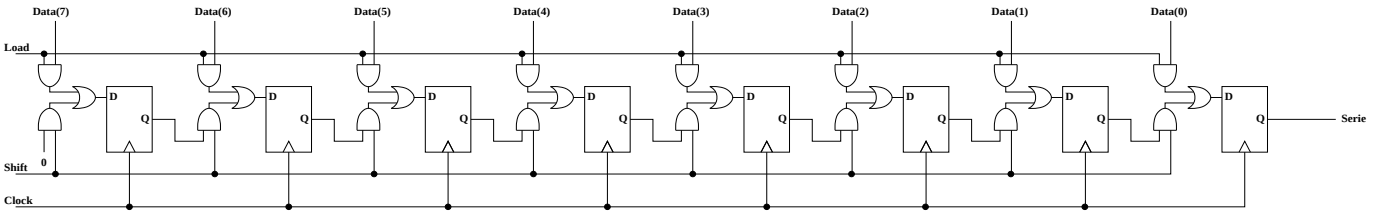


FIG. 0.2 – *Architecture du système*

Nous voyons alors que ce schéma fait appel à trois types de composant :

- bascule D ;
- porte ET ;
- porte OU.

Nous allons alors décrire le composant *Systeme* comme étant une structure de ces composants de base. Il faudra ensuite nous assurer que les composants de base sont disponibles dans la bibliothèque de travail.

Étant donné la structure très régulière de ce schéma, nous avons tout intérêt à utiliser l'instruction *generate*. Cette instruction permet, à l'aide de "boucles" *for*, de limiter l'écriture de code VHDL et de rendre cette description générique par rapport à un ou plusieurs paramètres.
Nous donnons sur la figure 0.3 le schéma du bloc élémentaire se répétant Nb fois.

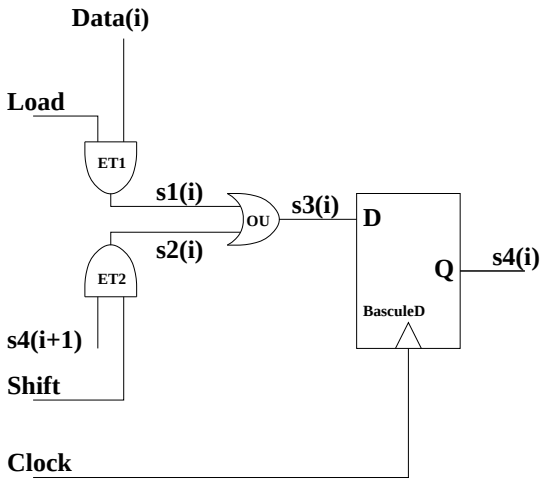


FIG. 0.3 – *Architecture d'un bloc du système*

```
ARCHITECTURE Structurelle OF Systeme IS

        COMPONENT BasculeD
                GENERIC (
                        TBasc : TIME := 10 ns);
                PORT (
                        D : IN Std_Logic ;
                        C : IN Std_Logic ;
                        Q : OUT Std_Logic );
        END COMPONENT;

        COMPONENT PorteET
                GENERIC (
                        TEt : TIME := 2 ns);
                PORT (
                        entree1 : IN Std_Logic ;
                        entree2 : IN Std_Logic ;
                        sortie : OUT Std_Logic );
        END COMPONENT;

        COMPONENT PorteOU
                GENERIC (
                        TOu : TIME := 2 ns);
```

```
                PORT (
                        entree1 : IN Std_Logic ;
                        entree2 : IN Std_Logic ;
                        sortie : OUT Std_Logic );
        END COMPONENT;

        SIGNAL s1 : Std_Logic_Vector (Nb-1 DOWNTO 0) ;
        SIGNAL s2 : Std_Logic_Vector (Nb-1 DOWNTO 0) ;
        SIGNAL s3 : Std_Logic_Vector (Nb-1 DOWNTO 0) ;
        SIGNAL s4 : Std_Logic_Vector (Nb DOWNTO 0) ;

BEGIN
        bloc : FOR i IN 0 TO Nb-1 GENERATE
                ETi : PorteET
                        PORT MAP (Load, Data(i), s1(i) );

                ETj : PorteET
                        PORT MAP (s4(i+1), Shift, s2(i) ) ;

                OUi : PorteOU
                        PORT MAP (s1(i),s2(i), s3(i) );

                BasculeDi : BasculeD
                        PORT MAP (s3(i), Clock, s4(i) );

        END GENERATE bloc;

        Serial <= s4(0);
        s4(Nb) <= '0';
END Structurelle ;
```

