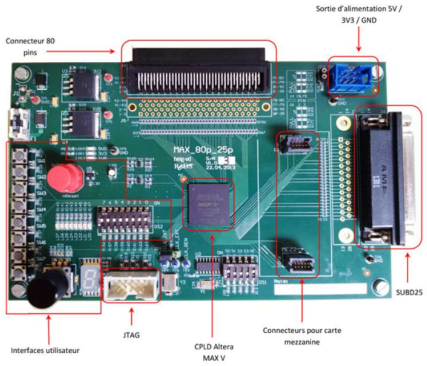
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Reconfigurable & Embedded
Digital Systems

Encodeur de priorité

CONCEPTION DE SYSTÈMES NUMÉRIQUES (CSN)



Auteur :	Spinelli Isaia et Muller Pierrick
Prof :	Etienne Messerli
Ing :	Sébastien Masle
Date :	03.10.2019
Salle :	A09 – HEIG-VD
Classe :	CSN

Table des matières

Introduction

Il s'agit de réaliser un circuit encodeur de priorité disposant de n entrées. La fonction d'un tel circuit est d'indiquer le numéro de l'entrée active ayant le degré de priorité le plus élevé. L'entrée avec l'indice 0 est la moins prioritaire et l'entrée avec l'indice le plus élevé est la plus prioritaire. Dans le cas d'un circuit à 4 entrées, c'est l'entrée in3 la plus prioritaire, et dans le cas d'un circuit à 16 entrées c'est l'entrée in15.

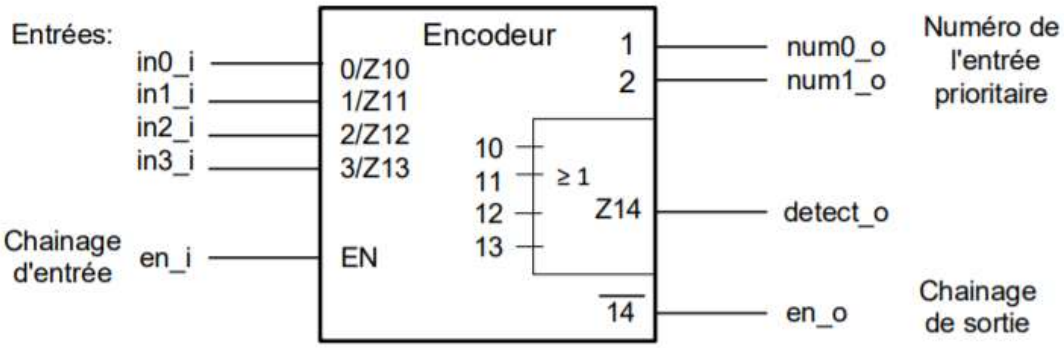
Nous allons réaliser dans une première étape une version à 4 entrées qui sera décrite à l'aide d'une description textuelle en VHDL. Dans une seconde étape nous allons réaliser une version à 16 entrées en utilisant une solution hiérarchique. Nous allons concevoir un schéma comprenant plusieurs modules à 4 entrées et de la logique.

Encodeur de priorité à 4 entrées

Spécifications

La première étape est la conception d'un encodeur de priorité comprenant 4 entrées. Ce composant doit être modulaire. Il comprend une entrée et une sortie de chaînage.

Voici le symbole de l'encodeur de priorité à 4 entrées :



Voici la table de vérité de l'encodeur de priorité à 4 entrées :

En_i	Entrées				En_o	Detect	Numéro	
	3	2	1	0			1	0
0	x	x	x	x	0	0	-	-
1	0	0	0	0	1	0	-	-
1	0	0	0	1	0	1	0	0
1	0	0	1	x	0	1	0	1
1	0	1	x	x	0	1	1	0
1	1	x	x	x	0	1	1	1

Ce module sera réalisé à l'aide d'une description VHDL synthétisable. Nous devrons analyser le fonctionnement du système afin de déterminer les instructions les plus adéquates et, si nécessaire, décomposer la description textuelle.

