INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

PROJET LOGIQUE PROGRAMMABLE

Timer Intel 8254

Fábio Eid Morooka

Heather McLean

Année universitaire 2018/2019

4ème année - EII

Table des métiers

Introduction

Ce projet consiste en la modélisation et la synthèse d'un timer Intel 8254 sur la plateforme DE10-Lite. Pour cela nous avons découpé le projet en deux parties. Dans la première, nous l'avons modélisé et simulé dans le logiciel Modelsim. Dans la deuxième partie, nous avons synthétisé le timer sur FPGA (MAX10) et l'a finalement intégré sur la carte DE10-Lite.

Pour simplifier le projet, nous avons passé en revue la solution fonctionnelle trouvée dans les Travaux Pratiques de MCSE. À partir de cette étude, nous avons divisé le projet du timer en trois entités. La première entité traite les communications de la CPU, c'est-à-dire qu'elle gère un mot de contrôle et envoie (ou reçoit) une valeur en 3 différents modes (on parle alors de Gestion dialogue). Une autre entité gère le bus buffer (Gestion buffer) et l'autre décompte la valeur du timer en mode d'opération 0 (Décopmt_mode0). Cette division du projet est illustrée à la figure 1 ci-dessous.

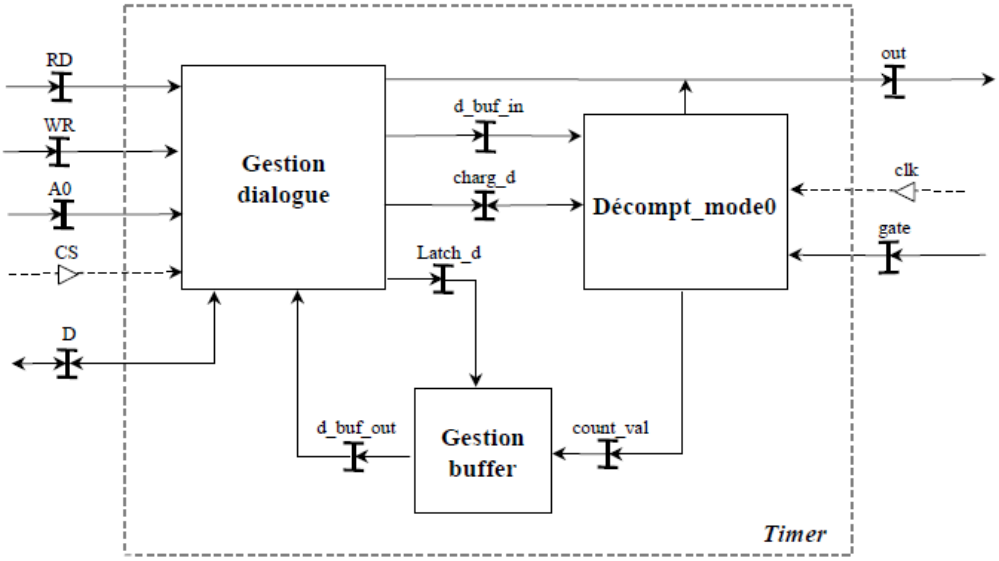


Figure 1 - Structure fonctionnelle du Timer 8254

Modélisation du timer dans l'environnement Modelsim

Modélisation de Dialogue CPU

Le CPU est un composant qui configure le timer en utilisant la variable D. Il s'agit d'une variable de 8 bits qui peut avoir deux types des données: un mot de contrôle ou une valeur. Dans le cas où la variable D a une valeur, nous avons deux possibilités: soit cette valeur sera envoyée au décompteur, soit elle sera envoyée à la CPU. S'il est envoyé à la CPU, il sera affiché sur l'écran à 7 segments de la carte DE10-Lite.

Le mot de contrôle est composé de quatre parties. Les deux premiers bits sont utilisés pour le *Select Counter* (SC). Les deux bits suivants sont les bits de *Read/Write* (RW). Il y a ensuite trois

bits pour le *Mode* (M) et un bit pour le *Binary Coded Decimal* (BCD). L'ordre des quatre parties dans le mot de contrôle est indiqué à la figure 2 ci-dessous.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

Figure 2 - Trame de la variable D

Dans notre projet, nous n'avons qu'un seul décompteur. Nous n'avons donc pas utilisé les bits SC. De plus nous n'avons pas utilisé les bits du *Mode*, parce que notre *Mode* est toujours 0. Enfin le bit BCD n'est pas également utilisé. Donc, dans notre projet, nous utilisons uniquement les bits RW. Les bits RW nous montrent la configuration qu'aura le timer lorsque la CPU lira la variable D. Cette partie de la trame D peut avoir quatre états: Latch, Least Significant Bit (LSB), Most Significant Bit (MSB) ou LSB et MSB ensemble (MSB+LSB). Ces options sont montrées dans la figure 3 au-dessous.

