Conversion Binaire vers BCD

Double dabble

Abel DIDOUH

Introduction

Nous nous intéressons à la conversion d'un code binaire naturel en code BCD. L'objectif est de convertir une donnée data_in représentant un code binaire naturel de N bits en BCD sur data_out.

Nous ne cherchons pas à optimiser le débit, mais plutôt les ressources, le temps de développement voire la latence.

Il existe différente façon de convertir un code binaire naturel en BCD.

Le premier principe consiste à décrémenter un compteur binaire naturel qui démarre par la valeur à convertir tandis qu'on incrémente un compteur BCD. Ce principe risque d'utiliser des ressources conséquentes lorsque N est élevé.

Le second principe consiste à réaliser des divisions successives par dix. Ce principe utilise des blocs de division.

Le troisième principe consiste en l'utilisation de l'algorithme « double dabble ». C'est ce dernier que nous allons utiliser pour convertir un binaire naturel en BCD.

Architecture

L'objectif est de concevoir une architecture combinatoire convertissant une donnée data_in représentant un code binaire naturel de N bits en BCD en utilisant l'algorithme double dabble.

Interface

Le composant a pour interface :

- N (de type positive)
- data_in (de type std_logic_vector) : entrée, donnée à convertir
- data_out (de type std_logic_vector) : sortie, donnée convertie

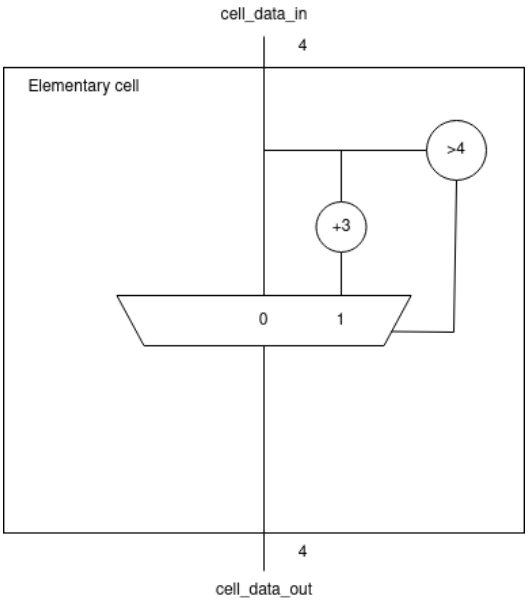
	Binaire naturel -> BCD
N	Nombre de bits
data_in	Code binaire naturel
Valeur décimale maximale	$2^N - 1$
data_out	BCD
Nombre de bits de data_out	4*ARRONDI_SUP[N*log10(2)]

Pseudo Code

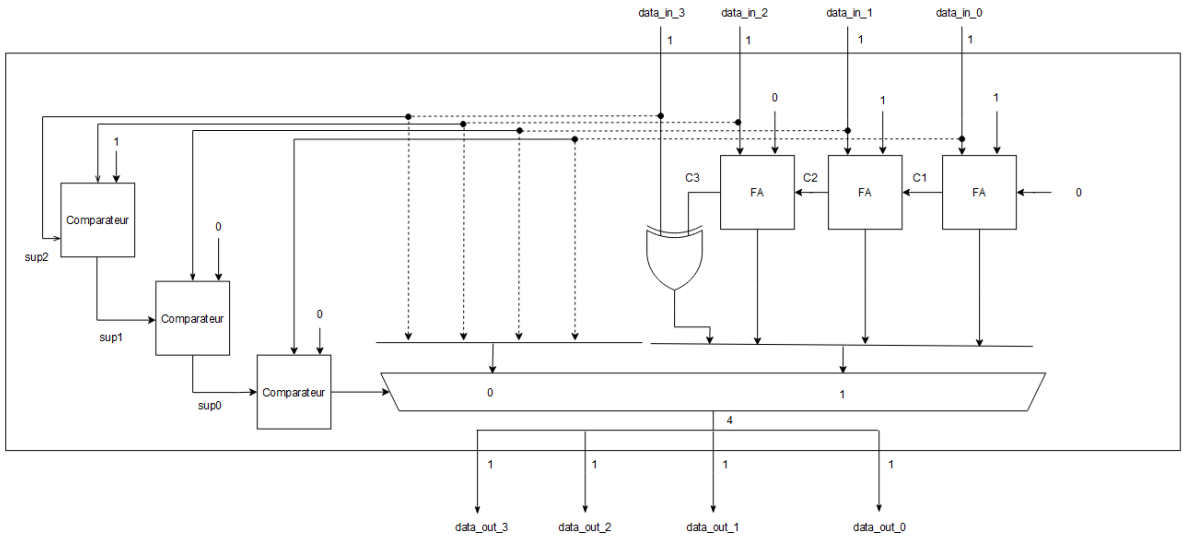
data_in : donnée de N bits
data_out : donnée de 4*ceil(log10(2**N))
tmp : donnée intermédiaire
tmp=0
tmp= data_in
Faire N fois :
    Faire pour chaque quatre bits :
        Si la valeur du digit > 4 alors additionner + 3 aux quatre bits (tmp)
data_out= tronquer tmp à la taille de data_out

Schéma

Schéma élémentaire d'une cellule :



Nous obtenons la cellule élémentaire suivante composé de 3 Full Adder, d'un multiplexeur avec deux entrées de 4 bits, d'un comparateur de supériorité sur 3 bits et d'un XOR.



Explications

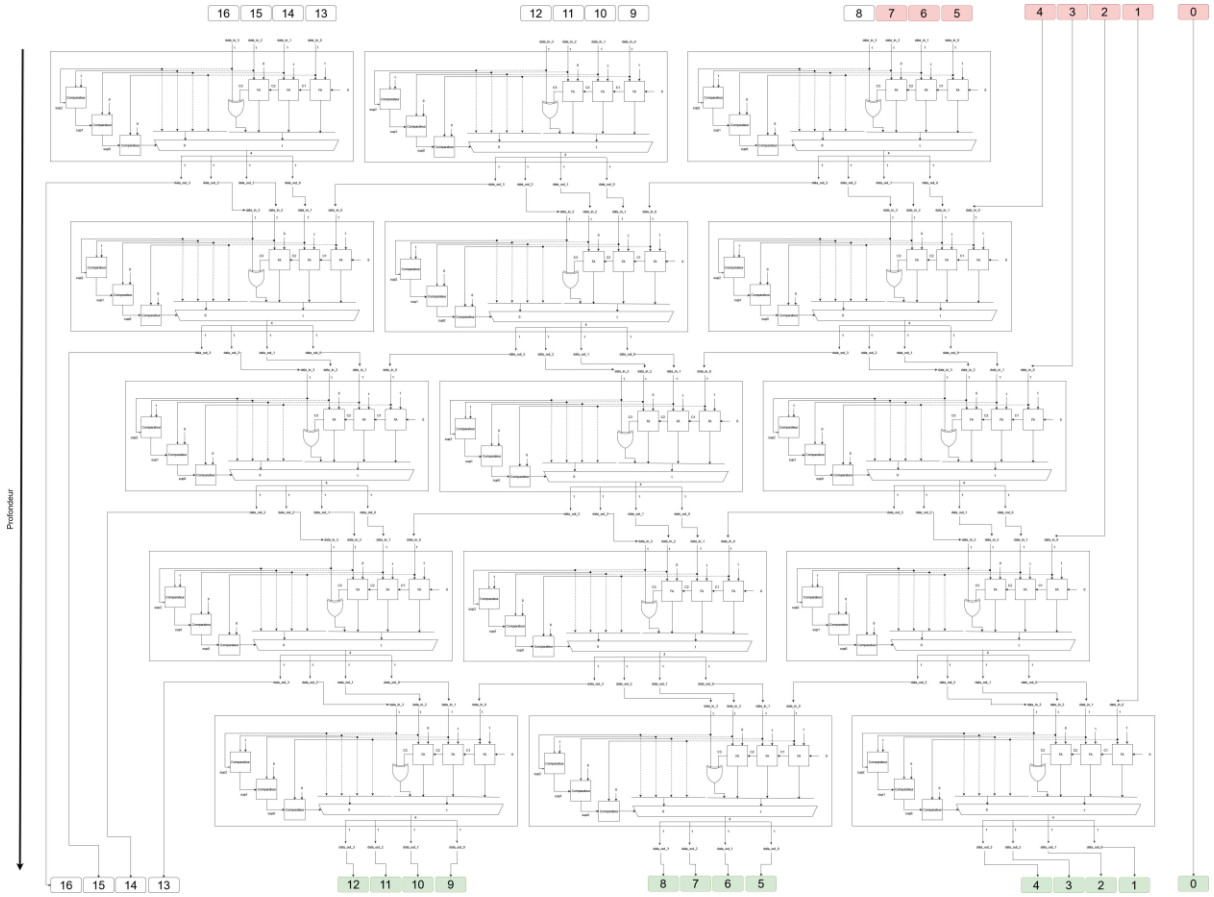
La table de vérité de la cellule élémentaire :

	D_IN	D_IN	D_IN	D_IN		D_OUT	D_OUT	D_OUT	D_OUT
0	0	0	0	0		0	0	0	0
1	0	0	0	1		0	0	0	1
2	0	0	1	0		0	0	1	0
3	0	0	1	1		0	0	1	1
4	0	1	0	0		0	1	0	0
5	0	1	0	1		1	0	0	0
6	0	1	1	0		1	0	0	1
7	0	1	1	1		1	0	1	0
8	1	0	0	0		1	0	1	1
9	1	0	0	1		1	1	0	0
10	1	0	1	0		X	X	X	X
11	1	0	1	1		X	X	X	X
12	1	1	0	0		X	X	X	x
13	1	1	0	1		X	X	X	X
14	1	1	1	0		X	X	X	X
15	1	1	1	1		X	X	X	X

Caractéristique de la cellule élémentaire :

- 3 Full Adder
- 3 Comparateur
- 1 XOR
- 1 MUX

Je prends N = 8, j'obtiens l'architecture suivante :