Description textuelle

Si l'entrée En_i n'est pas active, alors En_o et detect_o sont désactivés et num0_o ainsi que num1_0 sont « dont-care ».

Sinon

Si les entrées in0_i,in1_i,in2_i,in3_i sont désactivées, alors En_o est active et detect_o est désactivée et num0_o ainsi que num1_o sont « dont-care ».

Sinon

Si l'entrée in3_i est active, alors En_o est désactivée et detect_o est active et num0_o ainsi que num1_o sont active

Sinon

Si l'entrée in2_i est active, alors En_o est désactivée et detect_o est active et num0_o est désactivée et num1_o est active

Sinon

Si l'entrée in1_i est active, alors En_o est désactivée et detect_o est active et num0_o est active et num1_o est désactivée

Sinon

Si l'entrée in0_i est active, alors En_o est désactivée et detect_o est active et num0_o ainsi que num1_o sont désactivée.

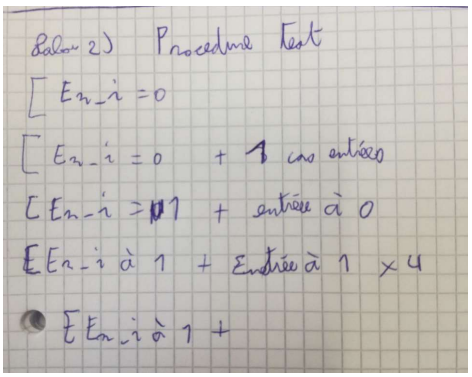
Description en VHDL

```
architecture flot_don of enc_prio_4in is
  signal num_s : std_logic_vector(1 downto 0);
  signal inAll0_s : std_logic;
  signal detect_s : std_logic ;
  signal en_s : std_logic ;
begin

  num_s <=   "11" when in3_i = '1' else
             "10" when in2_i = '1' else
             "01" when in1_i = '1' else
             "00" when in0_i = '1' else
             "--"; -- don't care

  inAll0_s <= '1' when (in3_i = '0' and in2_i = '0' and in1_i = '0' and in0_i = '0') else
              '0';

  detect_s <= '0' when en_i = '0' or inAll0_s = '1' else
              '1';

  en_s <= '1' when en_i = '1' and inAll0_s = '1' else
          '0';

  -- affectation valeurs de sortie
  num0_o <= num_s(0) ;
  num1_o <= num_s(1) ;
  detect_o <= detect_s;
  en_o     <= en_s;

end flot_don;
```

La structure du code VHDL nous était fournie, nous devions simplement ajouter la description en VHDL basée sur notre description textuelle. Afin de nous simplifier la vie, nous avons choisi de passer l'utilisation de signaux internes à l'architecture pour effectuer les assignations des sorties de notre encodeur. De plus, nous avons ajouté un signal « inAll0_s » nous permettant de savoir si toutes les entrées de notre encodeur étaient désactivées. Pour ce qui est du reste, on assigne simplement au vecteur de signaux « num_s » les valeurs de sorties correspondantes aux valeurs d'entrée selon la table de vérité, et l'on fait de même avec le signal detect_s et le signal en_s. Finalement, on assigne aux sorties de notre encodeur les signaux correspondants.

Simulation

Nous avons mis en place une procédure de test afin de vérifier le bon fonctionnement de notre implémentation VHDL :



Une fois cette procédure mise au propre, nous obtenons le tableau suivant, avec les résultats suivants après tests avec la console REDS :

Test	Résultat
En_i est désactivé	En_o est désactivé ainsi que detect_o. L'état de num1_o et num0_o n'est pas importants.
En_i est désactivé et une des entrée est activée	En_o est désactivé ainsi que detect_o. L'état de num1_o et num0_o n'est pas importants.
En_i est activé et les entrées sont toutes désactivées	En_o est activé, detect_o est désactivé et L'état de num1_o et num0_o n'est pas importants.
En_i est activé et toutes les entrées sont activées	En_o est activé ainsi que detect_o. num1_o est actif et num0_o est actif.