LIST. 4 - *Description de l'architecture structurelle de l'entité* Systeme

Notons la compacité de l'écriture de l'ensemble du schéma grâce à l'utilisation de l'instruction *generate*. Par rapport à une description "à plat" — où tout aurait été décrit — cette description à le gros avantage d'être générique par rapport au nombre de Std_Logic du vecteur d'entrée.
Comme nous l'avons dit précédemment, cette description structurelle s'appuie sur des composants de base, il faut donc s'assurer, pour pouvoir effectuer des simulations, que la bibliothèque de travail possède ces composants.
Supposons que ces composants n'existent pas en bibliothèque, il faut donc leur donner un modèle de fonctionnement. Pour cela on utilisera un modèle comportemental. Puis on s'attachera, par la suite, à réaliser ces composants à l'aide de composants encore plus simples.

0.3.2 Vues externes des sous systèmes

Chaque composant utilisé doit donc être décrit par sa vue externe (voir listing 5).

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY BasculeD IS
        GENERIC (
                TBasc : TIME := 10 ns);
        PORT (
                D : IN Std_Logic;
                C : IN Std_Logic ;
```

```
                Q : OUT Std_Logic );
END BasculeD;

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY PorteET IS
        GENERIC (
                TEt : TIME := 2 ns);
        PORT (
                entree1 : IN Std_Logic ;
                entree2 : IN Std_Logic ;
                sortie : OUT Std_Logic );
END PorteET;

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY PorteOU IS
        GENERIC (
                TOu : TIME := 2 ns);
        PORT (
                entree1 : IN Std_Logic ;
                entree2 : IN Std_Logic ;
                sortie : OUT Std_Logic);
END PorteOU;
```

LIST. 5 - *Description des entités des composants de base*

Ensuite chacun doit être associé à une description de fonctionnement. Nous la réalisons dans un premier temps (à des fins de simulation du système global) à l'aide de processus.

0.3.3 Vues internes comportementales des sous systèmes

```
ARCHITECTURE Comportementale OF BasculeD IS
BEGIN
        ProcessBasculeD : PROCESS
        BEGIN
                IF C = '1' AND C'event THEN
                        Q <= D AFTER TBasc;
                END IF;
                WAIT ON D, C;
        END PROCESS ProcessBasculeD;

END Comportementale ;


ARCHITECTURE Comportementale OF PorteET IS
BEGIN
        ProcessET : PROCESS
        BEGIN
                sortie <= entree1 AND entree2 AFTER TEt;
                WAIT ON entree1, entree2;
        END PROCESS ProcessET;
END Comportementale;
```

```
ARCHITECTURE Comportementale OF PorteOU IS
BEGIN
        ProcessOU : PROCESS
        BEGIN
                sortie <= entree1 OR entree2 AFTER TOu;
                WAIT ON entree1, entree2;
        END PROCESS ProcessOU;
END Comportementale;
```

LIST. 6 - *Description des architectures des composants de base*

0.3.4 Test des sous systèmes

Évidemment, comme nous venons de décrire de nouveaux composants, nous devons en réaliser les tests.

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestOU IS
END TestOU;

ARCHITECTURE Test OF TestOU IS

        COMPONENT PorteOU
                GENERIC (
                        TOu : TIME :=2 ns);
                PORT (
                        entree1 : IN Std_Logic;
                        entree2 :  IN Std_Logic ;
                        sortie : OUT Std_Logic );
        END COMPONENT;

        SIGNAL s_in1 : Std_Logic := '0' ;
        SIGNAL s_in2 : Std_Logic := '0' ;
        SIGNAL s_out : Std_Logic := 'Z' ;

--      FOR OU : PorteOU USE ENTITY work.PorteOU(Comportementale);
        FOR OU : PorteOU USE ENTITY work.PorteOU(Structurelle);

        SUBTYPE TabStimuli IS std_logic_vector(0 TO 2) ;

        TYPE TabTestVecteur IS ARRAY (INTEGER RANGE <>) OF TabStimuli;

        CONSTANT TV : TabTestVecteur := (
                ('0', '0', '0'),
                ('0', '1', '0'),
                ('1', '0', '1'),
                ('1', '1', '1') );

BEGIN
        OU  : PorteOU
                GENERIC MAP (5 ns)
                PORT MAP (s_in1, s_in2, s_out);

        ProcessSimulation : PROCESS
        BEGIN
```

```
                WAIT FOR 10 ns;
                FOR i in TV'RANGE LOOP
                        s_in1 <= TV(i)(0);
                        s_in2 <= TV(i)(1);
                        WAIT FOR 100 ns;
                        ASSERT (s_out = TV(i)(2))
                                REPORT "Erreur dans la simulation "
                                SEVERITY ERROR;
                END LOOP;
                WAIT;

        END PROCESS ProcessSimulation;
END Test ;
```