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

Figure 3 - Table de la variable RW

L'entité qui implémente la gestion CPU doit configurer le timer dans un ces modes selon les entrées reçues. C'est-à-dire que cette entité enverra au CPU la valeur qui est dans les bits moins forts ou les bits plus forts de la variable *d_buf_out*. Il envoie également au décompteur, la valeur dans les bits moins forts ou les bits plus forts de la variable *d_buf_in*.

Enfin cette entité peut aussi être dans l'état *Latch*, ce qui fait le décomptage. Dans ce cas, le buffer n'est pas mis à jour et la variable *d_buf_out* reste toujours avec la même valeur. La spécification fonctionnelle de cette entité est montrée dans la figure 4.

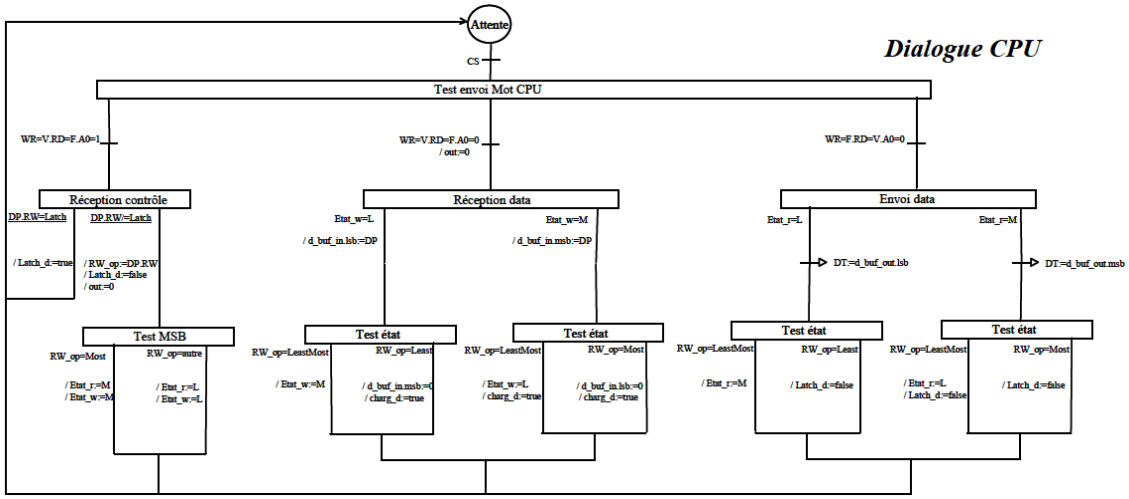


Figure 4 - Spécification fonctionnelle de la gestion CPU

Après avoir étudié le fonctionnement de cette entité, nous avons commencé à le modéliser. Cette partie a été réalisée au début sous une machine d'état, car nous avons 9 états différents (*attente, test_envoi, reception_controle, reception_data, evoi_data, testMSB, test_etat_recLSB, test_etat_recMSB, test_etat_envoi*). Nous avons également écrit une fonction qui nous permettait de traduire les noms d'états dans la représentation deux bits. Cela nous a permis de faire référence aux états par nom plutôt que par bits. Cette fonction est montrée dans l'annexe 1.

Nous avons ensuite modélisé l'entité gestionDialogue qui gère la CPU. Finalement, nous avons fait une simulation pour voir les résultats. Cette simulation est montrée dans la figure 5 au-dessous.

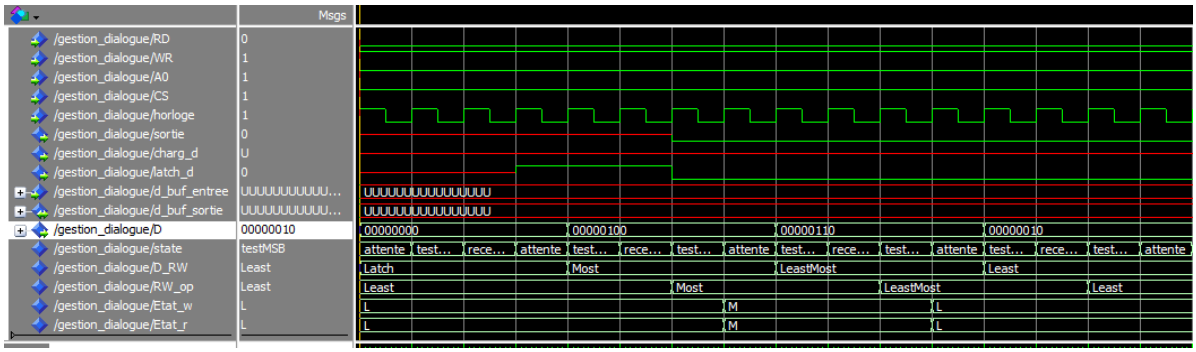


Figure 5 - Simulation initiale de l'entité gestionDialogue

À partir de la simulation, nous avons constaté que l'état a changé à chaque coup d'horloge, mais le comportement de cette entité n'était pas exactement parfait. Le comportement attendu était de changer la configuration du timer. Nous avons donc changé cette entité pour qu'elle puisse exécuter une fonction de la configuration du timer à chaque coup d'horloge. Enfin nous avons effectué une simulation pour assurer le bon fonctionnement de cette entité. Cette simulation est montrée dans la figure 6 ci-dessous.