De plus, nous avons pratiqué une série d'autre tests, notamment le test de l'encodeur avec En_i d'activé et chacune des entrées activées en forme « d'escalier » (D'abord la in0_i, puis in0_i et in1_i, etc...), et des tests avec En_i activé et avec les entrées activées de manière aléatoires (certaines activées, d'autres désactivées).

Voici le résultat de la compilation pour la simulation manuelle :

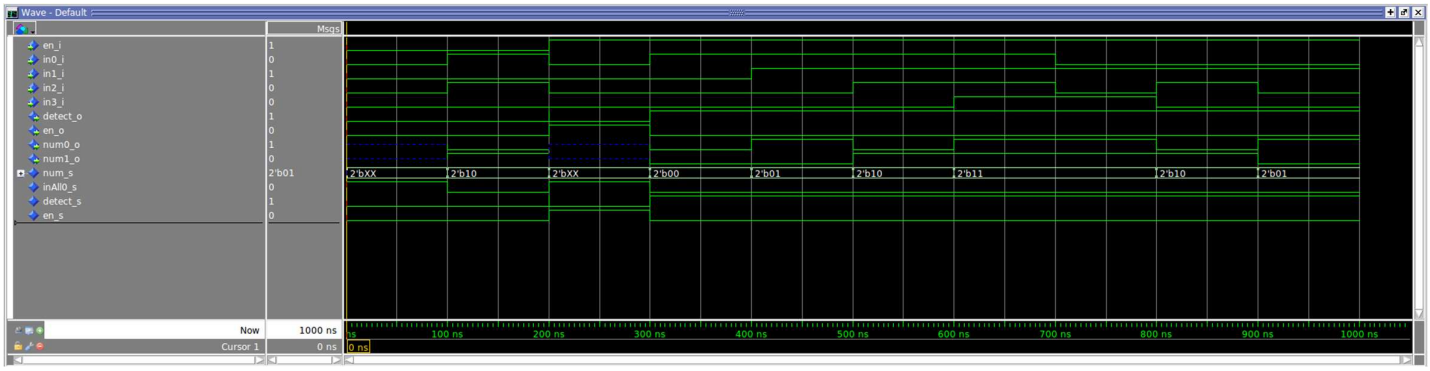
```
# Errors: 0, Warnings: 0
QuestaSim> vsim work.console_sim
# vsim work.console_sim
# Start time: 17:04:28 on Oct 03,2019
# ** Note: (vsim-3812) Design is being optimized...
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.console_sim(struct)#1
# Loading work.enc_prio_4in(flot_don)#1

VSIM 7>
```

Les résultats des tests avec la console reds sont disponibles ci-dessous. Plusieurs points intéressants peuvent être relevés :

- Pour les deux tests effectués avec En_i désactivé, nous avons une sortie « Dont care » pour le premier (entre 0 et 100 ns), ce qui n'est pas le cas pour le deuxième (entre 100 et 200 ns). Cela est dût au fait que les sorties ne s'intéressent pas au signal En_i, mais uniquement aux entrées de l'encodeur. On peut le voir dans le deuxième cas de « Dont care » (entre 200 et 300 ns), où En_i est activé mais toutes les autres entrées sont désactivées

- Entre 300 et 700 ns, on peut voir la forme « d'escalier » dont nous avons parlé plus tôt, et en observant la valeur de num0_o et num1_o, on sait qu'elle correspond à la table de vérité de notre encodeur et au fonctionnement voulu.

- Entre 700 et 1000 ns, nous avons effectué les tests aléatoires. Ici aussi, le fonctionnement est bon selon nos spécifications.