LIST. 7 - *Description de l'entité de test de la porte OU*

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestET IS
END TestET;

ARCHITECTURE Test OF TestET IS

        COMPONENT PorteET
                GENERIC (
                        TEt : TIME :=2 ns);
                PORT (
                        entree1 : IN Std_Logic;
                        entree2 :  IN Std_Logic ;
                        sortie : OUT Std_Logic );
        END COMPONENT;

        SIGNAL s_in1 : Std_Logic := '0' ;
        SIGNAL s_in2 : Std_Logic := '0' ;
        SIGNAL s_out : Std_Logic := 'Z' ;

--      FOR ET : PorteET USE ENTITY work.PorteET(Comportementale);
        FOR ET : PorteET USE ENTITY work.PorteET(Structurelle);

        SUBTYPE TabStimuli IS std_logic_vector(0 TO 2) ;

        TYPE TabTestVecteur IS ARRAY (INTEGER RANGE <>) OF TabStimuli;

        CONSTANT TV : TabTestVecteur := (
                ('0', '0', '0'),
                ('0', '1', '0'),
                ('1', '0', '0'),
                ('1', '1', '1') );

BEGIN
        ET  : PorteET
                GENERIC MAP (5 ns)
                PORT MAP (s_in1, s_in2, s_out);

        ProcessSimulation : PROCESS
        BEGIN
                WAIT FOR 10 ns;
```

```
                FOR i in TV'RANGE LOOP
                        s_in1 <= TV(i)(0);
                        s_in2 <= TV(i)(1);
                        WAIT FOR 100 ns;
                        ASSERT (s_out = TV(i)(2))
                                REPORT "Erreur dans la simulation "
                                SEVERITY ERROR;
                END LOOP;
                WAIT;

        END PROCESS ProcessSimulation;
END Test ;
```

LIST. 8 - *Description de l'entité de test de la porte ET*

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestBasculeD IS
END TestBasculeD;

ARCHITECTURE Test OF TestBasculeD IS

        COMPONENT BasculeD
                GENERIC (
                        TBasc : TIME := 10 ns);
                PORT (
                        D : IN Std_Logic;
                        C : IN Std_Logic ;
                        Q : OUT Std_Logic );
        END COMPONENT;

        SIGNAL s_d : Std_Logic := '0' ;
        SIGNAL s_clock : Std_Logic := '0' ;
        SIGNAL s_q : Std_Logic := 'Z' ;

--        FOR Bascule : BasculeD USE ENTITY work.BasculeD(Comportementale);
        FOR Bascule : BasculeD USE ENTITY work.BasculeD(Structurelle);

        SUBTYPE TabStimuli IS std_logic_vector(0 TO 2) ;

        TYPE TabTestVecteur IS ARRAY (INTEGER RANGE <>) OF TabStimuli;

        CONSTANT TV : TabTestVecteur := (
                ('0', '0'),
                ('1', '1'),
                ('0', '0') );

BEGIN
        Bascule  : BasculeD
                GENERIC MAP (15 ns)
                PORT MAP (s_d, s_clock, s_q);

        ProcessSimulation : PROCESS
        BEGIN
                WAIT FOR 10 ns;
                FOR i in TV'RANGE LOOP
```

```
                        s_d <= TV(i)(0);
                        WAIT FOR 100 ns;
                        ASSERT (s_q = TV(i)(1))
                                REPORT "Erreur dans la simulation "
                                SEVERITY ERROR;
                END LOOP;
                WAIT;
        END PROCESS ProcessSimulation;

        ProcessHorloge : PROCESS
                BEGIN
                        s_clock <= '0' ;
                        WAIT FOR 50 ns;
                        s_clock <= '1' ;
                        WAIT FOR 50 ns;
        END PROCESS ProcessHorloge;

END Test ;
```

LIST. 9 - *Description de l'entité de test de la bascule D*

0.4 Conception de niveau $N-2$: la bascule D

Il s'agit maintenant de décrire les différents éléments à partir de composants encore plus simples. Dans notre bibliothèque de travail, nous distinguons deux types de composants :

- des composants de type logique combinatoire (porte ET et OU) ;
- un composant de type séquentiel (dont l'état présent dépend d'un état précédent, bascule D).

