

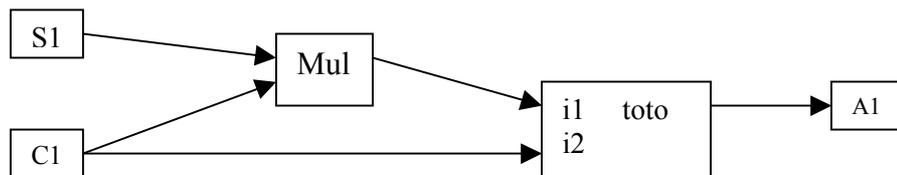
TUTORIAL SUR L'UTILISATION DE SynDEX-IC POUR LA SIMULATION

Comme nous l'avons vu dans les principes, **SynDEX-Ic** permet la génération automatique de code VHDL synthétisable, ce qui permet de couvrir aussi bien la conception de circuits dédiés (ASIC) que d'architectures à base de composants reconfigurables (FPGA).

Cette partie consiste à expliquer toutes les étapes qui permettent de passer d'une spécification algorithmique à la génération automatique de son implantation matérielle sous la forme d'un code VHDL synthétisable et simulable à l'aide d'outil de CAO. L'algorithme de l'application est spécifié sous forme d'un graphe de dépendance de données.

1- Les différentes étapes pour utiliser SynDEX-Ic :

Nous allons construire un circuit simple que nous traiterons en guise d'exemple afin de faciliter l'utilisation de l'outil SynDEX-Ic :



Avec :

S1 : capteur que l'on appellera "*input1*" et qui retourne un int sur 8 bits

C1 : Constant que l'on appellera "*constante*" et qui retourne un int sur 8 bits (la valeur de la constante est 2)

Mul : Opération qui fait une multiplication entre 2 int, elle retourne un int sur 8 bits

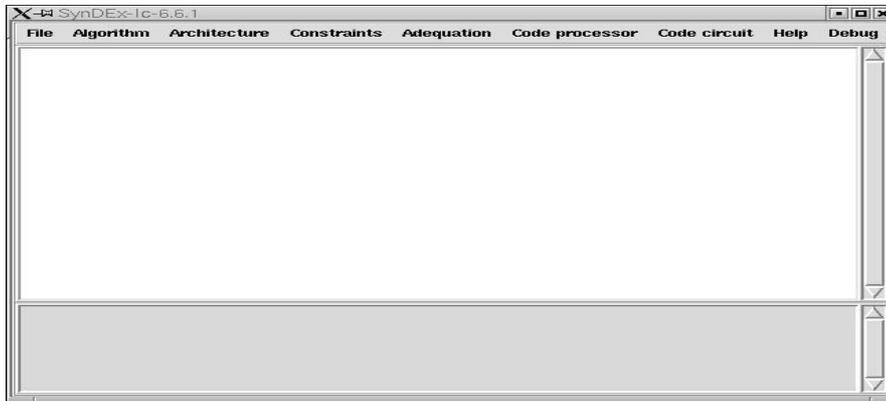
A1 : Actionneur que l'on appellera "*output*" et qui récupère un int sur 8 bits

Toto : Opération quelconque par exemple un composant dont la sortie est i1 puissance i2

Tous les fichiers nécessaires à la spécification et à l'implantation de cette application se trouvent dans le répertoire tutorial-synindex-ic

1-1 Lancement de l'outil SynDEX-Ic

Le fichier exécutable de SynDEX-IC s'appelle : SynDex-6.6.1
Voici la fenêtre principale de SynDEX-Ic, elle est identique à celle de SynDEX à la différence que nous avons un menu *Code circuit* :



1-2 brève présentation de SynDEX-Ic

SynDEX-Ic possède des définitions standard : ADD, MUL, SUB, WINDOW, INPUT, OUTPUT, ... Cependant, il faudra donner le code VHDL des composants INPUT (sensor) quelque soit l'application, celui des OUTPUT s'ils ont plus d'une entrée Data, celui des CONSTANT s'ils ont plus d'une sortie Data et celui des MEMORY s'ils ont plus d'une entrée Data et plus d'une sortie Data. C'est ainsi parce que ces composants ne sont pas définis dans la librairie VHDL de SynDEX-Ic vu que le constructeur ne connaît pas la description comportementale de ces composants.

