TP PWM

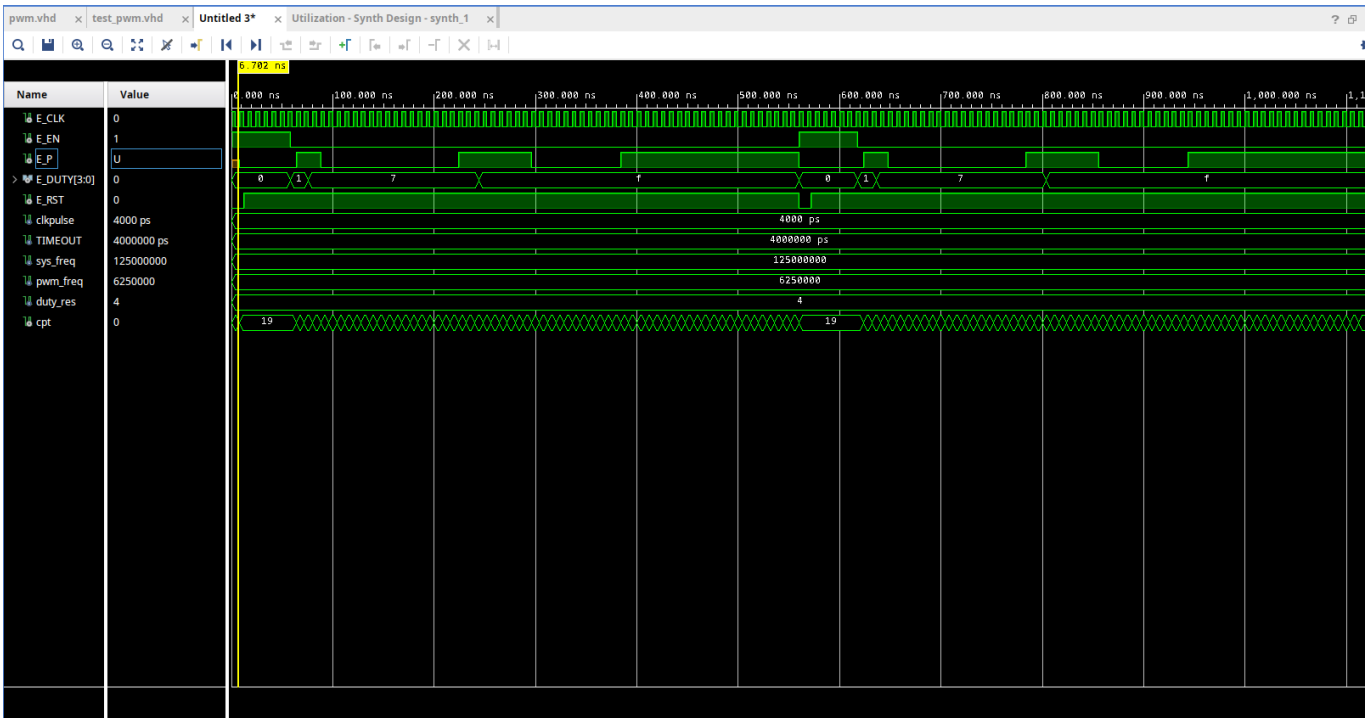
Partie 1 - Développement et implémentation du PWM

Développement, simulation comportementale

Pas grand chose à dire sur le développement du code lui-même, vous pouvez trouver mon code dans le fichier pwm.vhd.

Pour la simulation comportementale, on laisse les lignes 61-65 décommentées et les lignes 66-71 commentées dans le source de test (tst_pwm.vhd) : en l'absence de fichier tcl, il nous faut hardcoder les paramètres génériques.

Résultat de la simul comportementale :



Synthèse

On passe maintenant à la synthèse. On va l'éffectuer deux fois : une avec des paramètres qui facilitent l'analyse la simulation post-synthèse, et une fois avec les "bons" paramètres pour le cas réel.

On ajoute un fichier tcl, dont le contenu sera donc différent pour les 2 synthèses

Pour la simulation post-synthèse : `set_property generic {SYS_CLK=125000000 PWM_FREQ=6250000 DUTY_RES=4} [current_fileset]`.