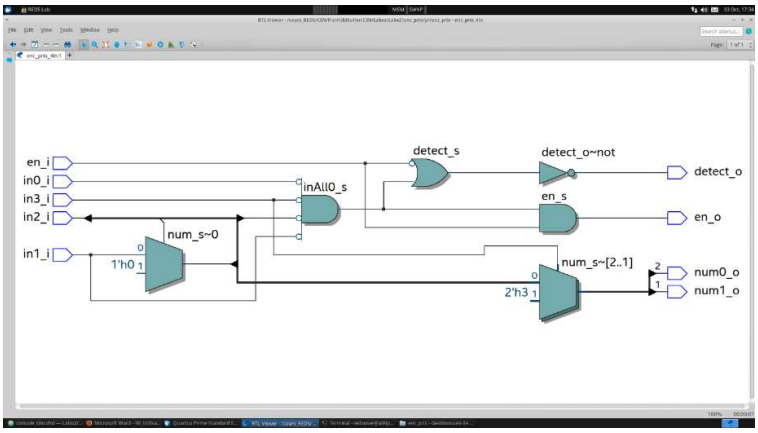


Synthèse

Une fois la synthèse effectuée, nous avons pu observer la vue RTL ainsi que la quantité de logique.

Vue RTL

Voici la vue RTL de notre encodeur de priorité à 4 entrées :



On peut constater qu'il a en effet synthétiser de façon optimiser la logique pour les sortis num0_o. On peut aussi reconnaitre notre signal interne « inAll0_s » qui est bien utilisé pour le detect_s et le en_s.

Quantité de logique obtenue

Voici les quantités de logique obtenue avec notre description VHDL :

eport - enc_prio_4in

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	M Bloc	DSP Elements	DSP 9x9	DS
1	|enc_prio_4in	4 (4)	0 (0)	0	0	0	0	0

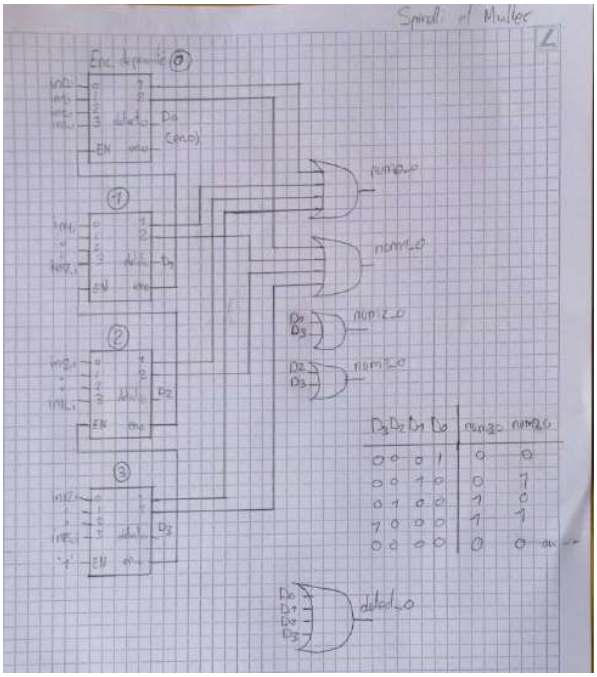
On peut constater dans le champ Combinational ALUTs que nous obtenons 4 éléments logiques.

Encodeur de priorité à 16 entrées

Analyse

Schéma hiérarchique

Nous avons commencé par établir un schéma hiérarchique afin de simplifier la future implémentation de notre description VHDL.



Nous avons commencé par placer les 4 encodeurs de priorité chainé a l'aide du signal EN permettant justement le chainage. Par suite de cela, nous nous sommes intéressés aux sorties detect_o afin de les réutiliser avec un encodeur de priorité pour gérer nos sortis num2_o et num3_o. Mais en établissant la table de vérité, nous nous sommes rendu compte que nous pouvions simplifier la logique en utilisant pas le signal detect_o le signal représenter le point le plus faible. Ainsi, nous avons pu utiliser seulement 2 portes OU au lieu d'un encodeur de priorité afin de générer les signaux de sortis num2_o et num3_o.

Pour ce qui est de gérer les signaux de sortie num0_o et num1_o nous pensions utiliser un multiplexeur mais nous nous sommes rendu compte qu'en modifiant notre encodeur de priorité à 4 entrées, il était possible de simplifier la logique en remplaçant les 2 multiplexeurs avec 2 portes OU.