1-3 Création des définitions

Ainsi, il faut créer les "définitions" non standard, c'est à dire des boites correspondant à des composants VHDL qui seront utilisés pour construire l'architecture et dont la "définition" n'existe pas sous SynDEX-Ic. Donc dans notre exemple, étant donné que toto n'est pas standard, il faudra créer la "définition" de "toto" que l'on appellera def_toto :

→ Création de la "définition" de "toto" (def_toto) :

- lancer *New Local Definition* dans le menu *Algorithm*
- choisir le type de la définition (dans notre exemple, ce sera *Function*)
- donner le nom de la "définition" (dans notre exemple, ce sera def_toto). Nous aurions pu aussi paramétrer la définition de toto en mettant en argument les variables à paramétrer : def_toto< N > (avec N la taille des éléments en entrée ou en sortie)

→ Création des ports d'entrée (i1 et i2)

- lancer *Create Port* dans le menu *edit*
- définir le nom et le type du port, dans notre exemple, ce sera :
? int i1
- faire les mêmes manipulations pour i2

→ Création des ports de sortie (o)

- lancer *Create Port* dans le menu *edit*
- définir le nom et le type du port, dans notre exemple, ce sera :
! int o

Ainsi, nous obtenons la fenêtre suivante :



1-4 Création des références

Une fois les définitions créées, l'algorithme de l'application sera construit sous la forme d'un graphe où chaque sommet (boîte) sera une référence à l'une des définitions précédentes. Pour cela, nous allons créer une nouvelle définition que nous appellerons par exemple "application" :

→ Création de "application"

- lancer *New Local Definition* dans le menu *Algorithm*
- choisir le type de la définition (dans notre exemple, ce sera *Function*)
- donner le nom de la "définition" (dans notre exemple, ce sera "application").
- faire de cette nouvelle définition l'algorithme principal en lançant *Main Definition* dans le menu *edit*

→ Inclure dans l'application les bibliothèques dont nous avons besoin :

- dans notre application, nous allons inclure la bibliothèque *int* :
lancer *Inclure Library* dans le menu *File* et choisir *int*

- Créer des références à toutes les définitions dont nous avons besoin :
- Création de toto :
 - * lancer *Create reference* dans le menu *edit*
 - * double click sur *def_toto*
 - * donner le nom de la référence, les paramètres si nécessaire et le nombre de répétition (dans notre cas, ce sera “*toto [1]*” étant donné que nous n’avons pas paramétré *def_toto*)

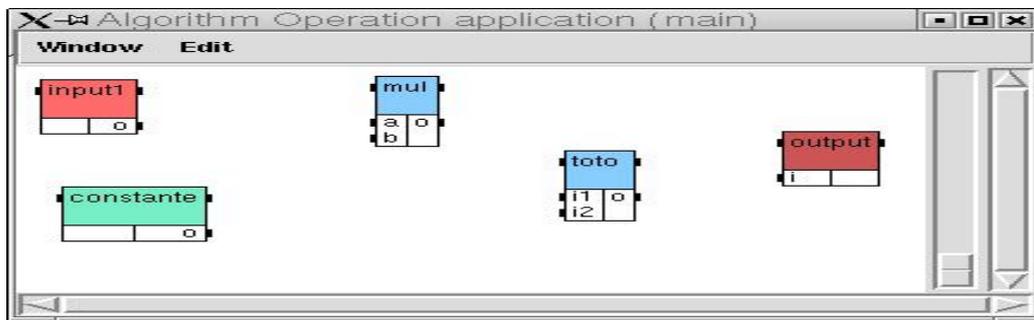
 - Création de input1 :
 - * lancer *Create reference* dans le menu *edit*
 - * choisir la librairie *int* en cliquant sur *int* (la taille de toutes les définitions standard de SynDEX-Ic sont paramétrées)
 - * double click sur *input*
 - * donner le nom de la référence, les paramètres si nécessaire et le nombre de répétition (dans notre cas, ce sera “*input1<1>[1]*” donc nous avons 1 seul élément sur le port de sortie de input1)