Par expérience, nous savons que les portes ET et OU peuvent être décrite directement à partir de transistors. Ceci n'est pas le cas de la bascule D qui doit, d'abord être décrite à l'aide de fonctions logiques de base.
Ceci explique pourquoi nous ne nous intéressons qu'à la bascule D dans ce paragraphe. Les autres composants dont nous avons besoin seront "matérialisés" dans la phase suivante de conception, de cette façon on ne parlera pas de transistors à ce niveau de conception.
Nous donnons ci-dessous le schéma d'une bascule D (voir figure 0.4).

0.4.1 Vue interne structurelle

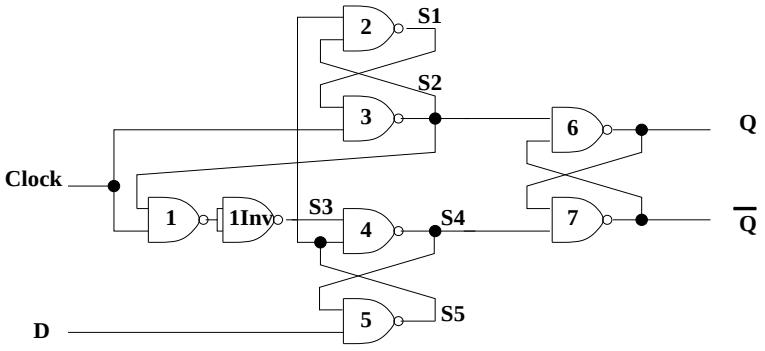


FIG. 0.4 – *Schéma d'une bascule D*

La description VHDL est alors la suivante :

```
ARCHITECTURE Structurelle OF BasculeD IS

        COMPONENT PorteNAND
                GENERIC (
                        TNand : TIME := 5 ns);
                PORT (
                        entree1 : Std_Logic ;
                        entree2 : Std_Logic ;
                        sortie : OUT Std_Logic );
        END COMPONENT;

        SIGNAL s1 : Std_Logic := 'Z' ;
        SIGNAL s2 : Std_Logic := 'Z' ;
        SIGNAL s3 : Std_Logic := 'Z' ;
        SIGNAL s3inv : Std_Logic := 'Z' ;
        SIGNAL s4 : Std_Logic := 'Z' ;
        SIGNAL s5 : Std_Logic := 'Z' ;
        SIGNAL s_sortie : Std_Logic := 'Z' ;
        SIGNAL s_sortiebarre : Std_Logic := 'Z' ;

BEGIN
        NAND1 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s2, C, s3inv);

        NAND1INV : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s3inv, s3inv, s3);

        NAND2 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s5, s2, s1);

        NAND3 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s1, C, s2);

        NAND4 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s3, s5, s4);

        NAND5 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s4, D, s5);

        NAND6 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s2, s_sortiebarre, s_sortie);

        NAND7 : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s4, s_sortie, s_sortiebarre);

        Q <= s_sortie;

END Structurelle;
```

LIST. 10 - *Description de l'architecture d'une bascule D*

Compte tenu de cette description, notre bibliothèque doit alors maintenant contenir les composants suivants :

- porte NAND ;
- porte ET ;
- porte OU.

0.4.2 Vue externe du composant de base

Ici les composants de base se limitent à la nouvelle fonction logique élémentaire nécessaire, c'est à dire à la porte NAND.