Pour la gestion de la sortis detect_o de l'encodeur de priorité 16 à 4, nous avons simplement utilisé une porte OU groupant tous les detect_o des encodeurs de priorité 4 à 2.

Modification de l'encodeur de priorité à 4 entrées

Ici, nous pouvons voir les modifications effectuées :

1. Au lieu de mettre les sortis num0_o et num1_o en dont'care, nous forçant les sortis à 0 afin que si toutes les entrées de notre encodeur soient à 0 les sortis aussi.
2. On s'assure que nos sortis num0_o et num1_o soit à 0 si l'encodeur n'est pas activé donc si l'enable n'est pas actif.

```
end enc_prio_4in ;

architecture flot_don of enc_prio_4in is
  signal num_s : std_logic_vector(1 downto 0);
  signal inAll0_s : std_logic;
  signal detect_s : std_logic ;
  signal en_s : std_logic ;
begin

  num_s <=   "11" when in3_i = '1' else
             "10" when in2_i = '1' else
             "01" when in1_i = '1' else
             "00" when in0_i = '1' else
             "00"; -- Utilisé pour la logique de l'encodeur prio 16

  inAll0_s <= '1' when (in3_i = '0' and in2_i = '0' and in1_i = '0' and in0_i = '0') else
              '0';

  detect_s <= '0' when en_i = '0' or inAll0_s = '1' else
              '1';

  en_s <= '1' when en_i = '1' and inAll0_s = '1' else
          '0';

  -- affectation valeurs de sortie
  num0_o <= num_s(0) when en_i = '1' else
            '0';
  num1_o <= num_s(1) when en_i = '1' else
            '0';
  detect_o <= detect_s;
  en_o     <= en_s;
```

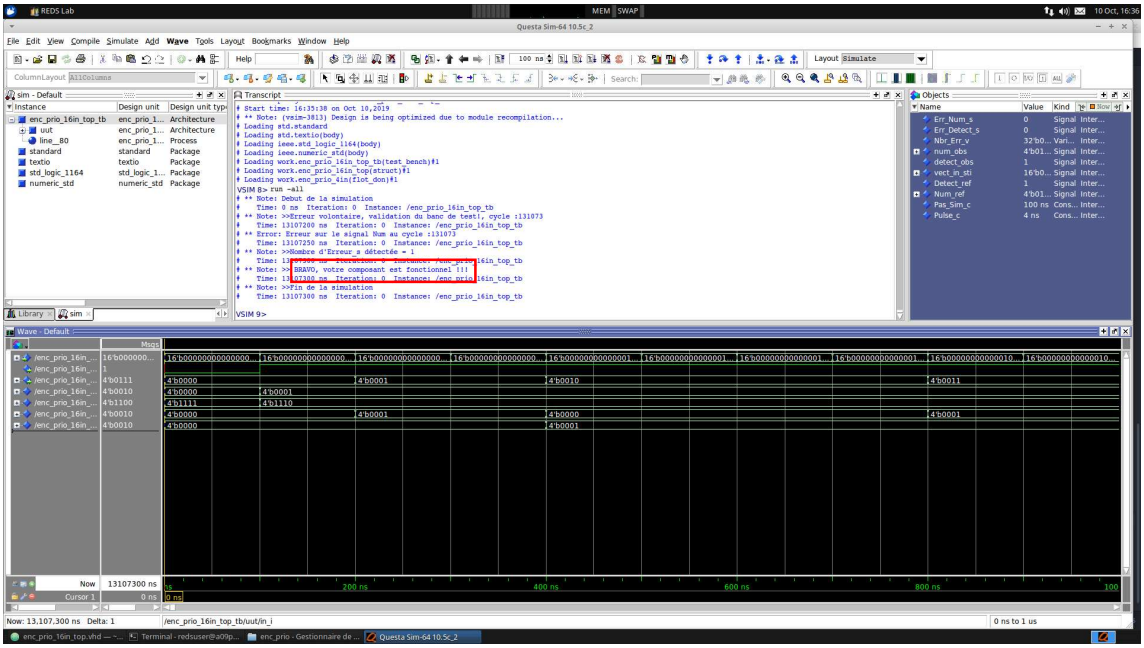
Nous avons effectué ces modifications afin de pouvoir s'assurer que les sortis num0_o et num1_o soient à 0 si elles ne sont pas utilisées par l'encodeur de priorité 16 à 4. Grâce à cela, nous avons pu grouper toutes les sortis num0_o avec une porte OU et de même pour les sortis num1_o afin de gérer les sortis num1_o et num0_o de l'encodeur de priorité 16 à 4.