La sortie de la synthèse nous confirme bien nos valeurs

```
INFO: [Synth 8-638] synthesizing module 'pwm'
[/nfs/home/camsi13/Documents/univ-documents/M2/SE/VHDL/TP1/pwm.vhd:40]
    Parameter sys_clk bound to: 125000000 - type: integer
```

```
	Parameter pwm_freq bound to: 6250000 - type: integer
	Parameter duty_res bound to: 4 - type: integer
```

Rapport de synthèse :

```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   15 |     0 |          0 |     53200 |  0.03 |
|   LUT as Logic          |   15 |     0 |          0 |     53200 |  0.03 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |   11 |     0 |          0 |    106400 |  0.01 |
|   Register as Flip Flop |   11 |     0 |          0 |    106400 |  0.01 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

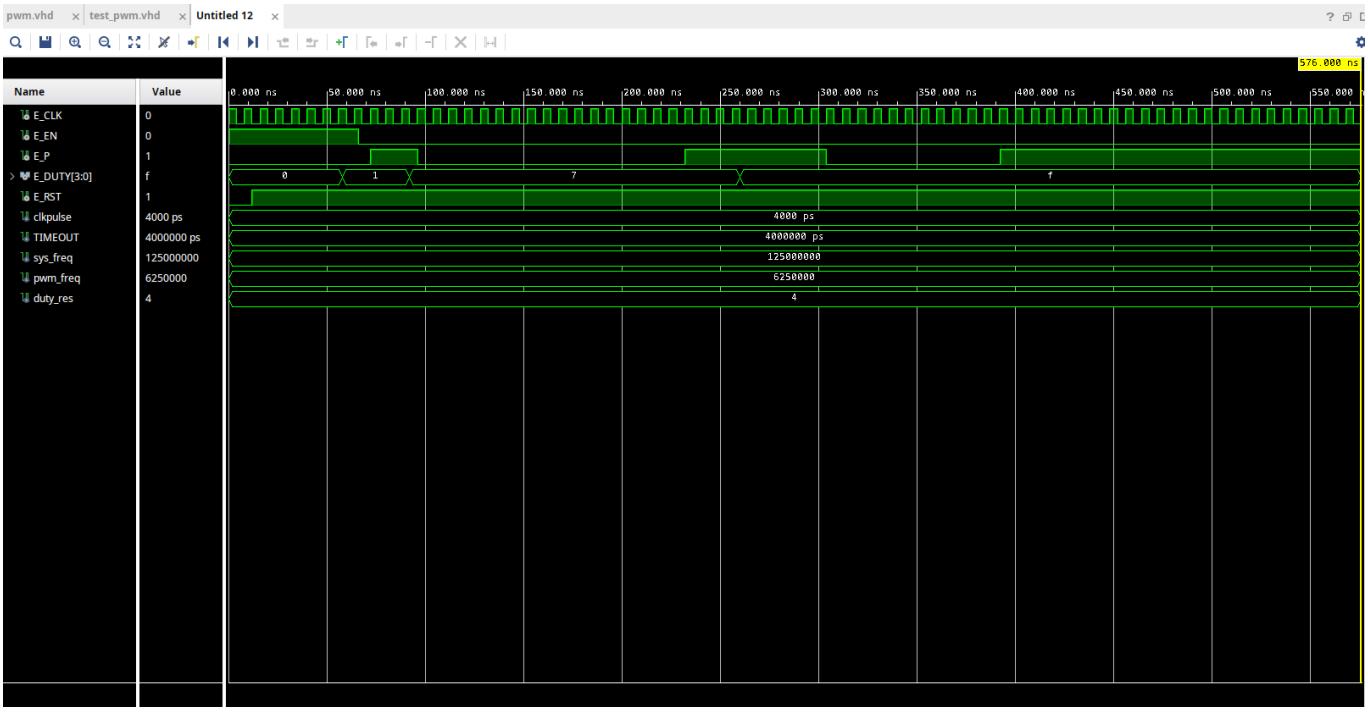
```
+----------+------+----------------------+
| Ref Name | Used | Functional Category |
+----------+------+----------------------+
| FDRE     |    8 |         Flop & Latch |
| LUT6     |    7 |                  LUT |
| IBUF     |    7 |                   IO |
| LUT2     |    4 |                  LUT |
| LUT4     |    3 |                  LUT |
| LUT1     |    3 |                  LUT |
| FDSE     |    3 |         Flop & Latch |
| LUT5     |    2 |                  LUT |
| OBUF     |    1 |                   IO |
| LUT3     |    1 |                  LUT |
| BUFG     |    1 |                Clock |
+----------+------+----------------------+
```

Pour simuler, on inverse les lignes commentées/décommentée dans test_pwm.vhdl : cette fois, on ne hardcode plus les génériques, et on doit mapper les ports.

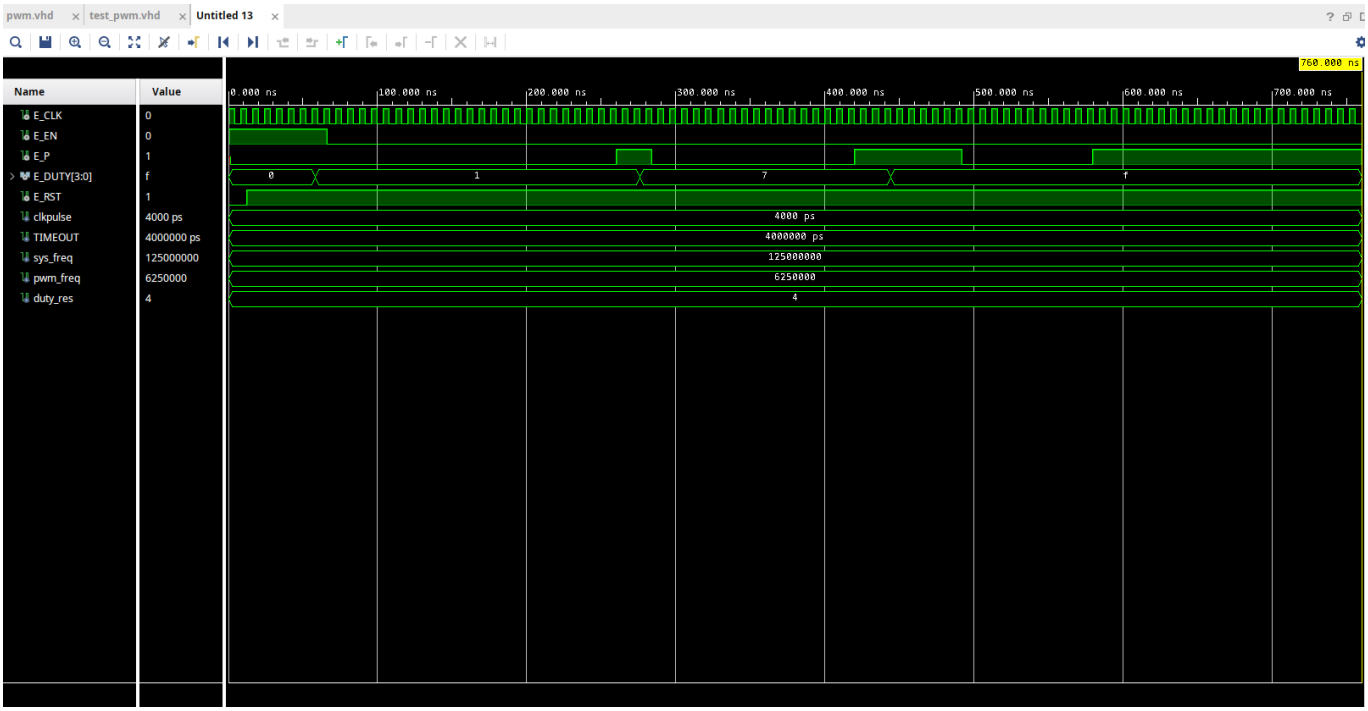
Après un premier essai raté pour cause d'utilisation des mauvaises valeurs génériques ...