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY PorteNAND IS
        GENERIC (
                TNand  : TIME := 5 ns);
        PORT (
                entree1 : IN Std_Logic ;
                entree2 : IN Std_Logic ;
                sortie : OUT Std_Logic);
END PorteNAND ;
```

LIST. 11 - *Description de l'entité* PorteNand

0.4.3 Vue interne comportementale du composant de base

```
ARCHITECTURE Comportementale OF PorteNAND IS
BEGIN
        ProcessPorteNAND  : PROCESS
        BEGIN
                sortie <= entree1 NAND entree2 AFTER TNand;
                WAIT ON entree1, entree2;
        END PROCESS ProcessPorteNAND  ;
END Comportementale;
```

LIST. 12 - *Description de l'architecture de la porte Nand*

0.4.4 Test du composant de base

Il s'agit donc de tester le bon fonctionnement de cette porte.

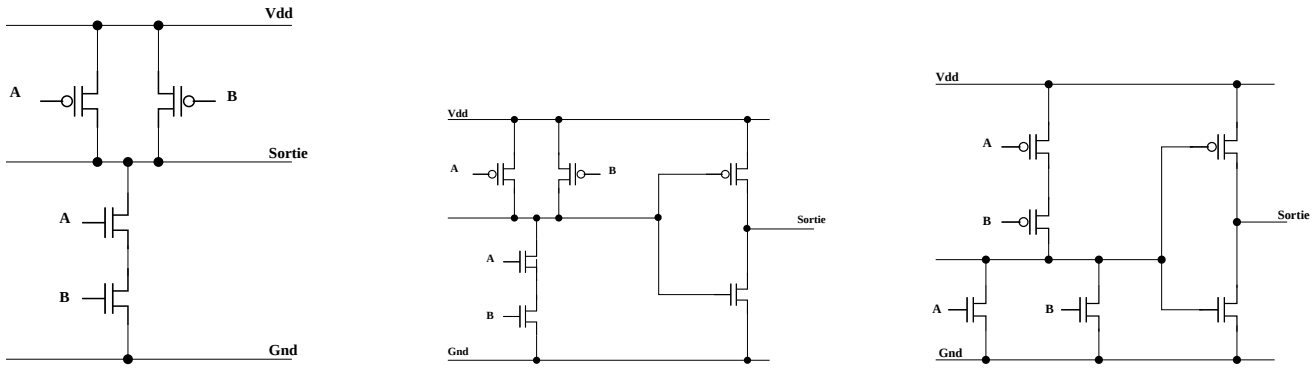
```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestNAND IS
END TestNAND ;

ARCHITECTURE Test OF TestNAND IS

        COMPONENT PorteNAND
                GENERIC (
```

```
                        TNand : TIME :=5 ns);
                PORT (
                        entree1 : IN Std_Logic;
                        entree2 :  IN Std_Logic ;
                        sortie : OUT Std_Logic );
        END COMPONENT;

        SIGNAL s_in1 : Std_Logic := '0' ;
        SIGNAL s_in2 : Std_Logic := '0' ;
        SIGNAL s_out : Std_Logic := 'Z' ;


--      FOR NET : PorteNAND USE ENTITY work.PorteNAND(Comportementale);
        FOR NET : PorteNAND USE ENTITY work.PorteNAND(Structurelle);



        SUBTYPE TabStimuli IS std_logic_vector(0 TO 2) ;

        TYPE TabTestVecteur IS ARRAY (INTEGER RANGE <>) OF TabStimuli;

        CONSTANT TV : TabTestVecteur := (
                ('0', '0', '1'),
                ('0', '1', '1'),
                ('1', '0', '1'),
                ('1', '1', '0') );

BEGIN
        NET  : PorteNAND
                GENERIC MAP (5 ns)
                PORT MAP (s_in1, s_in2, s_out);

        ProcessSimulation : PROCESS
        BEGIN
                WAIT FOR 10 ns;
                FOR i in TV'RANGE LOOP
                        s_in1 <= TV(i)(0);
                        s_in2 <= TV(i)(1);
                        WAIT FOR 100 ns;
                        ASSERT (s_out = TV(i)(2))
                                REPORT "Erreur dans la simulation "
                                SEVERITY ERROR;
                END LOOP;
                WAIT;

        END PROCESS ProcessSimulation;
END Test ;
```

LIST. 13 - *Description de l'entité de test* TestPorteNAND

0.5 Conception de niveau $N-3$

Si l'on imagine que ces composants de base n'existent pas en bibliothèque, il faut les réaliser. Ces composants sont alors décris à l'aide de transistors comme les schémas de la figure 0.5 le montre.



a) Porte NAND b) Porte ET c) Porte OU

FIG. 0.5 – *Schémas des portes logiques de base*

La structure de l'ensemble de ces composants est basée sur des transistors *PMos* et *NMos*. Nous devons alors compléter notre bibliothèque à l'aide des descriptions de ces transistors.

0.5.1 Vues internes structurelles des composants de base

Ici, nous décrivons les fonctions logiques de base comme étant des structures de transistors *PMos* et *NMos*.