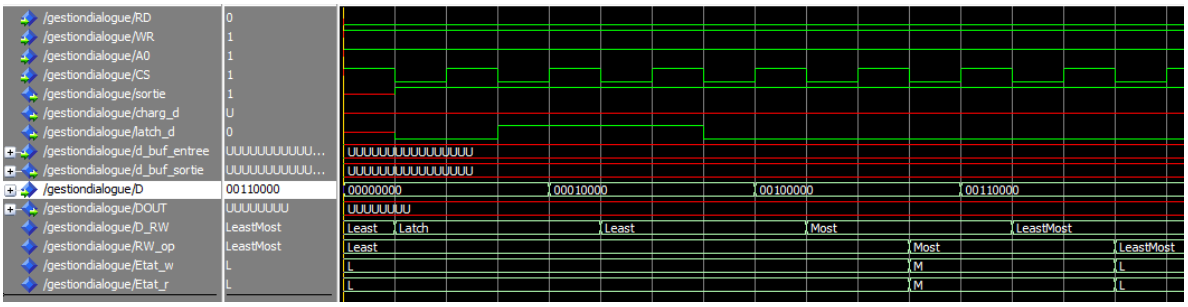


Figure 6 - Simulation finale de l'entité gestionDialogue

Les résultats de la simulation de la partie de réception de contrôle sont affichés, car c'est là que nous avons identifié l'erreur. Lorsque nous avons corrigé l'erreur, nous avons simulé les autres fonctions. Nous montrons ci-dessous (aux figures 7 et 8 respectivement) la simulation de l'écriture et de la lecture. Nous l'avons fait pour les modes LSB, MSB et enfin LSB + MSB.

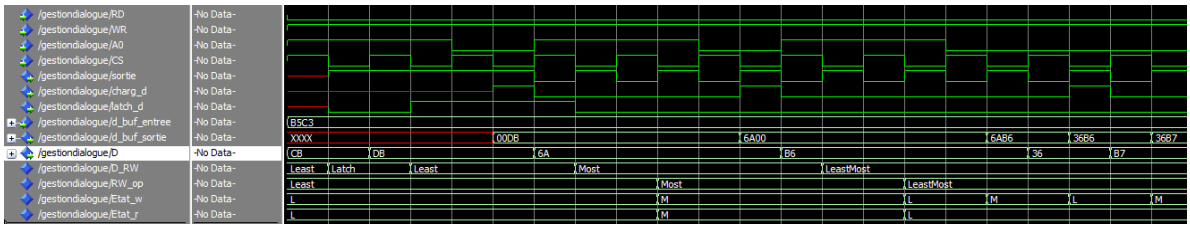


Figure 7 - - Simulation de l'entité gestionDialogue - Écriture

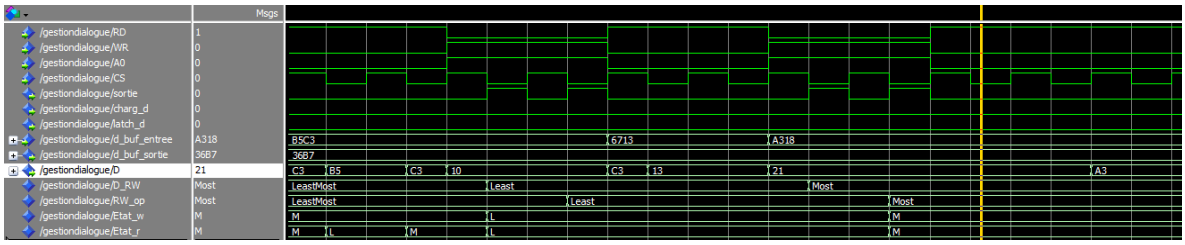


Figure 8 - - Simulation de l'entité gestionDialogue - Lecture

La description VHDL de cette entité est présentée à l'annexe 2. Dans la simulation, la variable D est décrite comme un bus à trois états bidirectionnel, implémenté dans cette entité. Cette entité est celle avec laquelle nous avons le plus de difficultés, car c'est l'entité principale qui assure le respect du cahier des charges. Au début, nous avons eu du mal à comprendre ses différentes caractéristiques, ainsi que ses entrées et ses sorties.