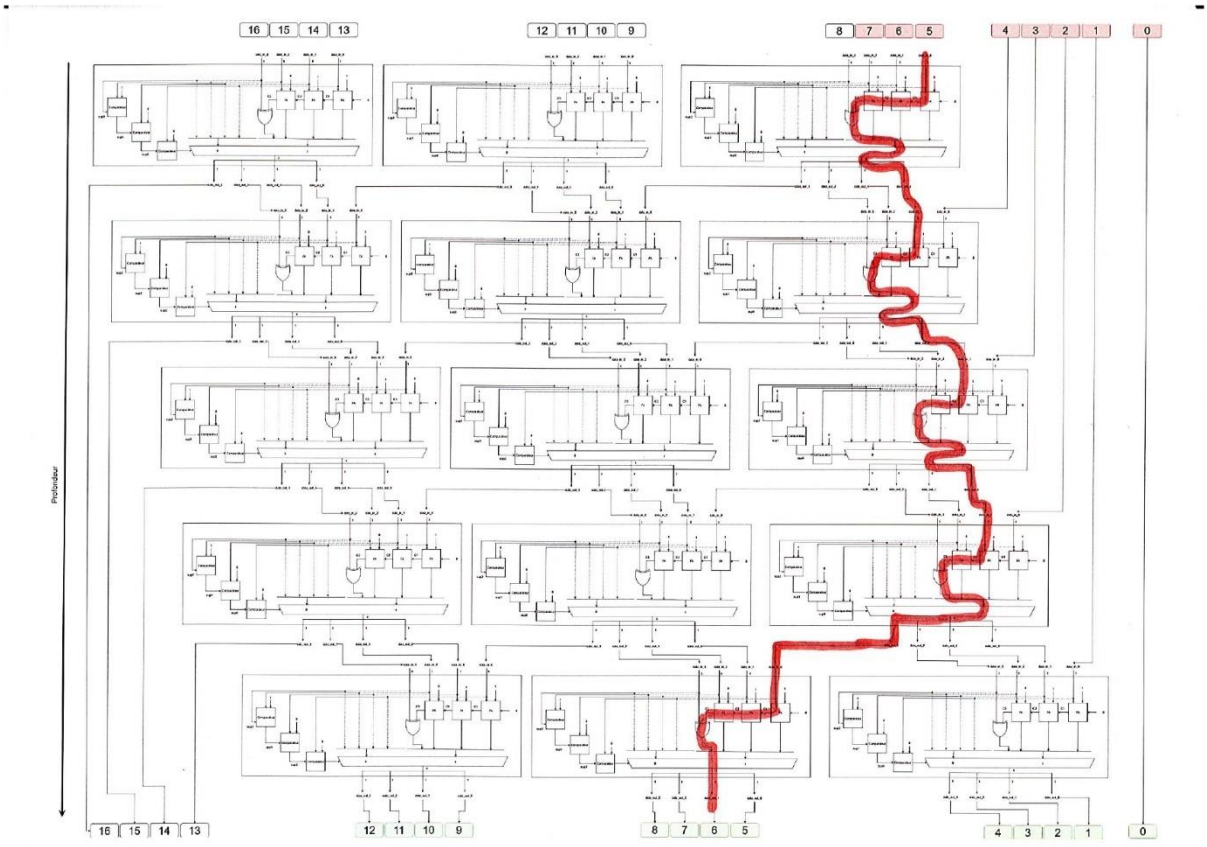


Le vecteur *data_in* est sur 8 bits. Le vecteur *data_out* est sur 4*log10(2^4) = 10 bits. Il y a 3 blocs car j'obtiens l'arrondi supérieur de (data_out/4) = 3 blocs. Je crée un vecteur binaire étendu sur 16 bits. Je crée un vecteur temporaire pour stocker toutes les valeurs issues de l'architecture. À la sortie, il me suffira de tronquer le vecteur pour obtenir une donnée adéquate au vecteur data_out.
La profondeur est égale à 5.

Une première boucle va itérer sur l'étage correspondant à la profondeur de l'architecture. La deuxième boucle va venir placer chaque bloc élémentaire sur chaque palier.

Chemin critique :

Je trouve le chemin critique suivant :



Délai

Je peux déterminer T_{min} pour NA = 8:

Pour la profondeur égale à 1 :

$$3 * \delta FA + 1 * \delta XOR + 1 * \delta MUX$$

Pour la profondeur comprise entre 2 et 4, j'obtiens pour chaque étage :

$$2 * \delta FA + 1 * \delta XOR + 1 * \delta MUX$$

Pour la profondeur égale à 5 :

$$3 * \delta FA + 1 * \delta XOR + 1 * \delta MUX$$

Finalement,

$$T_{min} = 12 * \delta FA + 5 * \delta XOR + 5 * \delta MUX$$

Nous pouvons généraliser pour NA quelconque. NA représente la taille du vecteur data_in.

$$T_{min} = 2 * (3 * \delta FA + 1 * \delta XOR + \delta MUX) + ((NA - A.SUP[log10(2^{NA})]) - 3) * (2 * \delta FA + 1 * \delta XOR + 1 * \delta MUX)$$

Cette équation permet de connaitre la profondeur. Je soustrais pour avoir le nombre d'étage intermédiaire.
$((NA - A.SUP[log10(2^{NA})]) - 3)$

Ressources

Je peux déterminer les ressources de mon architecture pour N = 8.

$$15 * (3 * FA + 1 * XOR + 1\ MUX + 1 * COMPARATEUR)$$
$$45 * (FA + COMPARATEUR) + 15 * (XOR + MUX)$$

Je peux généraliser pour NA quelconque :

$$A.SUP(log10(2^{NA})) * \left(NA - SUP(\frac{log10(2^{NA})}{4})\right) * (3 * FA + 3 * COMPARATEUR + 1 * XOR + 1 * MUX)$$

Simulation comportementale

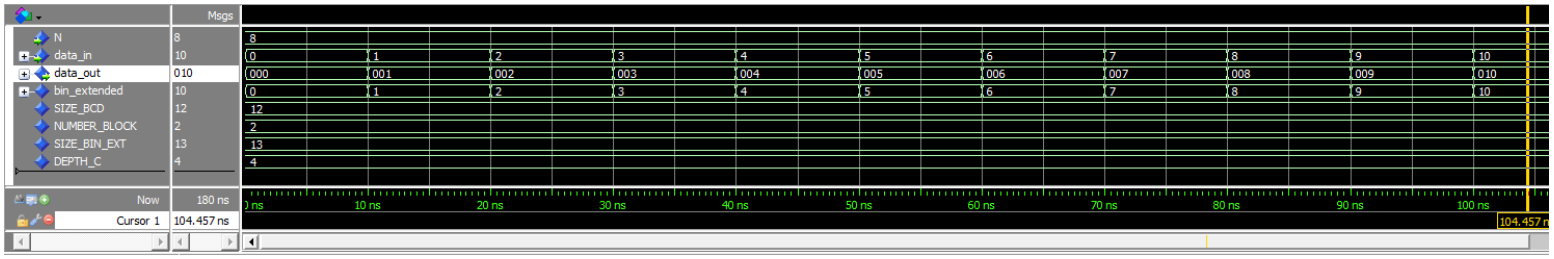
Dans cette partie, je vais réaliser une simulation Behavioral sur mon module bin2bcd. Puis je vais effectuer une simulation Behavioral sur une architecture de test. Cette architecture de test va instancier test.vhd. Je souhaite faire ce test pour savoir si mon module va correctement fonctionner lorsqu'en entrée il y a un registre data_in et en sortie il y a un registre data_out.

Pour toutes les simulations, le radix du signal data_in est du type unsigned et le radix du signal data_out est du type hexadécimal.

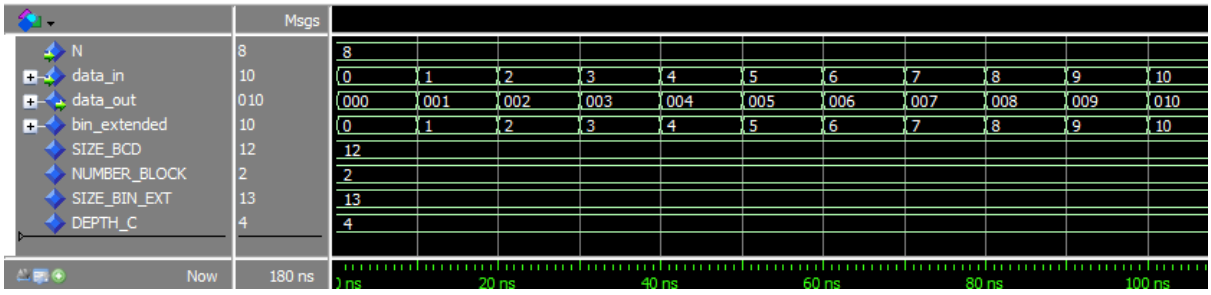
Simulation

Pour mon premier testbench, je prends N = 8. Soit s les stimulis appartenant à l'intervalle [0 ; 10].

J'obtiens la vue globale suivante :



Regardons en détails la simulation :



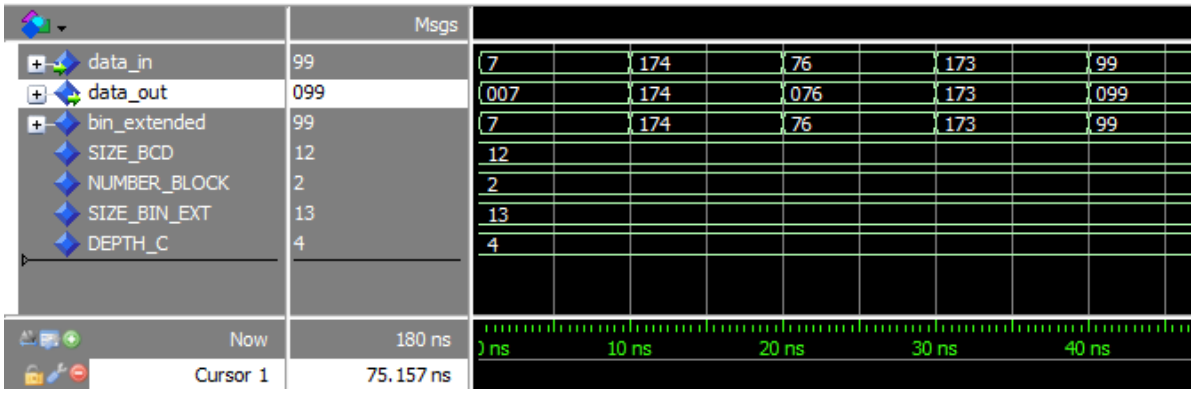
Pour les stimulis inférieur à 9, nous retrouvons bien la valeur sur le premier digit (unité). Pour la valeur 10 en décimal nous obtenons un 1 sur le deuxième digit (dizaine) et 0 sur le premier digit (unité).