```
ARCHITECTURE Structurelle OF PorteET IS

        COMPONENT PMOS
                GENERIC (
                        TPMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : IN Std_Logic ;
                        Drain : INOUT Std_Logic := 'Z');
        END COMPONENT ;

        COMPONENT NMOS
                GENERIC (
                        TNMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : INOUT Std_Logic := 'Z' ;
                        Drain : IN Std_Logic);
        END COMPONENT ;

        SIGNAL Vdd : Std_Logic := '1';
        SIGNAL Gnd : Std_Logic := '0';
        SIGNAL s1 : Std_Logic := 'Z' ;
        SIGNAL s2 : Std_Logic := 'Z' ;
        SIGNAL s3 : Std_Logic := 'Z' ;
        SIGNAL s_sortie : Std_Logic := 'Z' ;

BEGIN
        T1 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree1, Vdd, s1);
```

```
        T2 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree2, Vdd, s1);

        T3 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree1, s1, s2);

        T4 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree2, s2, Gnd);

        T5 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (s1, Vdd, s_sortie);

        T6 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (s1, s_sortie, Gnd);

        sortie <= s_sortie;

END Structurelle;


ARCHITECTURE Structurelle OF PorteOU IS

        COMPONENT PMOS
                GENERIC (
                        TPMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : IN Std_Logic ;
                        Drain : INOUT Std_Logic := 'Z');
        END COMPONENT ;

        COMPONENT NMOS
                GENERIC (
                        TNMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : INOUT Std_Logic := 'Z';
                        Drain : IN Std_Logic );
        END COMPONENT ;

        SIGNAL Vdd : Std_Logic := '1';
        SIGNAL Gnd : Std_Logic := '0';
        SIGNAL s1 : Std_Logic := 'Z' ;
        SIGNAL s2 : Std_Logic := 'Z' ;
        SIGNAL s3 : Std_Logic := 'Z' ;
        SIGNAL s_sortie : Std_Logic := 'Z' ;

BEGIN
        T1 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree1, Vdd, s1);

        T2 : PMOS
                GENERIC MAP (1 ns)
```

```
                PORT MAP (entree2, s1, s2);

        T3 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree1, s2, Gnd);

        T4 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree2, s2, Gnd);

        T5 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (s2, Vdd, s_sortie);

        T6 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (s2, s_sortie, Gnd);

        sortie <= s_sortie;

END Structurelle;


ARCHITECTURE Structurelle OF PorteNAND IS

        COMPONENT PMOS
                GENERIC (
                        TPMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : IN Std_Logic ;
                        Drain : INOUT Std_Logic  := 'Z');
        END COMPONENT ;

        COMPONENT NMOS
                GENERIC (
                        TNMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : INOUT Std_Logic  := 'Z';
                        Drain : IN Std_Logic );
        END COMPONENT ;

        SIGNAL Vdd : Std_Logic := '1';
        SIGNAL Gnd : Std_Logic := '0';
        SIGNAL s1 : Std_Logic := 'Z';
        SIGNAL s_sortie : Std_Logic ;

BEGIN
        T1 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree1, Vdd, s_sortie);

        T2 : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree2, Vdd, s_sortie);

        T3 : NMOS
                GENERIC MAP (1 ns)
```

```
                PORT MAP (entree1, s_sortie, s1);

        T4 : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (entree2, s1, Gnd);

        sortie <= s_sortie;

END Structurelle;
```

LIST. 14 - *Description de l'architecture des portes logiques*

0.5.2 Vues externes des transistors

a) Transistor PMos b) Transistor NMos

FIG. 0.6 – *Vues externes des transistors*

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE work.TransPack.ALL;

ENTITY PMOS IS
        GENERIC (
                TPMOS : TIME := 1 ns );
        PORT (
                Grille : IN Std_Logic ;
                Source : IN Std_Logic ;
                Drain : INOUT Std_Logic := 'Z');
END PMOS ;

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE work.TransPack.ALL;

ENTITY NMOS IS
        GENERIC (
                TNMOS : TIME := 1 ns );
        PORT (
                Grille : IN Std_Logic ;
                Source : IN Std_Logic ;
                Drain : INOUT Std_Logic := 'Z' );
END NMOS ;
```

LIST. 15 - *Description des entités* PMos *et* NMos

0.5.3 Vues internes des transistors PMos et NMos

Le fonctionnement des transistors est donné comme suit :

- un transistor *PMos* est passant lorsque sa grille est à l'état '0' ; dans ce cas le *Drain* et la *Source* sont au même potentiel. Dans la description comportementale du fonctionnement d'un tel transistor, nous dirons que le *Drain* prend l'état de la *Source*. En effet, la *Source* d'un transistor *PMos* étant toujours connectée (plus ou moins directement) au potentiel *Vdd*, l'état passant permet d'amener ce potentiel au niveau du *Drain*.