 - Création de mul :
 - * lancer *Create reference* dans le menu *edit*
 - * choisir la librairie *int* en cliquant sur *int* (la taille de toutes les définitions standard de SynDEX-Ic sont paramétrées)
 - * double click sur *Arit_mul*
 - * donner le nom de la référence, les paramètres si nécessaire et le nombre de répétition (dans notre cas, ce sera “*mul<1>[1]*” donc le ***mul*** ne traite qu’un seul élément)

 - Création de constante :
 - * lancer *Create reference* dans le menu *edit*
 - * choisir la librairie *int* en cliquant sur *int* (la taille de toutes les définitions standard de SynDEX-Ic sont paramétrées)
 - * double click sur *cst*
 - * donner le nom de la référence, les paramètres si nécessaire et le nombre de répétition (dans notre cas, ce sera “*constante<2>[1]*” donc nous avons 1 seul élément de valeur 2 sur le port de sortie de ***constante***)

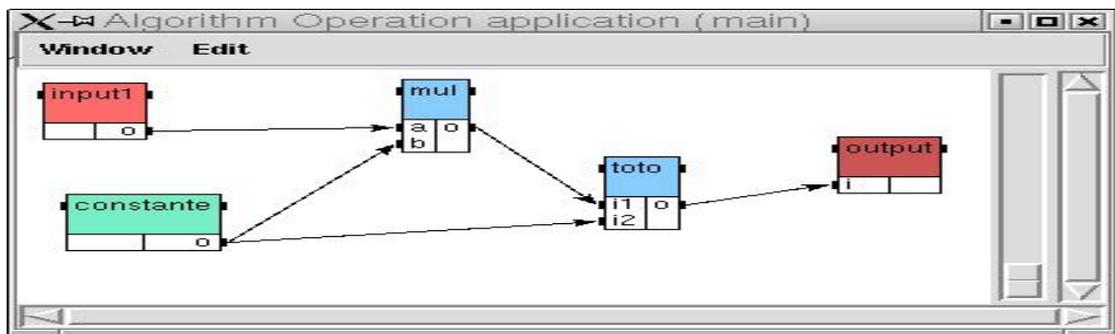
 - Création de output : faire les mêmes manipulations que pour input1
 - * lancer *Create reference* dans le menu *edit*
 - * choisir la librairie *int* en cliquant sur *int* (la taille de toutes les définitions standard de SynDEX-Ic sont paramétrées)
 - * double click sur *output*
 - * donner le nom de la référence, les paramètres si nécessaire et le nombre de répétition (dans notre cas, ce sera “*output<1>[1]*” donc nous avons 1 seul élément sur le port d’entrée de output)

Ainsi, nous obtenons la fenêtre suivante :



1-5 Création des arcs

Il faut ensuite ajouter des arcs entre ces références. Les arcs sont faits avec le bouton du milieu de la souris (restez appuyer). Ils représentent les liens de communication entre ces références. Ainsi, nous obtenons le graphe suivant :

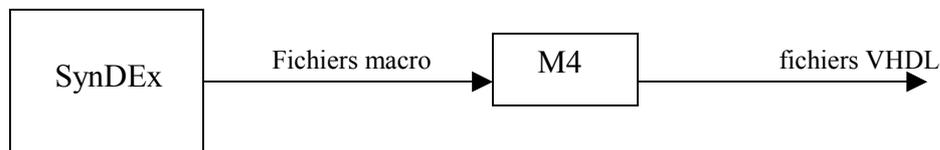


1-6 Génération du code VHDL

Une fois la spécification terminée, il est nécessaire de prendre en compte certains Paramètres avant de générer le code VHDL de notre application.