```
Running: /nfs/xilinx/Vivado/2023.1/bin/unwrapped/lnx64.o/xelab --incr --debug typical --relax --mt 8 --maxdelay -L xil_defaultlib -L simprims_ver -L secure
Using 8 slave threads.
Starting static elaboration
Pass Through NonSizing Optimizer
Completed static elaboration
INFO: [XSIM 43-4323] No Change in HDL. Linking previously generated obj files to create kernel
INFO: [USF-XSim-69] 'elaborate' step finished in '1' seconds
Time resolution is 1 ps
relaunch_sim: Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB): peak = 9557.586 ; gain = 0.000 ; free physical = 483 ; free virtual = 16755
run 10 s
Failure: SIMULATION TIMEOUT!!!
Time: 4 us  Iteration: 0  Process: /test_pwm/P_TIMEOUT  File: /nfs/home/camsi13/Documents/univ-documents/M2/SE/VHDL/TP1/test_pwm.vhd
$finish called at time : 4 us : File "/nfs/home/camsi13/Documents/univ-documents/M2/SE/VHDL/TP1/test_pwm.vhd" Line 57
```

... on arrive au bon résultat, pour la simulation post-synthèse FONCTIONNELLE



On passe maintenant à la simulation de timing post-synthèse :



... qui nous donne n'importe quoi, ce n'est pas du tout le comportement attendu. A l'heure actuelle, je n'ai toujours pas réussi à trouver la raison ni une solution, sur conseil de l'encadrant du TP j'ai décidé de simplement continuer.

On passe maintenant à la "vraie" synthèse : on change les valeurs dans le TCL : `set_property generic {SYS_CLK=125000000 PWM_FREQ=1 DUTY_RES=4} [current_fileset]`.

Là aussi on confirme les valeurs avec la sortie :