L'architecture fonctionne l'intervalle [0 ; 10].

Dans ce second testbench, je garde N = 8. Je crée un tableau de stimuli. Les stimulis sont les suivants :

Binaire	Valeur Hexa.	Valeur Décimal	BCD
0000_0111	07	7	007
1010_1110	AE	174	174
0100_1100	4C	76	76
1010_1101	AD	173	173
0110_0011	63	99	99

J'ai essayé de choisir des valeurs binaires cohérentes (aucune symétrie, …)
J'obtiens la vue globale suivante pour la simulation numéro 2.



Les tests automatiques m'assurent que j'ai la bonne valeur :

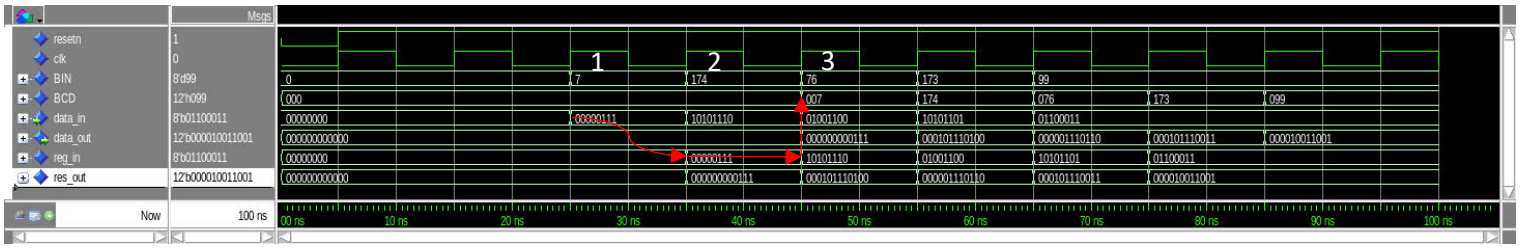
** Warning: **Test OK**
Time: 10 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Note: Resultat obtenu : **0x007**
Time: 10 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Warning: **Test OK**
Time: 20 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Note: Resultat obtenu : **0x174**
Time: 20 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Warning: **Test OK**
Time: 30 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Note: Resultat obtenu : **0x076**
Time: 30 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Warning: **Test OK**
Time: 40 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Note: Resultat obtenu : **0x173**
Time: 40 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Warning: **Test OK**
Time: 50 ns Iteration: 0 Instance: /tb_bin2bcd_v2
** Note: Resultat obtenu : **0x099**
Time: 50 ns Iteration: 0 Instance: /tb_bin2bcd_v2

Finalement, le composant BIN2BCD qui implémente l'algorithme double dabble est fonctionnel.

Regardons si notre composant est fonctionnel dans une architecture utilisant des registres en entrées et en sorties.

J'effectue une simulation Behavioral.

J'obtiens la vue globale suivante :

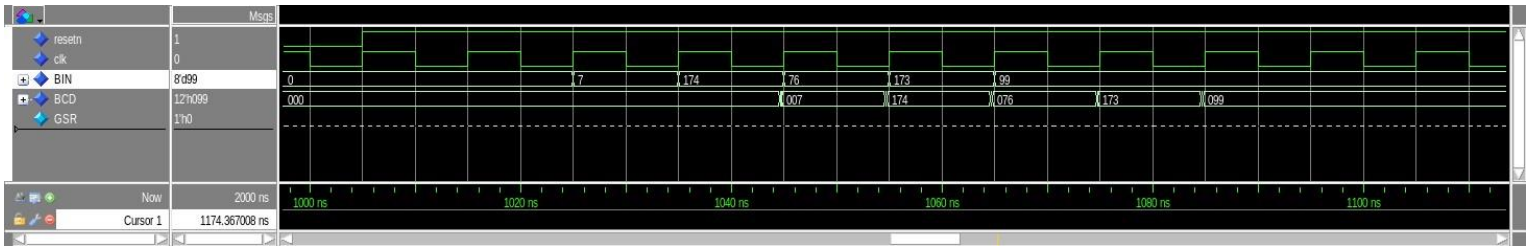


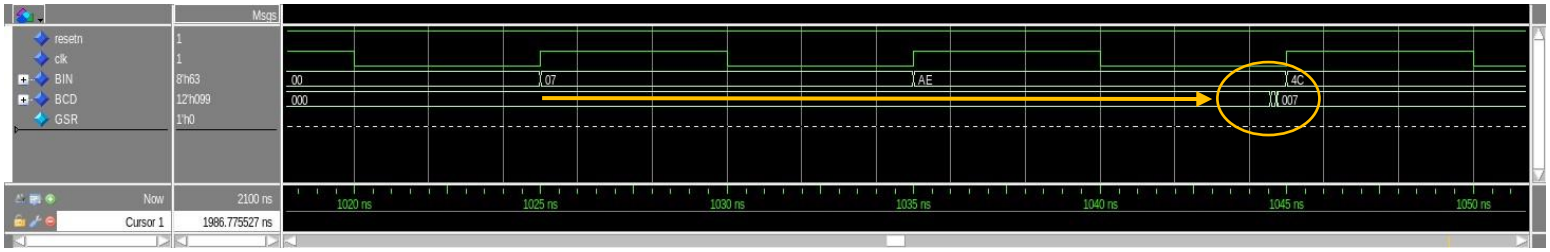
La simulation est fonctionnelle.
Lorsque l'architecture reçoit la donnée 2 fronts d'horloges montant plus tard, une donnée pertinente est disponible à la sortie de l'architecture. S'ensuit une nouvelle donnée pertinente à chaque front montant d'horloge clk.

Je réalise une simulation post-implémentation avec prise en compte des délais.

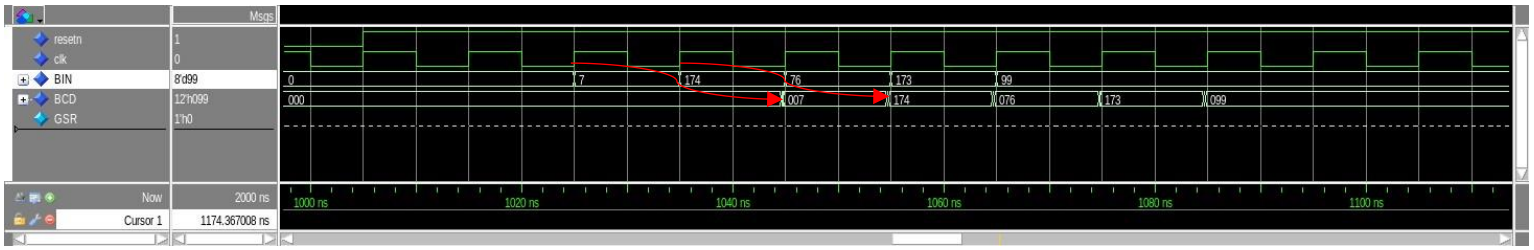
<u>Post Implémentation Timing Simulation :</u>

Vue globale de la simulation :





Entre l'horloge clk et l'horloge réelle qui arrive au registre, il y un arbre qui possède une latence. Entre le registre res_out et sa sortie BCD : il y a du routage et des plots. Il y a un temps de traversée non négligeable. Ce temps de traversé est supérieur à une période d'horloge. Tous les bits n'arrivent pas en même temps à cause des écarts dans les délais de routage des différents bits.



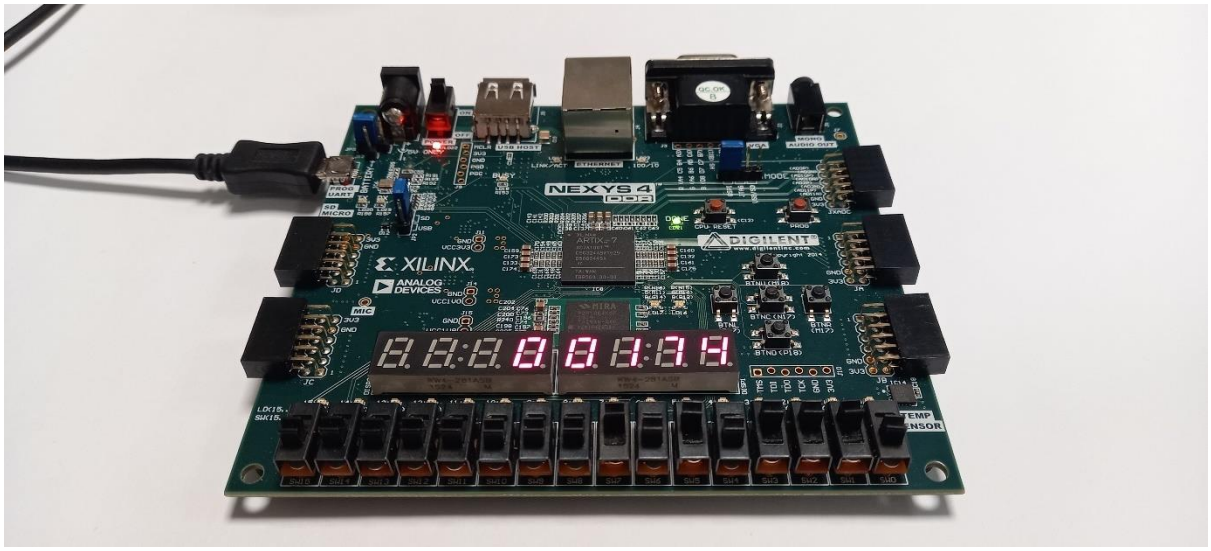
Il y a une latence de deux fronts montants d'horloges clk pour obtenir une donnée pertinente sur la sortie.