1-6-1 Avant la génération du code

- La génération de VHDL se fait en 2 étapes :
 - SynDEx-Ic génère 2 fichiers intermédiaires basés sur un macro-code; pour chaque sommet du graphe est généré une macro
 - Ces fichiers intermédiaires sont transformés en VHDL à l'aide de bibliothèques (que nous allons présenter) et d'un macro-processeur (ici GNU-m4).



L'utilisation de fichiers intermédiaires plutôt que de générer directement du VHDL permet d'avoir un générateur de code indépendant du langage cible (verilog, ...)

- Dans le fichier intermédiaire nous aurons au moins autant d'appel de macro que de sommets dans le graphe.
- Dans la phase suivante, chaque macro sera remplacée par sa définition VHDL tirée d'une librairie à l'aide du macro-processeur. Nous fournissons deux librairies :
 - La librairie VHDLlib.m4 qui contient les définitions des composants VHDL standard. Cette librairie se trouve dans le répertoire macros-syndex-ic
 - La librairie **nomapplication**.m4v qui contient les définitions des composants VHDL non standard (i.e dans l'exemple de la page 1 : le composant **toto** n'est pas standard). Cette librairie est à créer par l'utilisateur à partir du fichier model qui se trouve dans le répertoire macros-syndex-ic. Elle doit être créée dans le répertoire où se trouve le fichier .sdx de l'application mais aussi elle devra s'appeler **nomapplication**.m4v. Toutefois, si l'utilisateur oublie de créer cette librairie, SynDEx-Ic la crée par défaut. Donc une fois créée, c'est à l'utilisateur de faire les modifications nécessaires pour l'adapter à son application.

- Si l'utilisateur crée graphiquement de nouvelles définitions, il peut donner :

→ Leur code VHDL dans la fonction `Gen_entity_opn_not_in_lib` (pour les fonctions), la fonction `Gen_entity_actu_not_in_lib` (pour les actuators), la fonction `Gen_entity_sens_not_in_lib` (pour les sensors), la fonction `Gen_entity_memory_not_in_lib` (pour les Memory) ou dans la fonction `Gen_entity_const_not_in_lib` (pour les constantes) de `Appli.m4v`. Ainsi, ces différentes fonctions se présentent comme suit :

```
Gen_entity_opn_not_in_lib(typeopn, defname, portname1, dir1,
size1, tel1, type, portname2, dir2, size2, tel2, type, ...)
```

```
Gen_entity_sens_not_in_lib(typeopn, defname, portname1,
size1, tel1, type, portname2, size2, tel2, type, ...)
```

```
Gen_entity_actu_not_in_lib(typeopn, defname, portname1,
size1, tel1, type, portname2, size2, tel2, type, ...)
```

```
Gen_entity_const_not_in_lib(typeopn, defname, portname1,
size1, tel1, type, portname2, size2, tel2, type, ...)
```

```
Gen_entity_memory_not_in_lib(typeopn, defname, portname1,
dir1, size1, tel1, type, portname2, dir2, size2, tel2, type, ...)
```

Avec

Portname: nom du port

size: nombre d'élément du port

tel : taille en bit du port

type : type du port (int, float,...)

→ Ou bien le composant peut être compilé séparément comme nous allons le voir dans le point suivant.