```
INFO: [Synth 8-638] synthesizing module 'pwm'
[/nfs/home/camsi13/Documents/univ-documents/M2/SE/VHDL/TP1/pwm.vhd:40]
    Parameter sys_clk bound to: 125000000 - type: integer
    Parameter pwm_freq bound to: 1 - type: integer
    Parameter duty_res bound to: 4 - type: integer
```

On remarque que le rapport de synhèse au niveau de certaines valeurs, assez drastiquement :

```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   46 |     0 |          0 |     53200 |  0.09 |
|   LUT as Logic          |   46 |     0 |          0 |     53200 |  0.09 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |   55 |     0 |          0 |    106400 |  0.05 |
|   Register as Flip Flop |   55 |     0 |          0 |    106400 |  0.05 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

```
+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDSE     |   36 |        Flop & Latch |
| LUT4     |   28 |                 LUT |
| LUT2     |   26 |                 LUT |
| FDRE     |   19 |        Flop & Latch |
| CARRY4   |   11 |          CarryLogic |
| LUT3     |    8 |                 LUT |
| IBUF     |    7 |                  IO |
| LUT1     |    4 |                 LUT |
| LUT6     |    3 |                 LUT |
| OBUF     |    1 |                  IO |
| LUT5     |    1 |                 LUT |
| DSP48E1  |    1 |    Block Arithmetic |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+
```

Cela est du notament au fait que le compteur interne du PWM devient bien plus large.

Pour finir, on s'intéresse aux reports que vivado nous donne après la synthèse avec ces valeurs.

Design Timing Summary

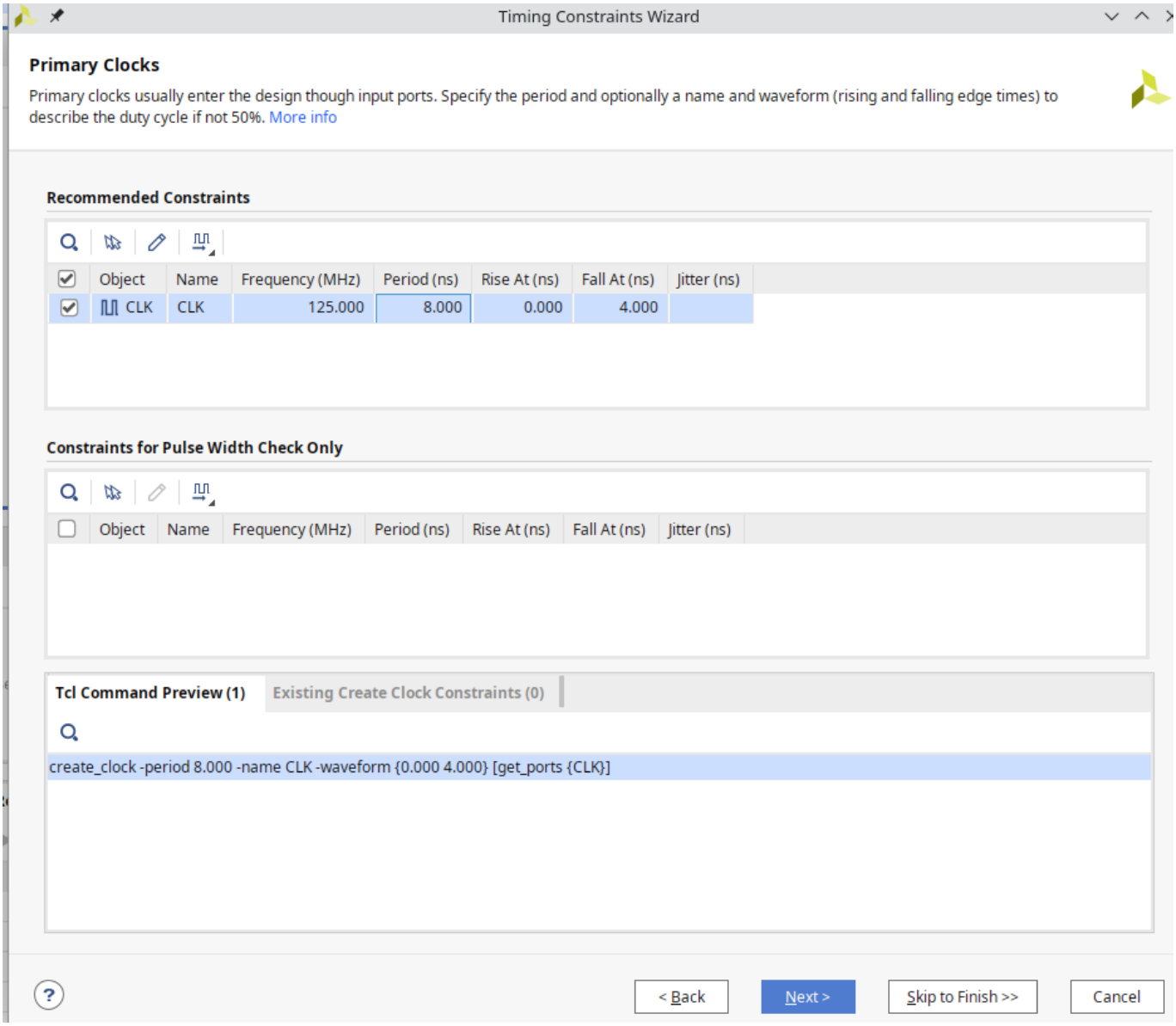
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.281 ns	Worst Hold Slack (WHS):	0.211 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	16	Total Number of Endpoints:	16	Total Number of Endpoints:	12

All user specified timing constraints are met.