Modélisation de gestion buffer

L'entité qui gère le buffer (variable *d_buf_out*) garantit que, sauf si le timer est à l'état *Latch*, la valeur du buffer sera mise à jour avec la valeur du décompteur. Si le timer est à l'état *Latch* nous continuons à décompter, mais on ne mettons pas à jour le buffer. Sa solution fonctionnelle est plus simple (montré dans la figure 9 ci-dessous). Elle vérifie la valeur de la variable *Latch_d* et, si cette variable est false, la valeur du bus *d_buf_out* devient la valeur du décompteur. Ce cas est produit dans le cas où nous voulons écrire la valeur du décompteur

dans le périphérique externe où quand nous voulons changer le mode du timer (MSB, LSB, MSB+LSB).

Cette fonction est activée en modifiant les valeurs des entrées *Latch_d* et *count_val*. Nous avons donc écrit un processus avec ces deux entrées. Son code VHDl est trouvé dans l'annexe 3. Pour cette entité nous n'avons pas eu beaucoup des difficultés, mais nous devons faire attention quand nous faisons l'intégration sur la carte, pour assurer qu'elle est bien implémentée par un latch.

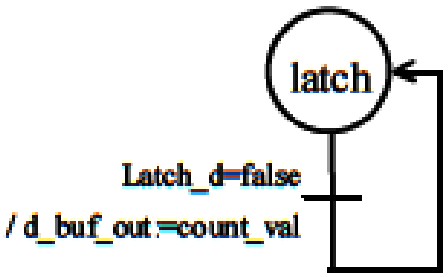


Figure 9 - Spécification fonctionnelle de la gestion du buffer

Modélisation de décompt_mode0, premier modèle

Le décompteur est un composant qui décompte la valeur du timer (sur 16 bits) en mode d'opération 0. Il se réinitialise lorsqu'il atteint zéro, donc sa valeur n'est jamais inférieur à zéro. Dans notre programme, lorsque le décompteur atteint zéro, il revient à la valeur maximale possible (tous les bits égaux à "1").

Cette entité utilise l'entrée GATE pour décompter. L'entrée GATE doit être égale à "1" pour que le décompteur puisse commencer. Et tant que l'entrée est égale à zéro, le décomptage arrête. Elle peut aussi charger une valeur transmise par le CPU, mais lorsque la variable *charg_d* est égale à "1", elle doit indiquer après que le chargement a été effectué correctement (en remettant la valeur de *charg_d* à "0").

De plus elle doit aussi indiquer quand sa valeur atteinte zéro, en utilisant la sortie variable *out*. Cette sortie égale a "0" si la valeur du décompteur est supérieur à zéro et "1" quand sa valeur atteint zéro. Après la remise à zéro, le compteur revient à sa valeur maximale, ce qui se produit lorsque les 16 bits ont la valeur "1". Cela nous donne 216 - 1 = 65535 (ou FFFF). Enfin, la CPU doit pouvoir lire la valeur actuelle du décompteur. Pour ce faire, il doit transmettre sa valeur à l'entité qui gère le buffer.

Nous avons modélisé cette entité en deux parties. Au début, nous l'avions modélisée comme une seule entité, mais nous l'avons modifiée pour ajouter des fonctionnalités supplémentaires. Dans cette première modélisation, nous avons utilisé deux variables du type buffer *resetCharg_d* et *setZero* pour réinitialiser les variables partagées *charg_d* et *out* respectivement. Le code de cette partie se trouve à l'annexe 4.

Nous avons simulé le décompteur pour vérifier que le chargement d'une valeur qui se trouve dans le bus *d_buf_in* a été bien fait. Il a également été noté que ce décomptage fonctionne bien. De plus, après le chargement de la valeur, la variable *resetCharg_d* est définie sur 1. Cela permet de réinitialiser la variable *charg_d*. De plus, lorsque le décompteur atteint 0, nous voyons que le signal *setZero* devient 1 et qu'il s'agit donc de la réinitialisation de la variable out. Cette simulation est illustrée à la figure 10 ci-dessous.