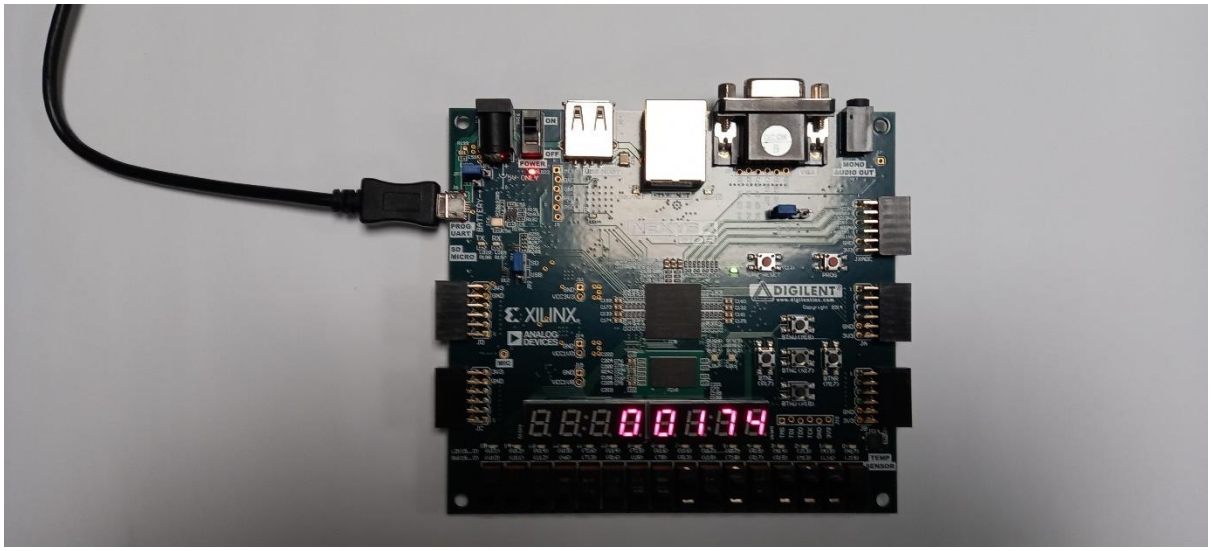
Le module bin2bcd est fonctionnel avec un registre en entrée et en sortie du module.

Nous pouvons passer à l'implémentation.

Démonstration

Je réalise une démonstration sur carte Nexys4DDR. Le mot binaire : 0b1010_1110 soit AE en hexadécimal. J'obtiens 174 sur l'afficheur 7 segment. L'architecture est fonctionnelle.





Comparaison théorie et pratique

L'objectif de cette partie est de comparer vos estimations théoriques (délai et ressources) et les résultats d'implémentation.
Dans un premier temps, je réalise une implémentation de test.vhd avec une contrainte de 10 ns et N = 8.

Cas 10 ns et N = 8

Rapport de synthèse

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
   2 Input    4 Bit       Adders := 10
+---Registers :
               12 Bit    Registers := 1
                8 Bit    Registers := 1
+---Muxes :
   2 Input   12 Bit        Muxes := 1
   2 Input    4 Bit        Muxes := 9
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
```

Vivado informe qu'il y a 10 additionneurs de 4 bits avec deux entrées.
L'outil de synthèse informe la présence d'un registre de 8 bits (correspondant data_in) et d'un registre de 12 bits (correspondant data_out).
Vivado nous informe de 9 multiplexeurs de 4 bits possédant deux entrées et d'un multiplexeur de 12 bits possédant deux entrées.

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |BUFG |     1|
|2     |LUT1 |     1|
|3     |LUT5 |     7|
|4     |LUT6 |     5|
|5     |FDCE |    18|
|6     |IBUF |    10|
|7     |OBUF |    12|
+------+-----+------+
```

Il n'y a aucun latch.

Relevé des ressources

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	10	0	0	63400	0.02
LUT as Logic	10	0	0	63400	0.02
LUT as Memory	0	0	0	19000	0.00
Slice Registers	18	0	0	126800	0.01
Register as Flip Flop	18	0	0	126800	0.01
Register as Latch	0	0	0	126800	0.00
F7 Muxes	0	0	0	31700	0.00
F8 Muxes	0	0	0	15850	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
18	Yes	-	Reset
0	Yes	Set	-
0	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice	4	0	0	15850	0.03
SLICEL	4	0			
SLICEM	0	0			
LUT as Logic	10	0	0	63400	0.02
using O5 output only	0				
using O6 output only	7				
using O5 and O6	3				
LUT as Memory	0	0	0	19000	0.00
LUT as Distributed RAM	0	0			
LUT as Shift Register	0	0			
Slice Registers	18	0	0	126800	0.01
Register driven from within the Slice	9				
Register driven from outside the Slice	9				
LUT in front of the register is unused	5				
LUT in front of the register is used	4				
Unique Control Sets	1		0	15850	<0.01

Relevé des ressources :

- 4 slices
- 10 LUTS
- 18 DFF

3. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	135	0.00
RAMB36/FIFO*	0	0	0	135	0.00
RAMB18	0	0	0	270	0.00

4. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	240	0.00

5. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	22	0	0	210	10.48
IOB Master Pads	11				
IOB Slave Pads	10				
Bonded IPADs	0	0	0	2	0.00
PHY_CONTROL	0	0	0	6	0.00
PHASER_REF	0	0	0	6	0.00
OUT_FIFO	0	0	0	24	0.00
IN_FIFO	0	0	0	24	0.00
IDELAYCTRL	0	0	0	6	0.00
IBUFDS	0	0	0	202	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	24	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	24	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	300	0.00
ILOGIC	0	0	0	210	0.00
OLOGIC	0	0	0	210	0.00

```
6. Clocking
-----------

+------------+------+-------+------------+-----------+-------+
|  Site Type | Used | Fixed | Prohibited | Available | Util% |
+------------+------+-------+------------+-----------+-------+
| BUFGCTRL   |    1 |     0 |          0 |        32 |  3.13 |
| BUFIO      |    0 |     0 |          0 |        24 |  0.00 |
| MMCME2_ADV |    0 |     0 |          0 |         6 |  0.00 |
| PLLE2_ADV  |    0 |     0 |          0 |         6 |  0.00 |
| BUFMRCE    |    0 |     0 |          0 |        12 |  0.00 |
| BUFHCE     |    0 |     0 |          0 |        96 |  0.00 |
| BUFR       |    0 |     0 |          0 |        24 |  0.00 |
+------------+------+-------+------------+-----------+-------+
```

Relevé des ressources :

- 0 block RAM
- 0 DSP
- 1 arbre d'horloge (BUFGCTRL)
- 0 PLL

Analyse du rapport de timings

```
------------------------------------------------------------------------|
Design Timing Summary
| ---------------------
------------------------------------------------------------------------

WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
-------      -------  ---------------------  -------------------
6.725        0.000                        0                   10

WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
-------      -------  ---------------------  -------------------
0.162        0.000                        0                   10

WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
--------     --------  ----------------------  --------------------
4.500        0.000                         0                    19

------------------------------------------------------------------------------
| Clock Summary
| -------------
------------------------------------------------------------------------------

Clock        Waveform(ns)       Period(ns)      Frequency(MHz)
-----        ------------       ----------      --------------
sys_clk_pin  {0.000 5.000}      10.000          100.000
```

Je retrouve la contrainte que j'ai spécifié dans le fichier de contrainte .xdc.

```
------------------------------------------------------------------------|
Intra Clock Table
| -----------------
------------------------------------------------------------------------
Clock             WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
-----             -------      -------  --------------------  -------------------
sys_clk_pin         6.725        0.000                     0                   10

WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
-------      -------  --------------------  -------------------
0.162        0.000                     0                   10

WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
--------     --------  ----------------------  --------------------
4.500        0.000                      0                    19
```

<u>Relevé :</u>

Slack = WNS = 7,476 ns

WNS ≥ 0 et WHS ≥ 0

<u>Calcul fréquence maximale :</u>

$$f_{max} = \frac{1}{period - WNS} = \frac{1}{(10 - 6.725) * 10^{-9}} = 305MHz$$

STA : chemin critique

Setup :