(à noter que le slack donné ici prend en compte le délai dans les gates, mais pas les interconnexions, qui n'ont pas encore été calculées)

Implémentation

On ajoute les constaints, via le wizard :



En plus de la contrainte de temps (qui va définir une spécification de clock à laquelle la clock que l'on utilise concrètement au final devra adhérer), o ajoute dans le xdc des indications liant nos ports à des ports hardware réels.

```
# Master Clock timing constraint
create_clock -period 8.000 -name CLK -waveform {0.000 4.000} [get_ports
CLK]
set_property PACKAGE_PIN K17 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property PACKAGE_PIN K18 [get_ports EN]
set_property IOSTANDARD LVCMOS33 [get_ports EN]
set_property PACKAGE_PIN P16 [get_ports RST]
set_property IOSTANDARD LVCMOS33 [get_ports RST]
set_property PACKAGE_PIN G15 [get_ports {DUTY[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[0]}]
set_property PACKAGE_PIN P15 [get_ports {DUTY[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[1]}]
set_property PACKAGE_PIN W13 [get_ports {DUTY[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[2]}]
set_property PACKAGE_PIN T16 [get_ports {DUTY[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DUTY[3]}]
set_property PACKAGE_PIN D18 [get_ports P]
set_property IOSTANDARD LVCMOS33 [get_ports P]
```

Pour l'implémentation, on s'apperçoit que les boutons, que l'on va utiliser pour le reset et le enable, sont en "normalement à 0", c'est à dire que le signal qu'on va recevoir est à 0 si le bouton n'est pas appuyé. Cela impliquerait que le PWM fonctionne tant que le bouton enable n'est pas appuyé, et reset tant que le bouton reset n'est pas appuyé (donc on doit appuyer pour que ça fonctionne, ce qui manque de sens)
On change le code de pwm.vhd pour que le reset n'ait lieu que si le bouton reset est appuyé ("reset à 1").
On pourrait laisser le enable tel quel (auquel cas le PWM fonctionnerait tant que rien n'est appuyé), mais je décide de ne le faire fonctionner que tant que le bouton est appuyé, on passer donc en "enable à 1".

Bistream generation

Il ne reste plus qu'à générer le bitstream et le transférer sur la carte.

Le génération du bitstream se fait sans problème. Je suis juste un peu confus par le log, qui semble indiquer que tous les éléments ont supprimés à l'optimisation :

```
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----
|  Optimization                                       |  Added Cells  |
Removed Cells  |  Optimized Cells/Nets  |  Dont Touch  |  Iterations  |
Elapsed   |
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----
|  LUT Combining                                      |            0  |
16  |                   16  |           0  |           1  |  00:00:00  |
|  Retime                                             |            0  |
0  |                    0  |           0  |           1  |  00:00:00  |
|  Very High Fanout                                   |            0  |
```

```
0  |                    0  |           0  |           1  |  00:00:00  |
|  DSP Register                                        |            0  |
0  |                    0  |           0  |           0  |  00:00:00  |
|  Shift Register to Pipeline                          |            0  |
0  |                    0  |           0  |           0  |  00:00:00  |
|  Shift Register                                      |            0  |
0  |                    0  |           0  |           0  |  00:00:00  |
|  BRAM Register                                       |            0  |
0  |                    0  |           0  |           0  |  00:00:00  |
|  URAM Register                                       |            0  |
0  |                    0  |           0  |           0  |  00:00:00  |
|  Dynamic/Static Region Interface Net Replication     |            0  |
0  |                    0  |           0  |           1  |  00:00:00  |
|  Total                                               |            0  |
16  |                   16  |           0  |           4  |  00:00:00  |
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-----
```