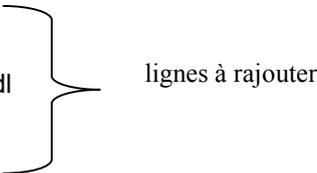
- Dans notre exemple, nous choisissons de compiler séparément le composant *input1* et de donner le code VHDL du composant *toto* dans *nomapplication.m4v* (créer au préalable à partir du model qui est dans le répertoire macros-syndex-ic). Pour cela nous allons procéder comme suit :

→ pour *input1* :

A partir de ce qui a été dit précédemment, SynDEx-Ic génère la macro Gen_entity_sens_not_in_lib(int, input, o, 1, 8, int). Etant donné que nous voulons compiler séparément le composant *input1*, nous allons insérer dans la définition de cette macro (macro qui doit se trouver dans *nomapplication.m4v*) les lignes suivantes :

Define('Gen_entity_sens_not_in_lib',

```
'Ifelse($2, input,
    'divert(5)'
    -- vcom inputbasicvect.vhdl
    'divert(0)'
    , )'
```



)

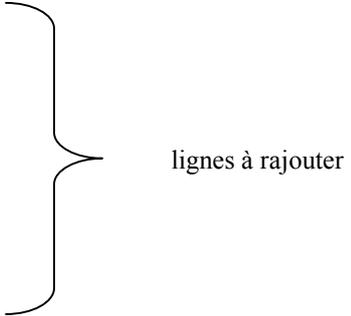
inputbasicvect.vhdl étant le fichier qui contient le VHDL du composant input1
Ces lignes auront pour effet de générer une ligne de commentaire à la fin du fichier VHDL indiquant la commande de compilation séparée

Veiller à ce que vous ayez les mêmes noms de ports sur le fichier compilé séparément et sur la spécification SynDEx

Si nous avons 2 fichiers VHDL à compiler pour des composants de type Sensor (Capteur d'entrée d'une application), nous aurions inséré les lignes suivantes :

Define('Gen_entity_sens_not_in_lib',

```
'Ifelse($2, input,
    'divert(5)'
    -- vcom inputbasicvect.vhdl
    'divert(0)'
    , 'Ifelse($2, input2,
        'divert(5)'
        -- vcom inputbasicmat.vhdl
        'divert(0)'
        , )'
```



)

avec input2 le nom de définition du deuxième Sensor
inputbasicmat.vhdl étant le fichier qui contient le VHDL du composant input2

→ Pour toto :

Etant donné que nous voulons rajouter le code VHDL de toto dans *nomapplication.m4v*, nous allons insérer dans la fonction `Gen_entity_opn_not_in_lib(int, def_toto, i1, in, 1, 8, int, i2, in, 1, 8, int, o, out, 1, 8, int)` de *nomapplication.m4v* les lignes suivantes :

`Define('Gen_entity_opn_not_in_lib',` } Ne pas rajouter cette ligne car déjà présente

Lignes à rajouter {

```
'lfalse($2, def_toto,
    library IEEE ;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_unsigned.all;
    use IEEE.numeric_std.all;
    library work;
    use work.definitions.all;
    ENTITY def_toto IS
    GENERIC(
        size1: integer:= $5; -- nombre d'élément du port1
        tel1: integer:= $6; -- taille en bit du port1
        size2: integer:= $10; -- nombre d'élément du port2
        tel2: integer:= $11; -- taille en bit du port2
        size3: integer:= $15; -- nombre d'élément du port3
        tel3: integer:= $16; ) ---- taille en bit du port3
    PORT ( i1 : in VECTOR$7(size1-1 downto 0);
          i2 : in VECTOR$12(size2-1 downto 0);
          o : out VECTOR$17(size3-1 downto 0);
          rst : in std_logic );
    END def_toto;
    ARCHITECTURE operator OF def_toto IS
        Signal sortiepuissance : signed (tel3-1 downto 0);
        Begin
            sortiepuissance <= signed(a(0)) ** signed(b(0));
            O(0) <= std_logic_vector(sortiepuissance);
        End operator;
    , )
```

) } Ne pas rajouter cette ligne car déjà présente

Si nous avons une deuxième fonction def_fct pour laquelle nous devons générer le code VHDL, nous aurions inséré les lignes suivantes :