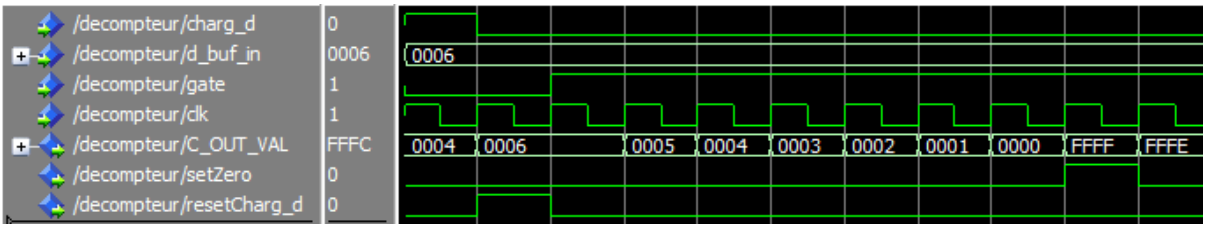


Figure 10 - Simulation du décompteur niveau 1

Modélisation de décompt_mode0, niveau 2

Ensuite nous avons amélioré le décompteur qui a été fait auparavant. Pour cela nous avons segmenté le décompteur en trois entités. L'entité resetDemandeChargement qui gère la variable qui remis à zéro la variable *charg_d*, l'entité detectionZero qui vérifie si la valeur du décompteur arrive à zéro et qui mettre à 1 la valeur de la sortie *out* et l'entité décompteur qui fait proprement dit le décomptage d'une valeur ainsi que fait le chargement d'une valeur transmis par la CPU dans le décompteur. La structure fonctionnelle du décompteur_mode 0 avec ces trois entités est dans la figure 11 ci-dessous.

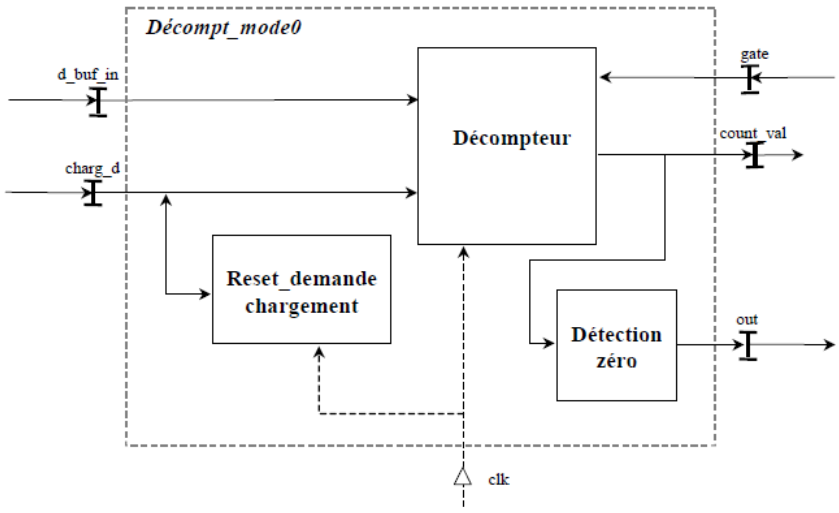


Figure 11 - Structure fonctionnelle du décompteur_mode 0

Le premier composant que nous avons fait était le décompteur, car nous avions déjà le code de base du décomptage grâce au premier modèle que nous avions. Le but de cette entité est de décompter une variable si l'entrée GATE est égale à "1". Il charge également la valeur transmise par la CPU lorsque la variable *charg_d* est égale à "1". Son code est dans l'annexe 5. Pour nous assurer que cette entité de décomptage a été bien réalisée, nous avons simulé

cette entité pour en évaluer le comportement. Cette simulation est illustrée à la figure 12 ci-dessous.

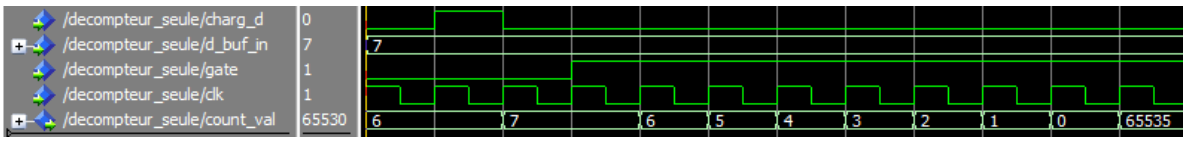


Figure 12 - Simulation du décompteur

Ensuite nous avons fait l'entité resetDemandeChargement. Cette entité est simple, car elle doit faire la gestion de la variable *charg_d* du décompteur_mode 0. Elle envoie une sortie (variable *resetCharg_d*) avec la valeur égale à "1" si la variable *charg_d* est égale à "1", sinon elle envoie "0". Cette sortie sera utilisée pour remettre la variable *charg_d* à "0". Son code figure à l'annexe 6.

Enfin nous avons modélisé l'entité detectionZero vérifie si la valeur du décompteur est égal à zéro. Si tel est le cas, la variable de sortie est définie sur "1". Sinon la variable de sortie est définie sur "0". Son code figure à l'annexe 7.

Pour finir ce modèle, nous avons créé une seule entité comprenant les trois entités (Annexe 8). Nous avons ensuite exécuté une simulation pour vérifier le comportement du décompteur_mode 0. Cette simulation est illustrée à la figure 13 ci-dessous.