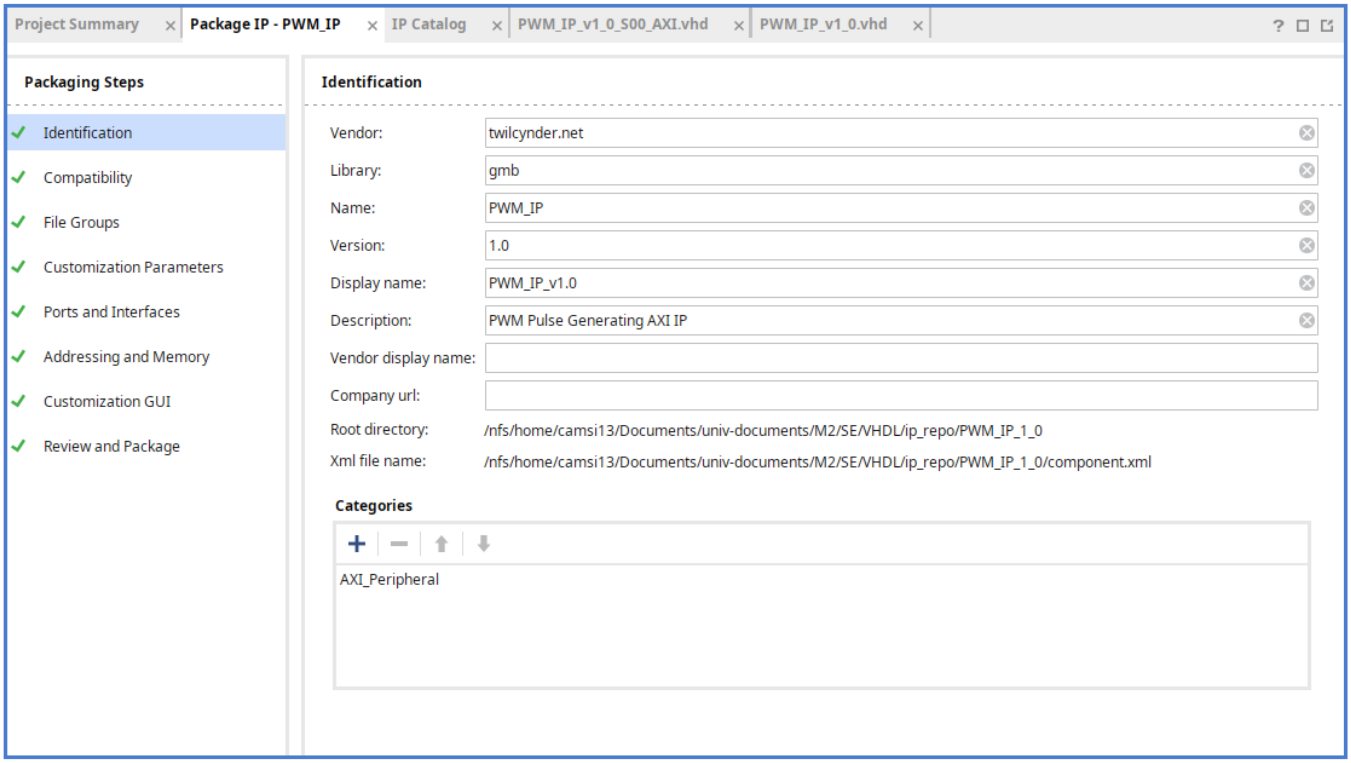
... mais on verra bien si ça marche toujours sur la carte.

Après une petite galère causée par le fait que je n'avais pas réalisé que je n'avais pas branché la carte, le bitstream est transféré : le PWN fonctionne correctement !

La preuve en images

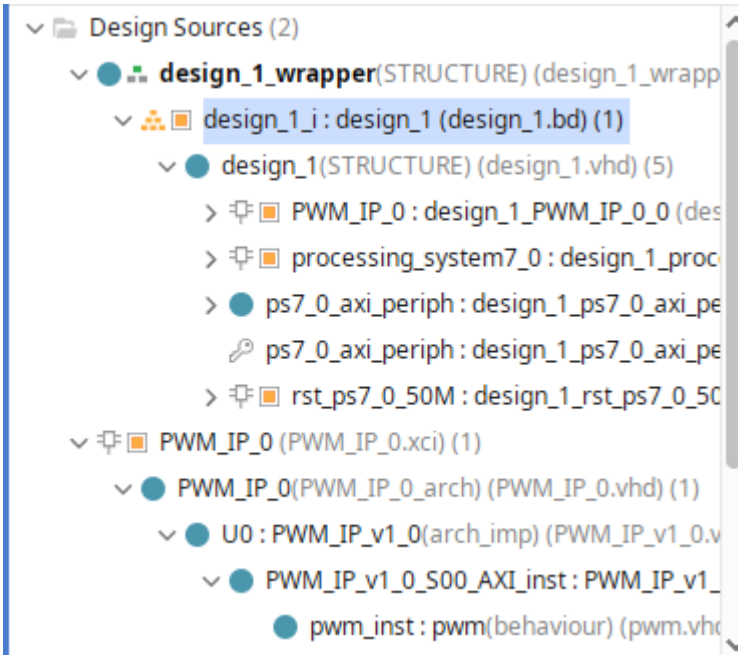
Partie 2 - IP Block

On passe donc à la création d'un bloc IP. Pour cela, je crée d'abord un nouveau projet dans lequel je pourrai utilise le bloc IP (que j'appelle PWM_IP), puis je je le crée en passant par l'option "Create and Package New IP" dans vivado. Je l'appelle PWM_IP (je m'apercevrai un peu plus tard que donner le même nom au projet et au bloc IP était une mauvaise idée, entermes de clarté)