Description en VHDL

La description VHDL de notre encodeur de priorité 16 à 4 se trouve en annexe.

Simulation automatique

Grâce aux logs, nous pouvons constater que tous les tests ont été passés avec succès.

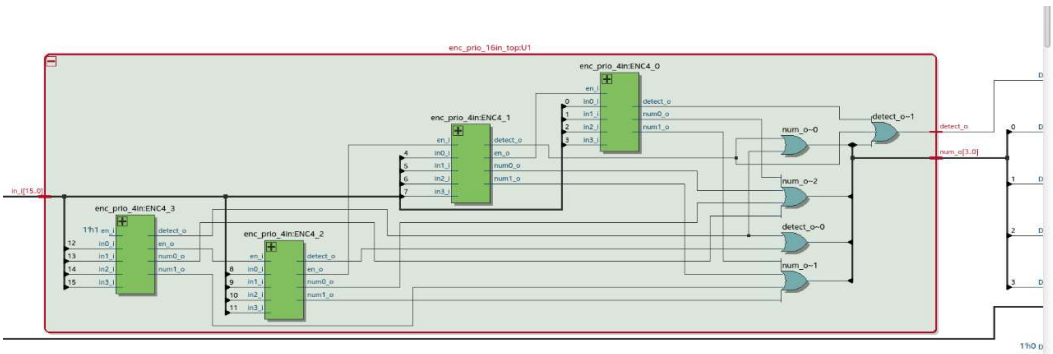


Nous nous sommes rendu compte après coup que les noms des signaux n'étaient pas visibles entièrement. Etant donné que le test Bench été passé avec succès de la compilation à l'exécution, ceci n'est pas un problème majeur.

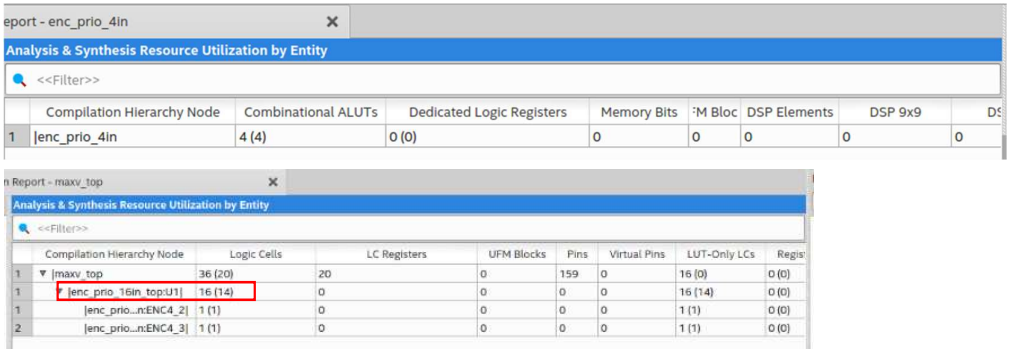
Synthèse

Vue RTL

Nous pouvons facilement voir ci-dessous la chainage des encodeurs de priorité 4 à 2. On peut voir que la logique de la vue RTL correspond à notre schéma hiérarchique à l'exception d'une simplification dans la création du signal detect_o qui utilise une porte OU avec 3 entrée au lieu de 4 en réutilisant une porte OU servant à créer le signal num2_o.