Define('Gen_entity_opn_not_in_lib', **}** Ne pas rajouter cette ligne car déjà présente

```
'Ifelse($2, def_toto,
    library IEEE ;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_unsigned.all;
    use IEEE.numeric_std.all;
    library work;
    use work.definitions.all;
    ENTITY def_toto IS
    GENERIC(
        size1: integer:= size1; -- nombre d'élément du port1
        tel1: integer:= tel1; -- taille en bit du port1
        size2: integer:= size2; -- nombre d'élément du port2
        tel2: integer:= tel2; -- taille en bit du port2
        size3: integer:= size3; -- nombre d'élément du port3
        tel3: integer:= tel3; ) ---- taille en bit du port3
    PORT ( i1 : in VECTOR$7(size1-1 downto 0);
          i2 : in VECTOR$12(size2-1 downto 0);
          o : out VECTOR$17(size3-1 downto 0);
          rst : in std_logic );
    END def_toto;
    ARCHITECTURE operator OF def_toto IS
        Signal sortiepuissance : signed (tel3-1 downto 0);
        Begin
            sortiepuissance <= signed(a(0)) ** signed(b(0));
            O(0) <= std_logic_vector(sortiepuissance);
        End operator;
, 'Ifelse($2, def_fct,
    library IEEE ;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_unsigned.all;
    use IEEE.numeric_std.all;
    library work;
    use work.definitions.all;
    ENTITY def_fct IS

    END def_fct;
    ARCHITECTURE operator OF def_fct

    END operator;
) , )'
```

) **}** Ne pas rajouter cette ligne car déjà présente

1-6-2 Génération du code VHDL :

→ Pour visualiser le graphe de voisinage de l'application,
Lancer *View graphe de voisinage* dans le menu *Code Circuit*.

→ Pour procéder à la génération automatique du code VHDL de cette application,
Lancer *generation VHDL* dans le menu *Code Circuit*

Ainsi, 2 fichiers VHDL seront générés dans le répertoire où se trouve le fichier .sdx de l'application:

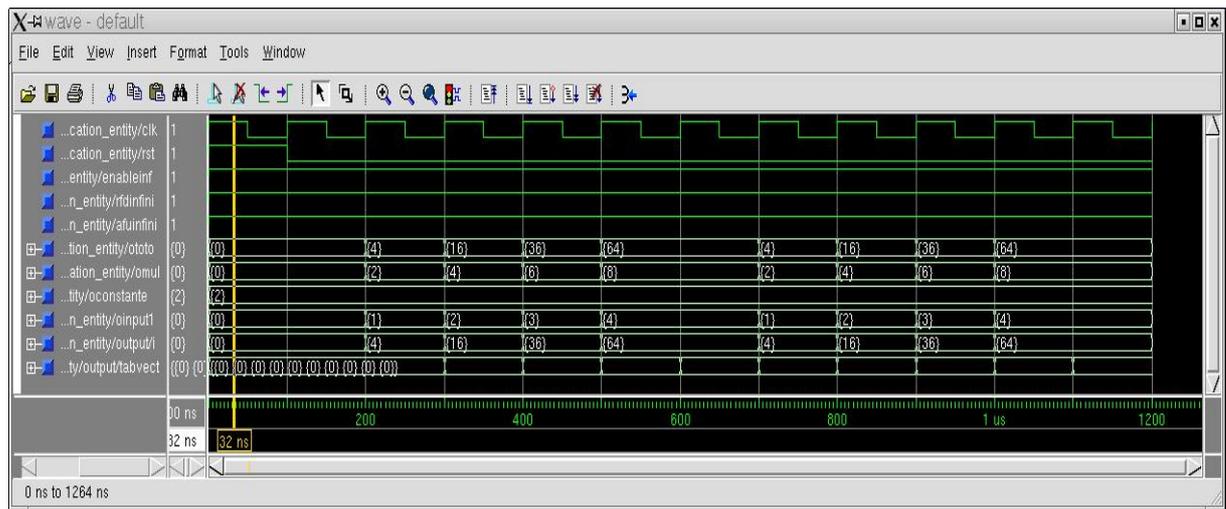
- *definition_(nom de la définition de l'application).vhd* (ce fichier contient la définition du package de l'application c'est dire la taille des différents types de ports utilisés)
- *(nom de la définition de l'application)_toplevel.vhd* : ce fichier contient les architectures de tous les composants utilisés et l'architecture de l'application elle-même. A la fin de ce fichier, en commentaires, sont générées les commandes shell à exécuter pour compiler les éventuels composants externes de l'application selon ce qui a été défini dans ***nomapplication.m4v***.