- un transistor *NMos* est passant lorsque sa grille est à l'état '1' ; dans ce cas le *Drain* et la *Source* sont au même potentiel. Dans la description comportementale du fonctionnement d'un tel transistor, nous dirons que le *Drain* prend l'état de la *Source*. En effet, la *Source* d'un transistor *NMos* étant toujours connectée (plus ou moins directement) au potentiel *GND*, l'état passant permet d'amener ce potentiel au niveau du *Drain*.

```
ARCHITECTURE Comportementale OF PMOS IS

BEGIN
        PROCESS
        BEGIN
                IF Grille ='0' THEN
                        IF         (EtatHaut(Source) AND EtatBas(Drain))
                          OR         (EtatBas(Source) AND EtatHaut(Drain))  THEN
                                ASSERT (false) REPORT "Court circuit dans un PMOS !!!" SEVERITY NOTE;
                        END IF;
                        Drain <= Source AFTER TPMOS;
                ELSE
                        Drain <= 'Z' AFTER TPMOS;
                END IF;
                WAIT ON Grille, Source, Drain;
        END PROCESS;

END Comportementale;

ARCHITECTURE Comportementale OF NMOS IS

BEGIN
        PROCESS
        BEGIN
                IF Grille ='1' THEN
                        IF         (EtatHaut(Source) AND EtatBas(Drain))
                          OR         (EtatBas(Source) AND EtatHaut(Drain))  THEN
                                ASSERT (false) REPORT "Court circuit dans un NMOS !!!" SEVERITY NOTE;
                       END IF;
                        Drain <= Source AFTER TNMOS;
                ELSE
                        Drain <= 'Z' AFTER TNMOS;
                END IF;
                WAIT ON Grille, Source, Drain;
        END PROCESS;

END Comportementale;
```

LIST. 16 - *Descriptions des architectures des transistors* PMos *et* NMos

Notez l'utilisation de l'instructions *Assert* qui permettent d'indiquer les instants ou un court circuit se forme entre le potentiel *Vdd* et *GND*.

Notons dans ce listing la présence des appels à deux fonctions qui permettent de tester si un bit est à l'état haut ou à l'état bas. Ces deux fonctions sont mises en place dans un package dont la description vous est donnée ci-dessous.

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

PACKAGE TransPack IS

        FUNCTION EtatHaut(A : IN Std_Logic) RETURN BOOLEAN;
        FUNCTION EtatBas(A : IN Std_Logic) RETURN BOOLEAN;

END TransPack;

PACKAGE BODY TransPack IS

        FUNCTION EtatHaut(A : IN Std_Logic) RETURN BOOLEAN IS
                VARIABLE result : BOOLEAN := false;
        BEGIN
                IF (A = '1' OR A = 'H') THEN
                        result := true;
                END IF;
                RETURN result;
        END EtatHaut;

        FUNCTION EtatBas(A : IN Std_Logic) RETURN BOOLEAN IS
                VARIABLE result : BOOLEAN := false;
        BEGIN
                IF (A = '0' OR A = 'L') THEN
                        result := true;
                END IF;
                RETURN result;
        END EtatBas;

END TransPack;
```

LIST. 17 - *Descriptions des architectures des transistors* PMos *et* NMos

0.5.4 Test des transistors PMos et NMos

```
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestPMOS IS
END TestPMOS;

ARCHITECTURE Test OF TestPMOS IS

        COMPONENT PMOS
                GENERIC (
                        TPMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : IN Std_Logic ;
```

```
                Drain : INOUT Std_Logic := 'Z');
        END COMPONENT;

        SIGNAL s_g : Std_Logic := '0';
        SIGNAL s_d : Std_Logic := '0' ;
        SIGNAL s_s : Std_Logic := '0' ;

        SUBTYPE TabStimuli IS std_logic_vector(0 TO 2) ;