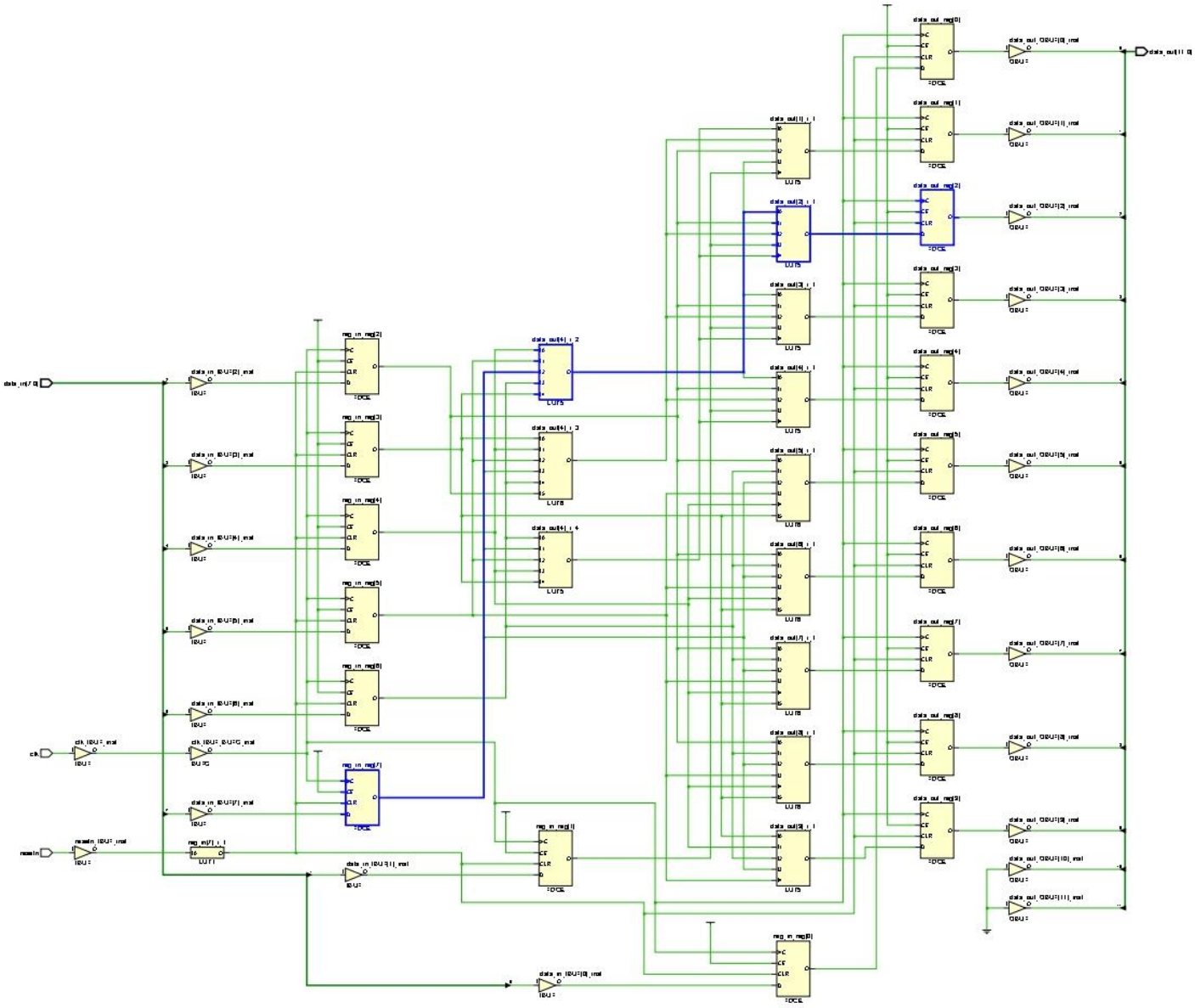
```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             6.725ns  (required time - arrival time)
  Source:                 reg_in_reg[7]/C
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin
{rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            data_out_reg[2]/D
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin
{rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             sys_clk_pin
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (sys_clk_pin rise@10.000ns - sys_clk_pin rise@0.000ns)
  Data Path Delay:        3.245ns  (logic 0.937ns (28.873%)  route 2.308ns (71.127%))
  Logic Levels:           2  (LUT5=2)
  Clock Path Skew:        -0.025ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.418ns = ( 14.418 - 10.000 )
    Source Clock Delay      (SCD):    4.782ns
    Clock Pessimism Removal (CPR):    0.339ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                       (clock sys_clk_pin rise edge)
                                                    0.000     0.000 r
    N15                                             0.000     0.000 r  clk (IN)
                       net (fo=0)                   0.000     0.000    clk
    N15                IBUF (Prop_ibuf_I_O)         0.948     0.948 r  clk_IBUF_inst/O
                       net (fo=1, routed)           2.016     2.963    clk_IBUF
    BUFGCTRL_X0Y0      BUFG (Prop_bufg_I_O)         0.096     3.059 r  clk_IBUF_BUFG_inst/O
                       net (fo=18, routed)          1.722     4.782    clk_IBUF_BUFG
    SLICE_X1Y57        FDCE                                       r  reg_in_reg[7]/C
  -------------------------------------------------------------------    -------------------
    SLICE_X1Y57        FDCE (Prop_fdce_C_Q)         0.456     5.238 r  reg_in_reg[7]/Q
                       net (fo=8, routed)           1.176     6.414    reg_in[7]
    SLICE_X0Y58        LUT5 (Prop_lut5_I2_O)        0.154     6.568 r  data_out[4]_i_2/O
                       net (fo=4, routed)           1.132     7.700    data_out[4]_i_2_n_0
    SLICE_X0Y58        LUT5 (Prop_lut5_I0_O)        0.327     8.027 r  data_out[2]_i_1/O
                       net (fo=1, routed)           0.000     8.027    res_out[2]
    SLICE_X0Y58        FDCE                                       r  data_out_reg[2]/D
  -------------------------------------------------------------------    -------------------
```

```
                    (clock sys_clk_pin rise edge)
                                                 10.000    10.000 r
  N15                                             0.000    10.000 r  clk (IN)
                    net (fo=0)                    0.000    10.000    clk
  N15               IBUF (Prop_ibuf_I_O)          0.814    10.814 r  clk_IBUF_inst/O
                    net (fo=1, routed)            1.911    12.725    clk_IBUF
  BUFGCTRL_X0Y0     BUFG (Prop_bufg_I_O)          0.091    12.816 r  clk_IBUF_BUFG_inst/O
                    net (fo=18, routed)           1.601    14.418    clk_IBUF_BUFG
  SLICE_X0Y58       FDCE                                         r  data_out_reg[2]/C
                    clock pessimism               0.339    14.757
                    clock uncertainty            -0.035    14.721
  SLICE_X0Y58       FDCE (Setup_fdce_C_D)         0.031    14.752    data_out_reg[2]
-------------------------------------------------------------------
                      required time                         14.752
                      arrival time                          -8.027
-------------------------------------------------------------------
                      slack                                  6.725
```

Le chemin critique se trouve entre le bit numéro 7 du registre reg_in et le bit numéro 2 du registre data_out.
Ici, il traverse deux LUT5.

Schématique du chemin critique :



Hold :

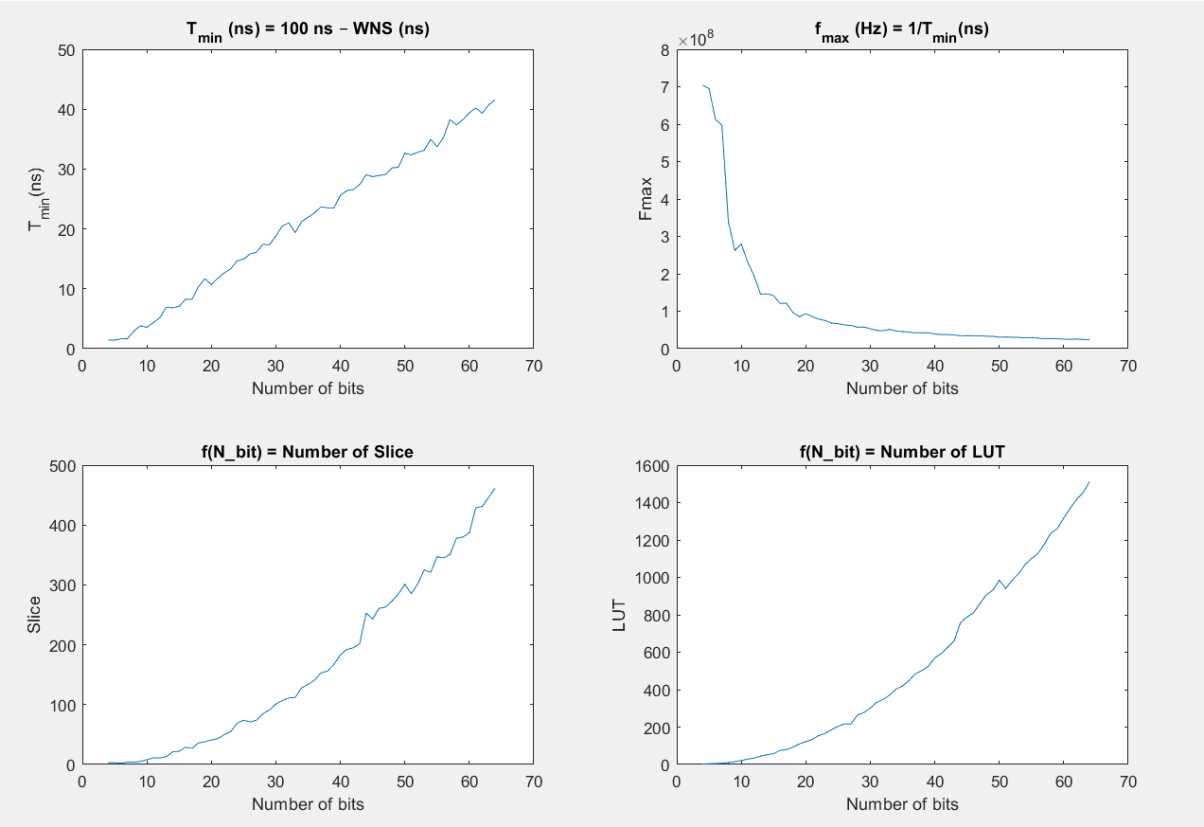
```
Min Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.162ns  (arrival time - required time)
  Source:                 reg_in_reg[3]/C
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns
fall@5.000ns period=10.000ns})
  Destination:            data_out_reg[9]/D
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns
fall@5.000ns period=10.000ns})
  Path Group:             sys_clk_pin
  Path Type:              Hold (Min at Fast Process Corner)
  Requirement:            0.000ns  (sys_clk_pin rise@0.000ns - sys_clk_pin rise@0.000ns)
  Data Path Delay:        0.267ns  (logic 0.186ns (69.735%)  route 0.081ns (30.265%))
  Logic Levels:           1  (LUT5=1)
  Clock Path Skew:        0.013ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    1.963ns
    Source Clock Delay      (SCD):    1.443ns
    Clock Pessimism Removal (CPR):    0.506ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         (clock sys_clk_pin rise edge)
                                                    0.000     0.000 r
    N15                                             0.000     0.000 r  clk (IN)
                         net (fo=0)                 0.000     0.000    clk
    N15                  IBUF (Prop_ibuf_I_O)       0.177     0.177 r  clk_IBUF_inst/O
                         net (fo=1, routed)         0.640     0.817    clk_IBUF
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)       0.026     0.843 r  clk_IBUF_BUFG_inst/O
                         net (fo=18, routed)        0.601     1.443    clk_IBUF_BUFG
    SLICE_X1Y57          FDCE                                   r  reg_in_reg[3]/C
  -------------------------------------------------------------------    -------------------
    SLICE_X1Y57          FDCE (Prop_fdce_C_Q)       0.141     1.584 r  reg_in_reg[3]/Q
                         net (fo=8, routed)         0.081     1.665    reg_in[3]
    SLICE_X0Y57          LUT5 (Prop_lut5_I0_O)      0.045     1.710 r  data_out[9]_i_1/O
                         net (fo=1, routed)         0.000     1.710    data_out[9]_i_1_n_0
    SLICE_X0Y57          FDCE                                   r  data_out_reg[9]/D
  -------------------------------------------------------------------    -------------------

                         (clock sys_clk_pin rise edge)
                                                    0.000     0.000 r
    N15                                             0.000     0.000 r  clk (IN)
                         net (fo=0)                 0.000     0.000    clk
    N15                  IBUF (Prop_ibuf_I_O)       0.365     0.365 r  clk_IBUF_inst/O
                         net (fo=1, routed)         0.695     1.060    clk_IBUF
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)       0.029     1.089 r  clk_IBUF_BUFG_inst/O
                         net (fo=18, routed)        0.874     1.963    clk_IBUF_BUFG
    SLICE_X0Y57          FDCE                                   r  data_out_reg[9]/C
                         clock pessimism           -0.506     1.456
    SLICE_X0Y57          FDCE (Hold_fdce_C_D)       0.092     1.548    data_out_reg[9]
  -------------------------------------------------------------------
                         required time                        -1.548
                         arrival time                          1.710
  -------------------------------------------------------------------
                         slack                                 0.162
```