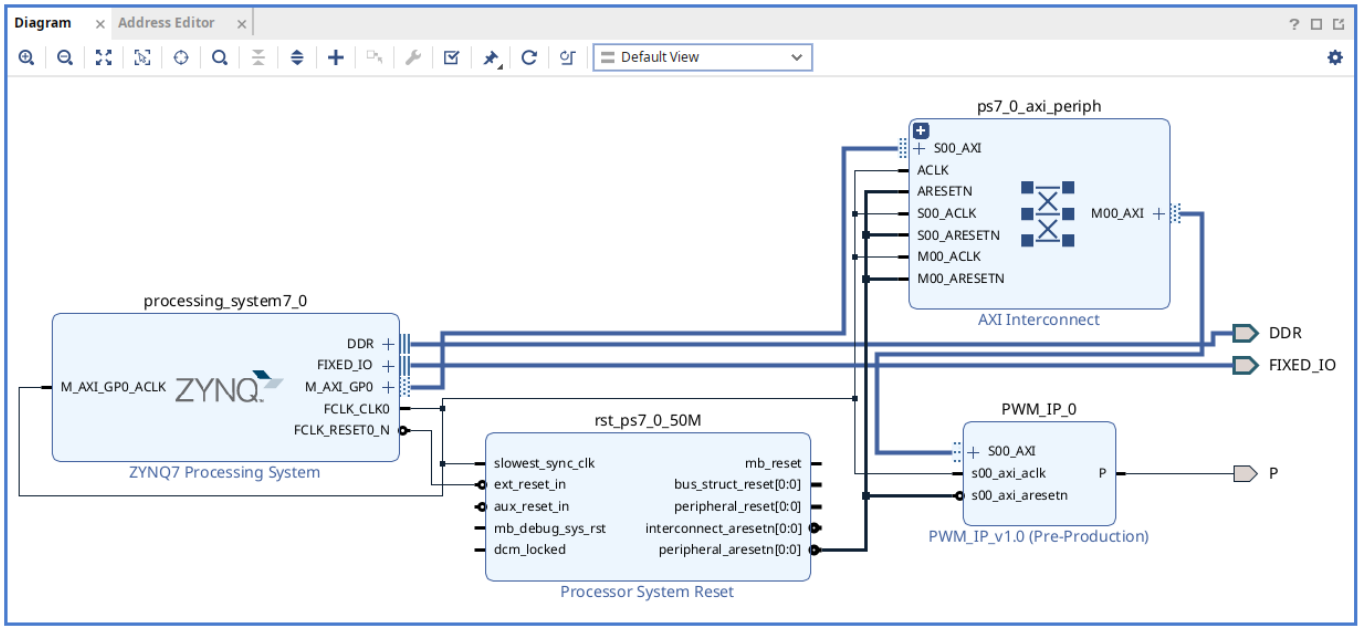


J'ajoute le code source de mon PWN précédemment développé.

Une fois l'IP PWN terminé, on peut repasser au projet PWM_IP, auquel on ajoute le bloc IP.



Block design :



(Il a juste fallu ajouter manuellement le port de sortie "P", connecté à la sortie du bloc IP).

Je lance ensuite l'implémentation, avec le même fichier de contraintes xdc que précédemment, de manière à bien router mes entrées/sorties (notament ma sortie P) aux pins de la carte.

La génération de bitstream se passe normalement ; cette fois au lieu de transférer vers la carte on export en fichier xsa.

Vitis

On lance ensuite l'IDE vitis, afin de pouvoir utiliser mon PWN directement via du code C. J'importe le xsa exporté précédemment en tant que hardware platform dans une nouvelle application (empty c++ app, standalone/bare-metal).
Comme fichier source, j'utilise le myAXIpwm.c fourni.