Quantité de logique et évolution



eport - enc_prio_4in

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	M Bloc	DSP Elements	DSP 9x9	DS
1	|enc_prio_4in	4 (4)	0 (0)	0	0	0	0	0

Report - maxv_top

Analysis & Synthesis Resource Utilization by Entity

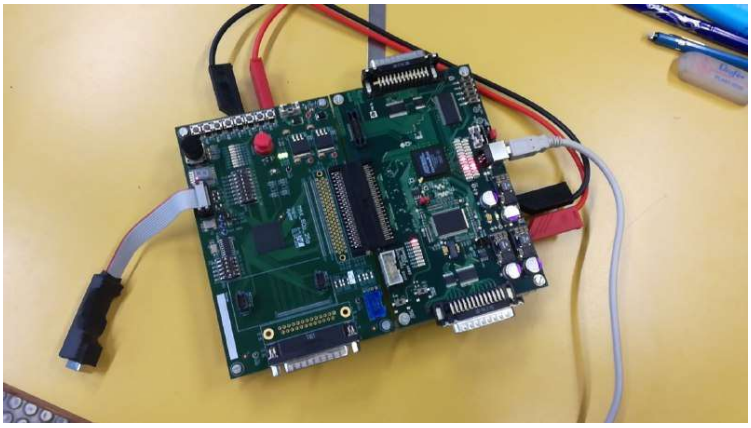
<<Filter>>

	Compilation Hierarchy Node	Logic Cells	LC Registers	UFM Blocks	Pins	Virtual Pins	LUT-Only LCs	Regis
1	▼ |maxv_top	36 (20)	20	0	159	0	16 (0)	0 (0)
1	▼ |enc_prio_16in_top:U1|	16 (14)	0	0	0	0	16 (14)	0 (0)
1	|enc_prio...n:ENC4_2|	1 (1)	0	0	0	0	1 (1)	0 (0)
2	|enc_prio...n:ENC4_3|	1 (1)	0	0	0	0	1 (1)	0 (0)

On peut voir que l'évolution des quantités de logique utilisées est linéaire car avec 4 encodeurs de priorité 4 à 2 utilisant chacun 4 éléments logique on obtient un total de 16 éléments logiques pour l'encodeur de priorité 16 à 4.

Intégration

L'intégration a été réalisée avec la carte « maxv-25-80 » ainsi que la carte « ConsoleUSB-2 » qui permet de simuler la console REDS pour les tests.



Test de l'intégration

Sur le montage, nous avons testé tous les cas de priorité avec une entrée activée. Tous les résultats étaient cohérents avec la spécification. De plus, nous avons effectué les 4 tests présents dans le tableau (p.5) qui se sont aussi révélés corrects.

La validation a été effectuée le 10.10.19 par M. Messerli.

Conclusion

Difficultés rencontrées

- Compréhension de la logique sous-jacente à l'utilisation d'entité externe lors d'une description VHDL.
- Prise en main de la méthodologie de test et de compilation d'une description VHDL.

Compétences acquises

- Création de description VHDL.
- Consolidation de la méthodologie de test, compilation, synthèse et intégration.
- Analyse et établissement d'un schéma hiérarchique.

Résultats obtenus

Nous avons réussi à mettre en place toutes les étapes qui nous étaient demandé dans ce laboratoire. Les 2 description VHDL sont synthétisable et intégrable. Finalement, nous pensons qu'à l'aide de notre schéma hiérarchique nous avons pu trouver une solution optimisée en termes de quantité de logique.