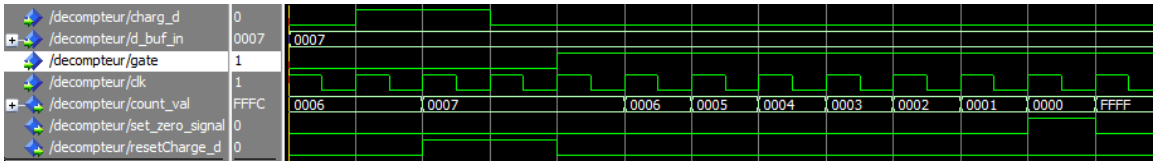
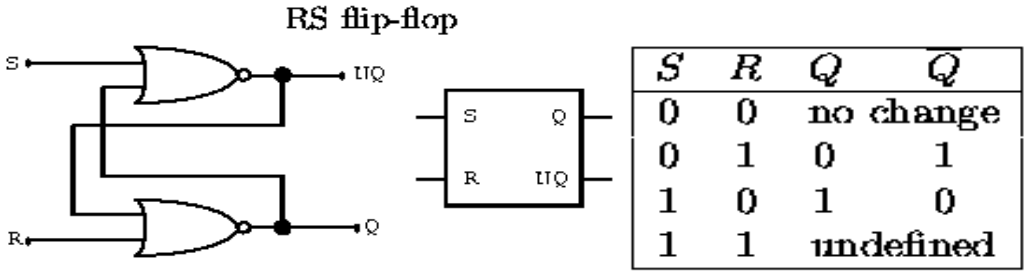


Figure 13 - Deuxième simulation du décompteur

Modélisation du timer complet

Enfin, avec tous les composants modélisés et déjà testés séparément, nous avons procédé à la modélisation du timer Intel 8254 dans le logiciel. Avant de procéder à l'édition du timer, nous avons réfléchi aux variables partagées de la spécification. Nous avons recherché des variables qui pourraient poser un problème, car elles ont un accès en écriture par deux entités différentes et ont identifié *charg_d* et *out*. Nous avons vu que ces entités en particulier utilisaient simplement ces variables partagées pour écrire 1 ou 0. En raison de la simplicité de ces variables, nous les avons modélisées sous la forme d'une bascule RS (Reset-Set). La description VHDL figure à l'Annexe 9.



S	R	Q	$\overline{Q}$
0	0	no change	
0	1	0	1
1	0	1	0
1	1	undefined	

Figure 14 - Description d'une bascule RS (portes logiques et tableau-vérité)

Les autres variables étaient unidirectionnelles, ce qui signifie qu'une entité l'a écrite et une autre l'a lue. Un simple signal logique peut être utilisé pour ceux-ci. Enfin, nous avons créé l'entité timer (donnée dans l'annexe 10). Il s'agit de l'entité de niveau supérieur dans laquelle nous avons lié les composants précédents à l'aide de signaux intermédiaires.

Tests du timer dans l'environnement Modelsim

Pour effectuer des tests dans l'environnement Modelsim, nous avons modifié le bus directionnel D pour un bus d'entrée DIN et un bus de sortie DOUT. Nous avons ajouté deux sorties : *charg_d* (1 bit) et *d_buf_out* (16 bits) afin de mieux visualiser la simulation qui sera implémentée dans la carte.

Dans la première simulation (figure 15 ci-dessous), nous voyons les fonctions d'écriture et de lecture de valeurs dans MSB + LSB.