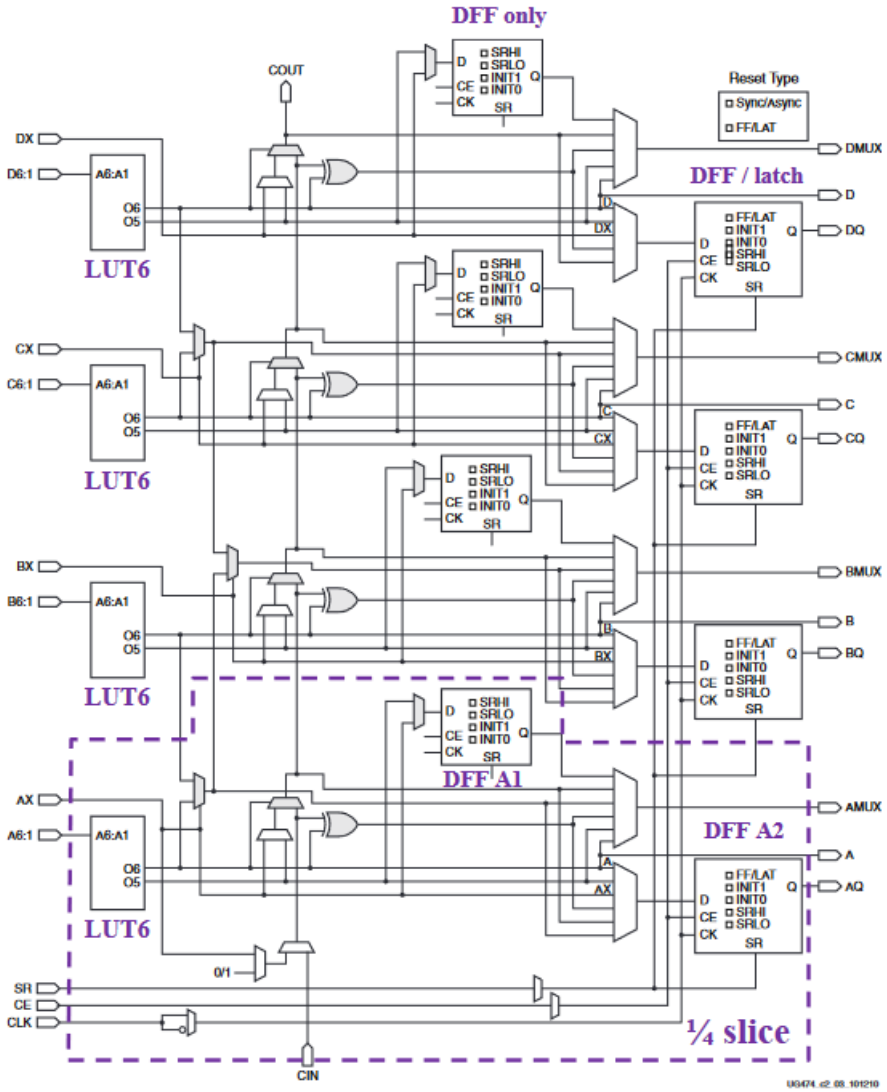
Je réalise des implémentations successives en utilisant le script test.sh. Le script incrémente de 1 la valeur N qui commence à 4 et fini à 64. Ce nombre correspond au nombre de bits de data_in. La contrainte sur l'horloge est égale à 100 ns.

Je trace T_{min} (ns) en fonction du nombre de bit. La fréquence maximale f_{max} (Hz) en fonction du nombre de bit. Le nombre de slice en fonction du nombre de bit et le nombre de LUT en fonction du nombre de bit.



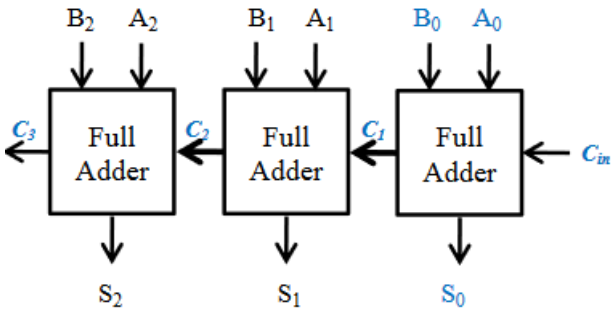
Etant donné que notre architecture utilise un bloc élémentaire combinatoire, il est nécessaire d'étudier le slice de type L.

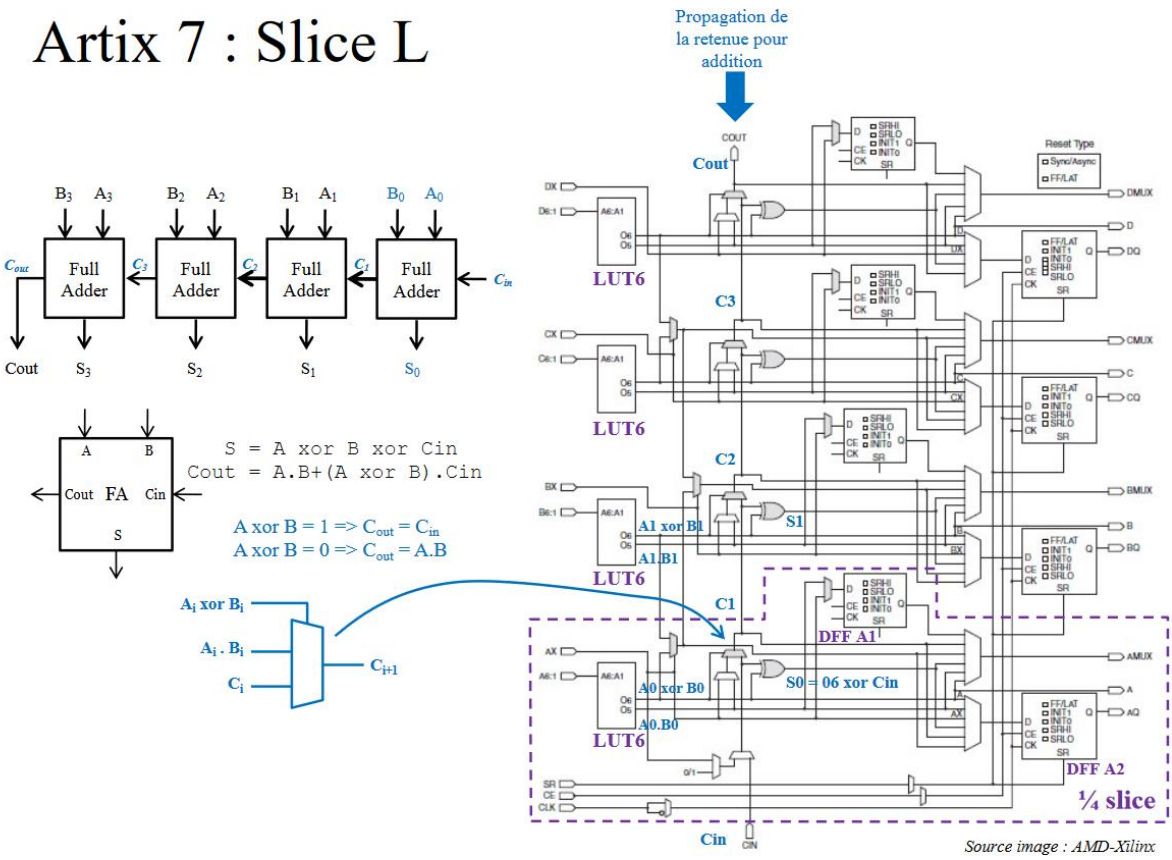
Xilinx Artix 7 : Slice L



Source image : AMD-Xilinx

Le bloc élémentaire de notre architecture utilise 3 Full Adder.





<u>Référence :</u> Cours SEI-4201A FPGA – Madame EXERTIER

Dans un quart de slice, il y a un LUT 6 au sein duquel se trouvent deux LUT 5. On peut également trouver un Full Adder. Ainsi, dans une Slice L, il y a quatre Full Adders et quatre LUT 6.

Rappelons que mon bloc élémentaire nécessite trois Full Adders. En théorie, le bloc élémentaire va utiliser ¾ d'un slice L.

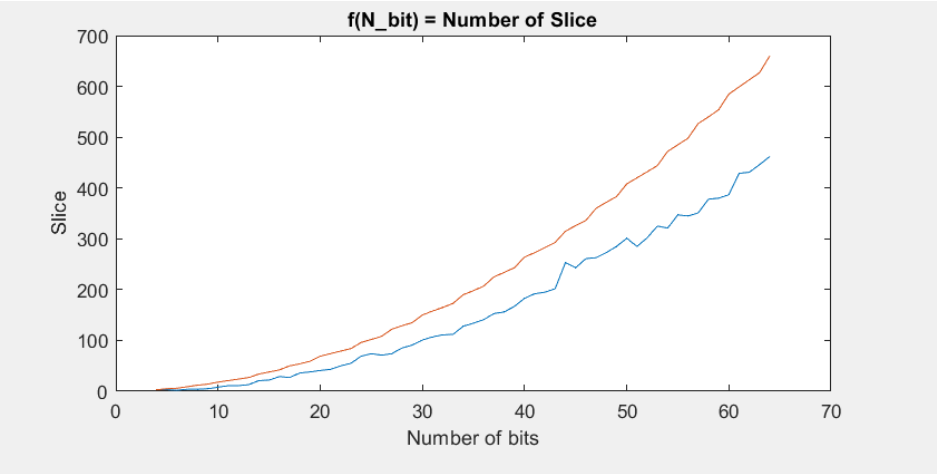
Ainsi, pour obtenir le nombre théorique de slices utilisés, il me suffit de multiplier la profondeur de mon architecture par le nombre de blocs qu'il y a sur une ligne, ce qui me donne le nombre total de blocs élémentaires dans mon architecture. Ensuite, il faut multiplier ce dernier par ¾. J'obtiens finalement le nombre théorique total de slices dans mon architecture.