        TYPE TabTestVecteur IS ARRAY (INTEGER RANGE <>) OF TabStimuli;


        CONSTANT TV : TabTestVecteur := (
                ('0', '0', '0'),
                ('0', '1', '1'),
                ('1', '0', 'Z'),
                ('1', '1', 'Z'),
                ('0', '1', '1') );

BEGIN
        T : PMOS
                GENERIC MAP (1 ns)
                PORT MAP (s_g, s_s, s_d);

        ProcessSimulation : PROCESS
        BEGIN
                WAIT FOR 10 ns;
                FOR i in TV'RANGE LOOP
                        s_g <= TV(i)(0);
                        s_s <= TV(i)(1);
                        WAIT FOR 100 ns;
                        ASSERT (s_d = TV(i)(2))
                                REPORT "Erreur dans la simulation "
                                SEVERITY ERROR;
                END LOOP;
                WAIT;

        END PROCESS ProcessSimulation;
END Test;

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY TestNMOS IS
END TestNMOS;

ARCHITECTURE Test OF TestNMOS IS

        COMPONENT NMOS
                GENERIC (
                        TNMOS : TIME := 1 ns );
                PORT (
                        Grille : IN Std_Logic ;
                        Source : INOUT Std_Logic := 'Z';
                        Drain : IN Std_Logic );
        END COMPONENT;

        SIGNAL s_g : Std_Logic := '0';
        SIGNAL s_d : Std_Logic := '0' ;
```

```
        SIGNAL s_s : Std_Logic := '0';

        SUBTYPE TabStimuli IS std_logic_vector(0 TO 2) ;

        TYPE TabTestVecteur IS ARRAY (INTEGER RANGE <>) OF TabStimuli;


        CONSTANT TV : TabTestVecteur := (
                ('0', '0', 'Z'),
                ('0', '1', 'Z'),
                ('1', '0', '0'),
                ('1', '1', '1'),
                ('0', '1', 'Z') );

BEGIN
        T : NMOS
                GENERIC MAP (1 ns)
                PORT MAP (s_g, s_s, s_d);

        ProcessSimulation : PROCESS
        BEGIN
                WAIT FOR 10 ns;
                FOR i in TV'RANGE LOOP
                        s_g <= TV(i)(0);
                        s_s <= TV(i)(1);
                        WAIT FOR 100 ns;
                        ASSERT (s_d = TV(i)(2))
                                REPORT "Erreur dans la simulation "
                                SEVERITY ERROR;
                END LOOP;
                WAIT;

        END PROCESS ProcessSimulation;
END Test;
```

LIST. 18 - *Description des entités de test* TestPMos *et* TestNMos

0.6 Conclusion

Dans ce document, nous avons décrit, à l'aide d'un exemple relativement simple, une méthodologie de conception que l'on appelle descendante (ou *top bottom*). Cette méthodologie est particulièrement bien adaptée aux problèmes que l'on souhaite résoudre de façon optimale, ce qui est le cas lorsque l'on désire réaliser un ASIC (Applied Specific Intergrated Circuit).

Le système est, dans un premier temps, abordé de façon comportementale afin de s'assurer de la bonne compréhension du cahier des charges.
Après validation du système, on débute la conception. Cette conception est réalisée par raffinement successif depuis la description comportementale du système jusqu'au niveau de la bibliothèque disponible. Pour cet exemple, nous avons volontairement posé comme hypothèse que nous ne disposions d'aucune bibliothèque et d'aucun outil informatique permettant de prendre en charge tout ou partie de la description.
Notre démarche de conception nous a donc amenée à envisager la description d'éléments de très bas niveau (transistors). Évidemment, dans la plupart des cas, vous n'aurz jamais à descendre aussi loin dans la description, les bibliothèques disponible vous permettront de vous arrêter aux niveaux logique,

fonctions, composants (selon les cas).

Ce document propose une démarche très méthodologique, puisque pour chaque système, nous avons décomposé notre approche par une description comportementale tout d'abord, puis par une description structurelle ensuite.
À chaque fois, nous avons réalisé les composants de test des entitées. Et avant de nous lancer dans la conception d'une entitée nous avons d'abord validée celle-ci de façon comportementale. Après conception, nous avons évidemment validé la nouvelle architecture.

Cette approche descendant, par raffinement successifs, nous permet d'avancer progressivement vers la solution finale, tout en ayant le minimum de remise en cause de ce qui à déjà été réalisé.