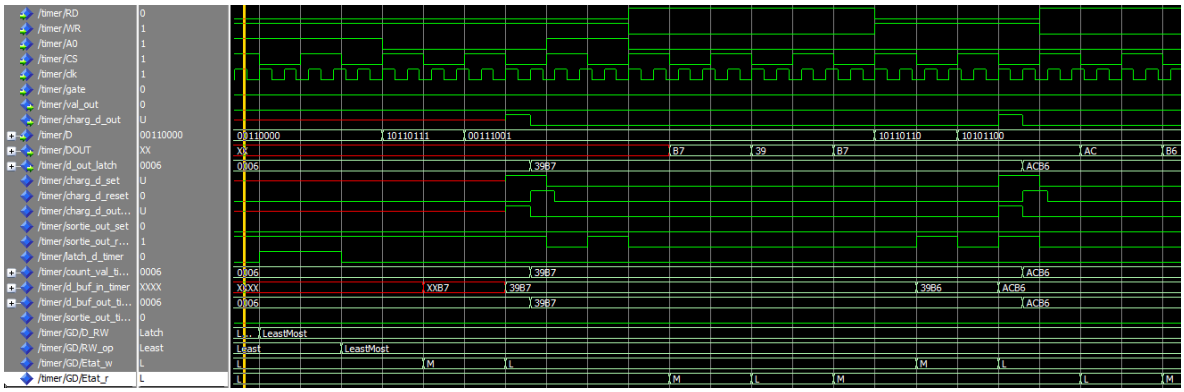


Figure 15 - Simulation d'écriture et lecture du timer MSB+LSB

Nous avons ensuite effectué une simulation pour vérifier le comportement du décomptage, du buffer et de la commande latch. La figure 16 ci-dessous illustre une simulation du décomptage d'une valeur chargée. Dans ces résultats, nous voyons la valeur du décompteur décrémente ainsi que la fonctionnalité correcte de la commande latch. Comme nous nous y attendions, la sortie du décompteur *d_out_latch* cesse de recevoir la valeur du décompteur, même si cette valeur continue à diminuer. Nous voyons également le signal interne *sortie_out_set* défini sur "1" lorsque la valeur du décompteur atteint zéro. Ce signal est le signal défini de la bascule RS de la variable partagée *out*. Par conséquent, nous voyons

également la bascule correctement simulée, car la valeur de la sortie d'horloge *val_out* est définie sur "1" en même temps que le signal output_out_set est défini sur "1".

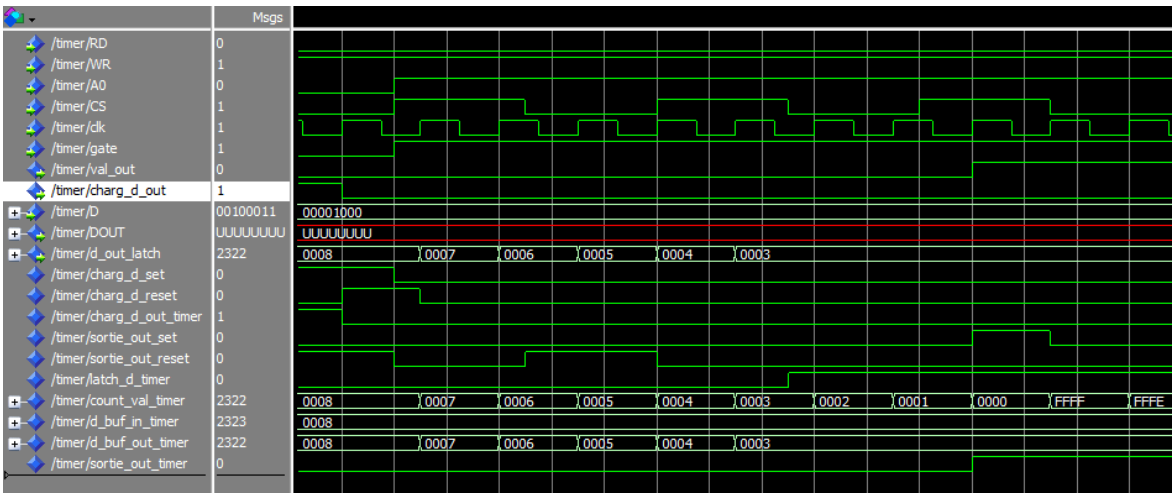


Figure 16 - Simulation de décomptage du timer

Ensuite, nous avons continué la simulation pour voir si la commande latch a été bien modélisée. Nous voyons dans la figure 17 ci-dessous que la sortie du timer *d_out_latch* a la valeur 0x0003 quand la sortie du décompteur *count_val_timer* change des valeurs. Au moment que nous sortons de l'état *latch*, nous voyons bien que la sortie du timer assume la valeur du buffer. Nous voyons que cette entité a été bien réalisée. En plus, nous voyons que la sortie qui montre quand le timer atteint zéro (variable *val_out*) est mis à "0" à ce moment, car nous avons changé la configuration du timer.