Pour obtenir le nombre théorique de LUT utilisé

À l'aide de Matlab, je trace une courbe théorique et une courbe issue des fichiers d'implémentation de Vivado.

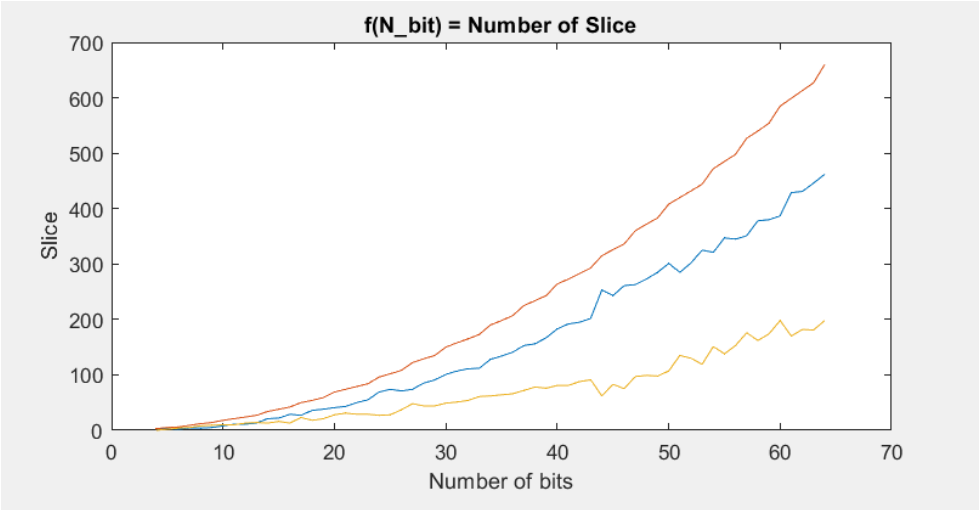
J'obtiens la courbe suivante. La courbe bleue représente les valeurs d'implémentation du nombre de slice en fonction du nombre de bit de data_in

tandis que la courbe orange correspond aux nombres de slice théorique en fonction du nombre de bits sur data_in.



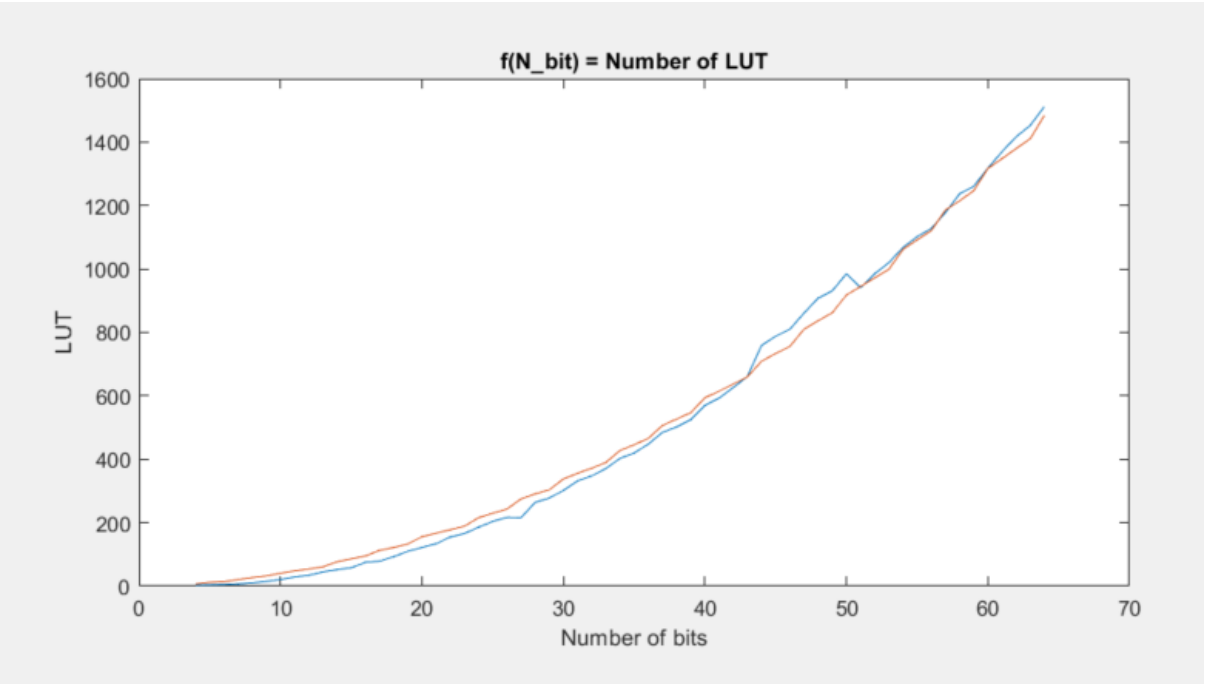
Nous pouvons relever que les courbes évoluent de manière quadratique.

On peut s'intéressé à la différence entre le théorique et l'implémentation effectué par Vivado. La courbe jaune correspond à cette différence.



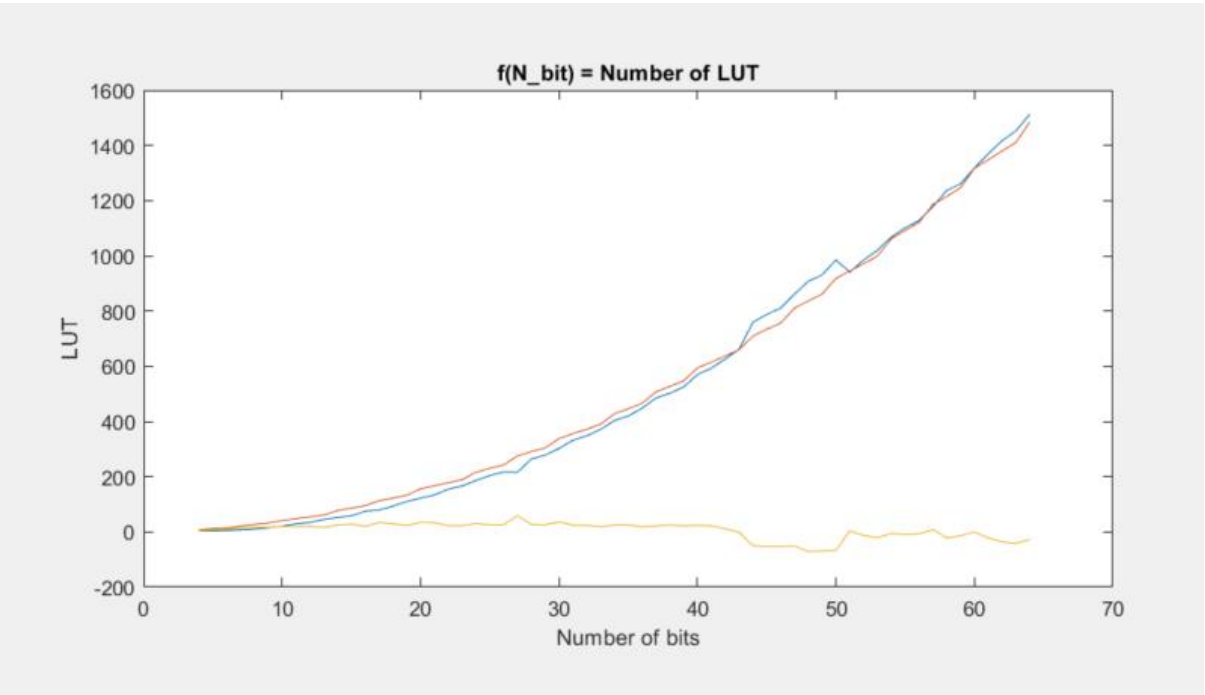
Lorsque le nombre de slice approche le nombre de bits égale à 64, l'écart entre le nombre théorique de slice nécessaires et le nombre réellement utilisé devient notable. Cela suggére que mon architecture théorique n'est pas optimisée. En conséquence, l'outil de synthèse a réussi à optimiser mon architecture en supprimant des blocs élémentaires. Il y a un rapport entre la valeur théorique et la valeur d'implémentation égale en moyenne sur cette série à 2.45.

Dans cette section, je vais m'interréssé au nombre de LUT logique en fonction du nombre de bit. J'obtiens la courbe suivante.