La première tentative de build ne marche pas : en effet, le fichier aprt du principe que j'ai utilisé le même nom que sur le sujet pour mon bloc IP.

```
Build Console [myPWM_app, Debug]
a9-linaro-pre-build-step

make --no-print-directory main-build
Building file: /nfs/home/camsi13/Downloads/myAXIpwm.c
Invoking: ARM v7 g++ compiler
arm-none-eabi-g++ -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/myAXIpwm.o" -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -I/nfs/h
/nfs/home/camsi13/Downloads/myAXIpwm.c: In function 'int main()':
/nfs/home/camsi13/Downloads/myAXIpwm.c:14:33: error: 'MYAXIPWM_S00_AXI_SLV_REG0_OFFSET' was not declared in this scope; did you
   14 | #define PWM_CTRL_REG            MYAXIPWM_S00_AXI_SLV_REG0_OFFSET
      |                                 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/nfs/home/camsi13/Downloads/myAXIpwm.c:27:32: note: in expansion of macro 'PWM_CTRL_REG'
   27 |         CR = MYAXIPWM_mReadReg(PWM_CTRL_REG, PWM_BASE_ADR);
      |                                ^~~~~~~~~~~~
/nfs/home/camsi13/Downloads/myAXIpwm.c:11:25: error: 'XPAR_MYAXIPWM_0_S00_AXI_BASEADDR' was not declared in this scope; did you
   11 | #define PWM_BASE_ADR    XPAR_MYAXIPWM_0_S00_AXI_BASEADDR
      |                         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/nfs/home/camsi13/Downloads/myAXIpwm.c:27:46: note: in expansion of macro 'PWM_BASE_ADR'
   27 |         CR = MYAXIPWM_mReadReg(PWM_CTRL_REG, PWM_BASE_ADR);
      |                                              ^~~~~~~~~~~~
/nfs/home/camsi13/Downloads/myAXIpwm.c:27:14: error: 'MYAXIPWM_mReadReg' was not declared in this scope; did you mean 'PWM_IP_r
   27 |         CR = MYAXIPWM_mReadReg(PWM_CTRL_REG, PWM_BASE_ADR);
      |              ^~~~~~~~~~~~~~~~
      |              PWM_IP_mReadReg
/nfs/home/camsi13/Downloads/myAXIpwm.c:18:43: warning: '<<' in boolean context, did you mean '<'? [-Wint-in-bool-context]
   18 | #define ENABLE_MASK                     (1<<31)
      |                                           ^
/nfs/home/camsi13/Downloads/myAXIpwm.c:28:23: note: in expansion of macro 'ENABLE_MASK'
   28 |         CR = CR & not(ENABLE_MASK);
      |                       ^~~~~~~~~~~
make[1]: *** [src/subdir.mk:23: src/myAXIpwm.o] Error 1
make: *** [makefile:46: all] Error 2

14:57:09 Build Finished (took 219ms)
```

.

Il faut donc que je corrige tous les "myAXIpwm" en "PWM_IP" (dont au niveau du `#include "myAXIpwm.h"`).

Cette fois le build fonctionne.

```
14:55:24 **** Incremental Build of configuration Debug for project myPWM_app ****
make all
make --no-print-directory pre-build
a9-linaro-pre-build-step

make --no-print-directory main-build
Building file: /nfs/home/camsi13/Downloads/myAXIpwm.c
Invoking: ARM v7 g++ compiler
arm-none-eabi-g++ -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/myAXIpwm.o" -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -I/nfs/h
/nfs/home/camsi13/Downloads/myAXIpwm.c: In function 'int main()':
/nfs/home/camsi13/Downloads/myAXIpwm.c:19:43: warning: '<<' in boolean context, did you mean '<'? [-Wint-in-bool-context]
   19 | #define ENABLE_MASK                     (1<<31)
      |                                           ^
/nfs/home/camsi13/Downloads/myAXIpwm.c:29:23: note: in expansion of macro 'ENABLE_MASK'
   29 |         CR = CR & not(ENABLE_MASK);
      |                       ^~~~~~~~~~~
Finished building: /nfs/home/camsi13/Downloads/myAXIpwm.c

Building target: myPWM_app.elf
Invoking: ARM v7 g++ linker
arm-none-eabi-g++ -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -Wl,-build-id=none -specs=Xilinx.spec -Wl,-T -Wl,../src/lscript
Finished building target: myPWM_app.elf

Invoking: ARM v7 Print Size
arm-none-eabi-size myPWM_app.elf  |tee "myPWM_app.elf.size"
   text    data     bss     dec     hex filename
  30925    2184   22640   55749    d9c5 myPWM_app.elf
Finished building: myPWM_app.elf.size


14:55:24 Build Finished (took 421ms)
```

On n'oublie pas de bouger le bridge sur les pins de configuratiton du Programming Mode de la carte pour passer en JTAG.

Il ne reste plus qu'a uploader depuis Vitis ...

... et ça ne fonctionne pas. Malheureusement, l'IDE ne me donne aucun message d'erreur, la console reste entièrement vide.
Après un certain temps à essayer de trouver une solution, j'ai malheureusement fini par capitulier, ne trouvant pas la source du problème.