→ Pour visualiser le code généré par SynDEx-Ic pour cette application,
Lancer *View code VHDL* dans le menu *Code Circuit*

1-7 Résultat de simulation

Après avoir lancé les commandes shell pour compiler le VHDL obtenu, nous allons utiliser Modelsim (outils Mentor Graphics) pour simuler l'application.

Exemple de simulation (simulation de l'application de la page 1):



Avec

Clk : horloge qui rythme l'entrée et la sortie des données

Enableinf : signal qui valide l'entrée et la sortie des données

oinput1 : la sortie de input1

output1 : l'entrée de output

ototo : la sortie du composant toto

omul : la sortie du composant mul

2- Les différentes règles à respecter :

Pour le bon fonctionnement du logiciel c'est à dire pour que le VHDL généré soit correct, il faut suivre certaines règles :

- les noms de références doivent être différents des noms de définitions
- Chaque Constante de valeur n est associée à une définition
- L'architecture d'un sommet Constant n'est généré automatiquement que s'il ne possède qu'un seul port de sortie
- L'architecture d'un sommet Actuator(Actionneur en sortie d'une application) n'est généré automatiquement que s'il ne possède qu'un seul port d'entrée. Dans la phase de simulation, les Components Actuator ne représentent que des mémoires de stockage.
- L'architecture d'un sommet Memory n'est généré automatiquement que s'il ne possède qu'un seul port de sortie et qu'un seul port d'entrée.
- L'architecture d'un sommet non standard n'est généré automatiquement que s'il est défini dans la librairie Appli.m4v. Comme nous l'avons vu précédemment, le module VHDL de ce sommet peut aussi être compilé séparément, mais il faut veiller à ce que nous ayons **les mêmes noms de port**, le même nom de définition sur la spécification algorithmique et sur le module VHDL compilé séparément.
- Pour ne pas avoir d'erreurs de compilation en simulant le VHDL généré, respecter l'ordre des commandes Shell générées à la fin du fichier VHDL
- Il est strictement interdit de choisir dans la spécification des noms de référence ou des noms de définition qui sont mots clé en VHDL (exemple select, in, out, ...)
- à Chaque application correspond un nom de définition

ANNEXE

→ L'application présentée à la page 1 est contenue dans le fichier *exemptutorial.sdx* qui se trouve dans le répertoire *macros-syndex-ic*

→ La librairie des opérations non standard est le fichier *application.m4v* qui se trouve dans le répertoire *macros-syndex-ic*. La définition VHDL du composant *toto* se trouve dans *application.m4v* et les lignes de commande à rajouter pour le composant *input1* se trouve également dans le fichier *application.m4v*

→ Pour générer et simuler le code VHDL :

- Charger *exemptutorial.sdx* dans SynDEx
- Générer le code VHDL : lancer *generation vhdl* dans le menu *Code Circuit*
- Visualiser le code VHDL : lancer *View code vhdl* dans le menu *Code circuit*
- Quitter SynDEx-IC
- Récupérer le fichier *vhdl* correspondant à l'entrée *input1* dans le répertoire *macros-syndex-ic*
- Lancer les lignes de commandes qui sont en commentaire à la fin du fichier généré.
- Ouvrir MODELSIM par la commande *vsim*
- Charger l'application
- Lancer la simulation