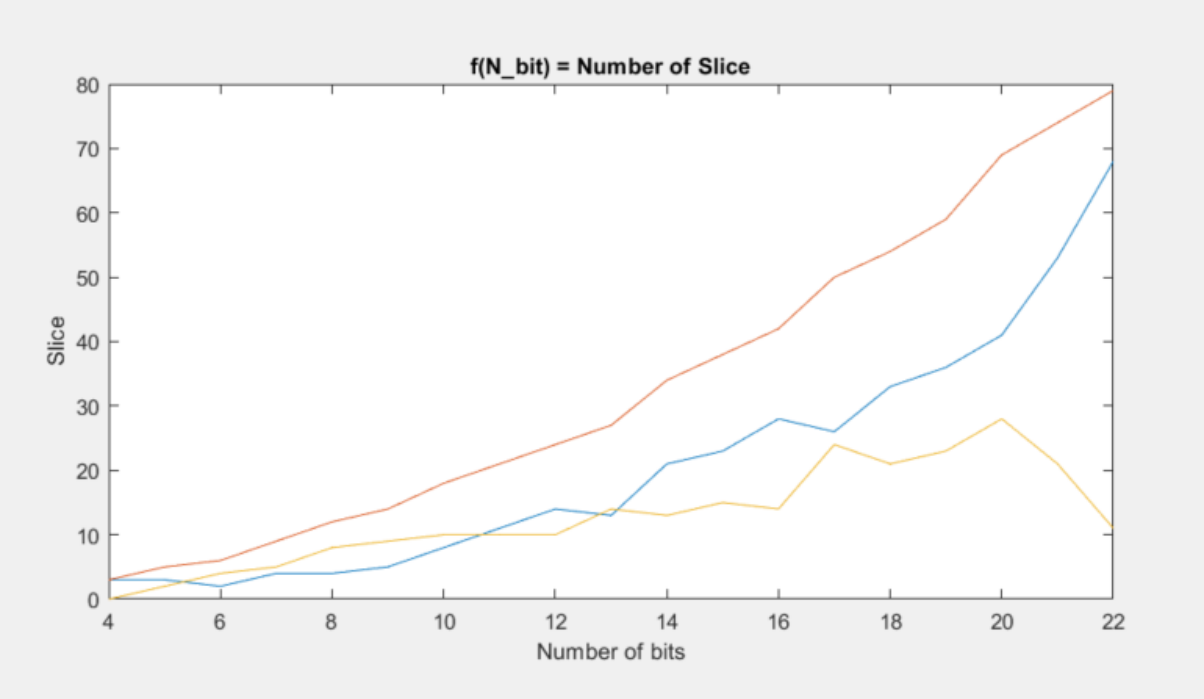
La courbe bleue correspond au résultat lors de l'implémentation du nombre de LUT(LUT5 ou LUT6) en fonction du nombre de bit de data_in. La courbe orange représente la valeur théorique du nombre de LUT dans mon architecture. Il y a un rapport entre la valeur théorique et la valeur d'implémentation égale en moyenne sur cette série à 0.55.

Nous pouvons souligner que les deux courbes évoluent de manière quadratique.

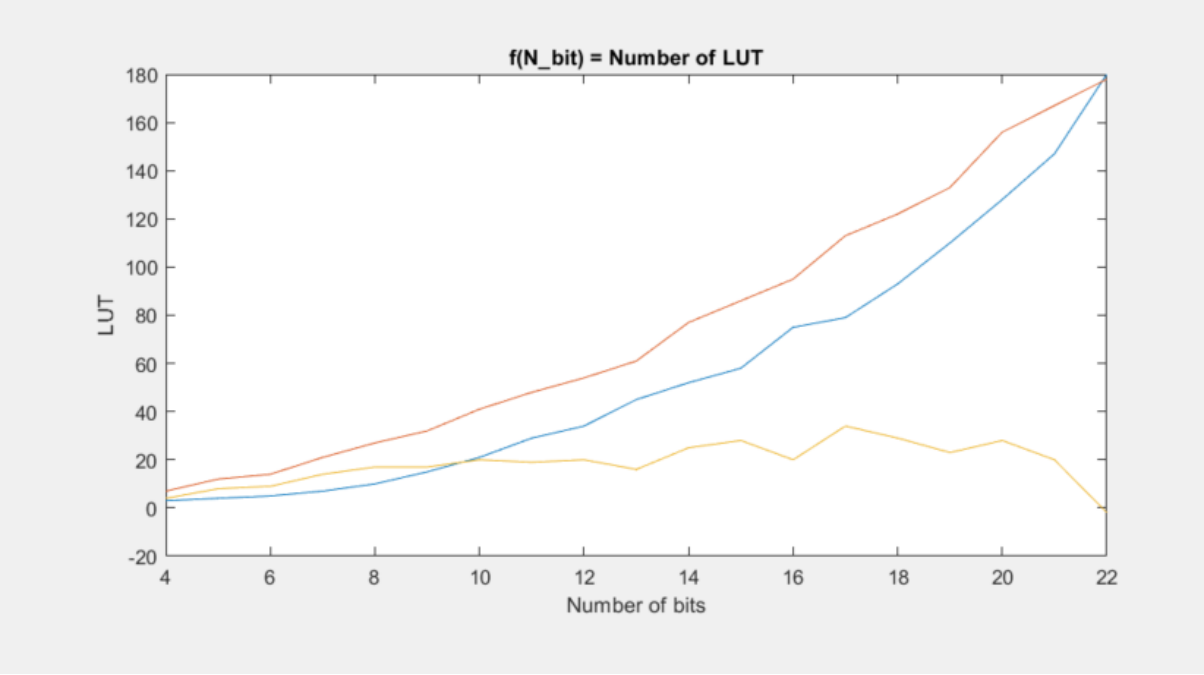


La courbe jaune correspond à la différence entre la courbe théorique et la courbe d'implémentation.

Je change la période de l'horloge à 10 ns. Regardons si nous obtenons les mêmes courbes.



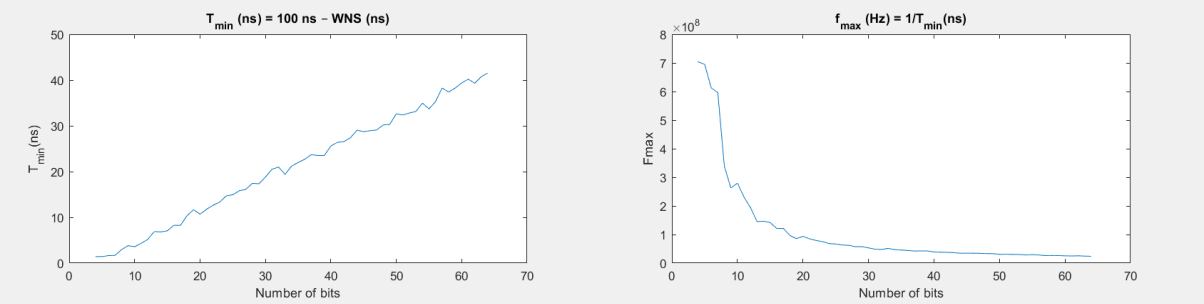
Il y a un rapport entre la valeur théorique et la valeur d'implémentation égale en moyenne sur cette série à 1.88.



Il y a un rapport entre la valeur théorique et la valeur d'implémentation égale en moyenne sur cette série à 2.04.

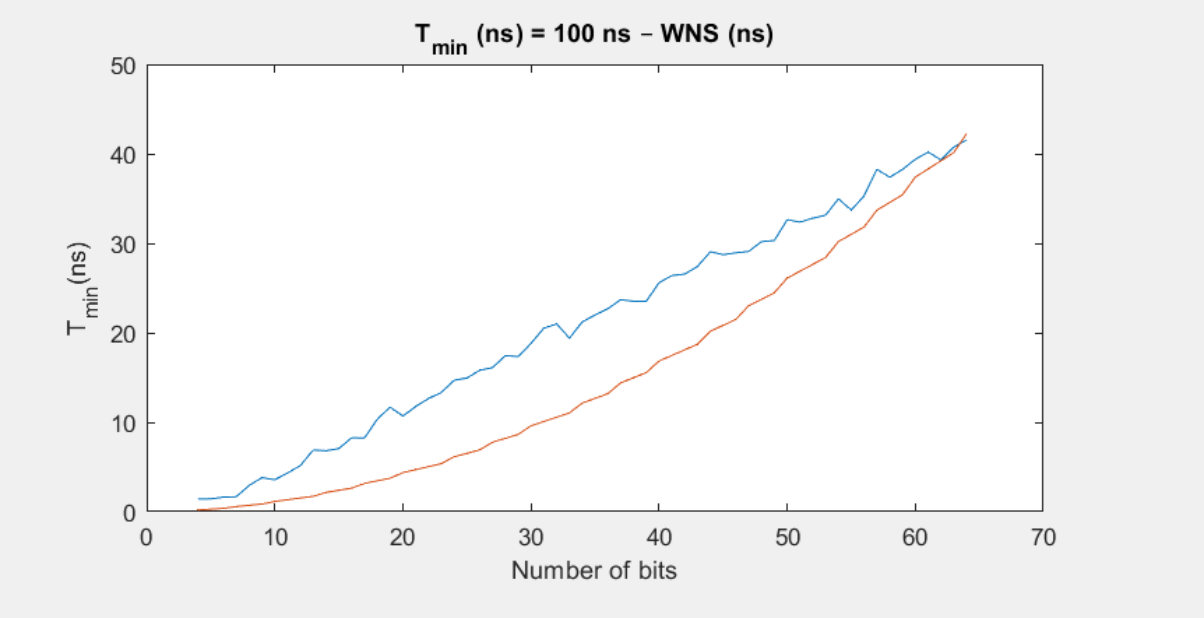
Comme pour une implémentation à 100 ns, l'estimation théorique fournit plus de slice et de LUT. C'est l'outil de synthèse qui va optimiser les ressources.

<u>Pour le délai :</u>



Étant donné que je n'ai pas pu trouver la valeur du temps de propagation (t_{pd}) pour un XOR ou un Full Adder, j'ai entrepris de consulter le rapport *post-route_timing_summary* lorsque N = 4 dans ma conception. Après examen, j'ai identifié le chemin critique qui traverse un LUT3, avec un délai de passage de 0.048 ns. En extrapolant cette information, j'ai présumé que le délai de propagation pour mon bloc élémentaire bin2bcd serait également de 0.048 ns.

Néanmoins, cette information rend le modèle théorique simpliste. Nous omettons le délai de routage. De plus, l'outil de synthèse va tenter d'optimiser l'architecture ce qui peux affecter le temps de propagation.



La courbe bleue représente le résultat d'implémentation pour T_{min}(ns) en fonction du nombre de bits. La courbe orange est le modèle théorique. Je remarque que la courbe bleue augmente au fur et mesure que le nombre de bit augmente. Cela est dû au délai de routage et à la logique traversée dans l'architecture. La courbe orange semble rejoindre la courbe bleue aux extrémités. Cependant, elle ne se rapproche pas de la courbe d'implémentation.

Conclusion :

L'outil de synthèse a permis de réduire le nombre de Slices et de LUTs inutilisés par notre architecture. Cela nous a permis de réaliser des économies en ressources matérielles. Par conséquent, nous avons pu augmenter notre temps de développement sans avoir à passer du temps sur une optimisation de notre algorithme qui aurait pu entraîner des erreurs de développement. D'autre part, il a été difficile de caractériser le délai, car l'outil de synthèse prend en compte le délai de routage et diverses optimisations.

FIN