Date : 16.10.19

Nom de l'étudiant : Spinelli Isaia et Muller Pierrick

```vhdl
--------------------------------------------------------------------------------
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- Fichier      : enc_prio_16in_top
-- Description  : Encodeur de priorite à 16 entrees
--
-- Auteur       : E. Messerli
-- Date         : 30.09.2009
-- Version      : 0.0
--
-- Utilise      : Labo systeme numerique
--
--| Modifications |------------------------------------------------------------
-- Ver  Auteur  Date       Description
-- 1.0   GAA  31.08.2016   Adaptation fichier vhdl pour utilisation avec Quartus
-- 1.1   EMI  05.10.2016   Adaptation pour encodeur a 16 entress
--                         entity: utilise vecteurs pour in_i et num_o
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.ALL;
--use ieee.numeric_std.ALL;

entity enc_prio_16in_top is
   port(
      in_i     : in     std_logic_vector(15 downto 0);
      detect_o : out    std_logic;
      num_o    : out    std_logic_vector(3 downto 0)
   );
end enc_prio_16in_top ;

architecture struct of enc_prio_16in_top is

  -- Component Declarations
  component enc_prio_4in
    port(
    en_i     : in     std_logic;
    in0_i    : in     std_logic;
    in1_i    : in     std_logic;
    in2_i    : in     std_logic;
    in3_i    : in     std_logic;
    detect_o : out    std_logic;
    en_o     : out    std_logic;
    num0_o   : out    std_logic;
    num1_o   : out    std_logic
    );
  end component;

  for all : enc_prio_4in use entity work.enc_prio_4in(flot_don);

  -- Internal Declarations
  constant ENABLE      : std_logic:='1';

  signal vect_detect_s, vect_enable_s : std_logic_vector(3 downto 0);
  signal vect_num0_s, vect_num1_s : std_logic_vector(3 downto 0);

begin

  ENC4_3 : enc_prio_4in port map (
      en_i => ENABLE,
      in0_i => in_i(12)  ,
      in1_i => in_i(13)  ,
      in2_i => in_i(14)  ,
      in3_i => in_i(15)  ,
      detect_o => vect_detect_s(3) ,
      en_o  => vect_enable_s(3) ,
      num0_o =>  vect_num0_s(3) ,
```

```
70          num1_o  => vect_num1_s(3)
71      );
72
73      ENC4_2 : enc_prio_4in port map (
74          en_i => vect_enable_s(3),
75          in0_i => in_i(8)  ,
76          in1_i => in_i(9)  ,
77          in2_i => in_i(10)  ,
78          in3_i => in_i(11)  ,
79          detect_o => vect_detect_s(2) ,
80          en_o  => vect_enable_s(2) ,
81          num0_o =>  vect_num0_s(2) ,
82          num1_o  => vect_num1_s(2)
83      );
84
85      ENC4_1 : enc_prio_4in port map (
86          en_i => vect_enable_s(2),
87          in0_i => in_i(4)  ,
88          in1_i => in_i(5)  ,
89          in2_i => in_i(6)  ,
90          in3_i => in_i(7)  ,
91          detect_o => vect_detect_s(1) ,
92          en_o  => vect_enable_s(1) ,
93          num0_o =>  vect_num0_s(1) ,
94          num1_o  => vect_num1_s(1)
95      );
96
97      ENC4_0 : enc_prio_4in port map (
98          en_i => vect_enable_s(1),
99          in0_i => in_i(0)  ,
100         in1_i => in_i(1)  ,
101         in2_i => in_i(2)  ,
102         in3_i => in_i(3)  ,
103         detect_o => vect_detect_s(0) ,
104         en_o  => vect_enable_s(0) ,
105         num0_o =>  vect_num0_s(0) ,
106         num1_o  => vect_num1_s(0)
107     );
108
109     num_o(3) <= vect_detect_s(3) or vect_detect_s(2);
110     num_o(2) <= vect_detect_s(3) or vect_detect_s(1);
111     num_o(1) <= vect_num1_s(3) or vect_num1_s(2) or vect_num1_s(1) or vect_num1_s(0);
112     num_o(0) <= vect_num0_s(3) or vect_num0_s(2) or vect_num0_s(1) or vect_num0_s(0);
113     detect_o <= vect_detect_s(3) or vect_detect_s(2) or vect_detect_s(1) or
        vect_detect_s(0);
114
115 end struct;
116
```