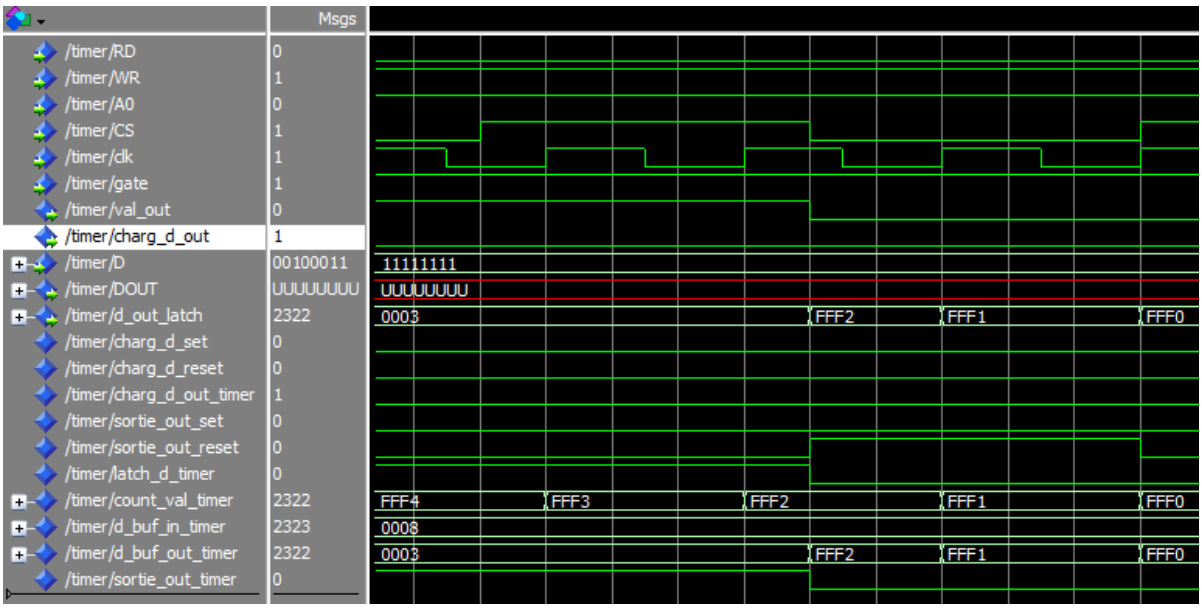


Figure 17 - - Simulation de la commande latch du timer

Synthèse du timer dans l'environnement Quartus Lite

Après la validation de la simulation du projet dans l'environnement Modelsim, nous avons étudié la synthèse du projet. Nous avons vu les vues RTL (*Register Transfer Level*) de différentes entités pour mieux comprendre comment elles ont été synthétisées et éventuellement apporter des corrections si nous constatons une anomalie.

Synthèse de l'entité "gestionDialogue.vhd"

Nous avons commencé par étuder la synthèse de l'entité qui fait la Dialogue CPU. Nous avons aperçu qui cette entité est très complexe avec des nombreux éléments logiques et registres. Nous pouvons voir cette complexité dans la compilation de l'entité, montrée dans la figure 18 ci-dessous.

Top-level Entity Name	gestionDialogue
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	50 / 49,760 (< 1 %)
Total registers	33
Total pins	55 / 360 (15 %)

Figure 18 - Compilation de l'entité qui fait le Dialogue CPU

Nous avons vu aussi la vue RTL de cette entité. Nous avons vu les registres nécessaires et présents dans notre description VHDL (*d_buf_sortie*, *DOUT*, etc.) ainsi que les éléments de liaison qui réalisent le comportement décrit par les graphes et le code VHDL.

Synthèse de l'entité "gestionBuffer"

Dans la synthèse de l'entité qui gère le buffer, il est intéressant de remarquer qu'elle a été bien faite, parce que dans le vu RTL (figure 19 ci-dessous) nous voyons des latches et nous remarquons également que, dans les résultats de la compilation (figure 20 ci-dessous), il n'y a pas des bascules.

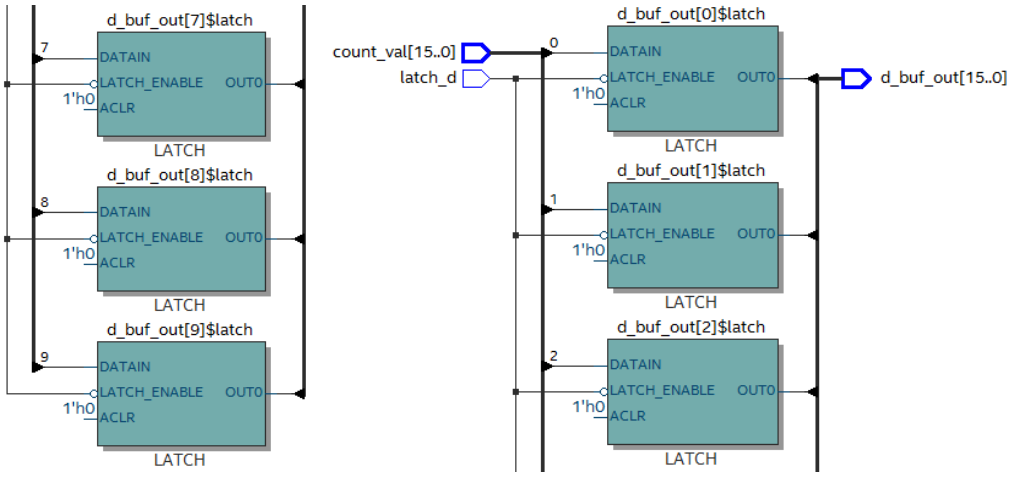


Figure 19 - Vu RTL de l'entité "gestionBuffer"

Top-level Entity Name	gestionBuffer
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	17 / 49,760 (< 1 %)
Total registers	0
Total pins	33 / 360 (9 %)

Figure 20 - Résultats de la compilation de l'entité "gestionBuffer"

Synthèse du décompteur

Enfin, la dernière entité sur laquelle nous avons étudié la synthèse est le décompteur. Après synthèse, nous avons obtenu le RTL présenté à la figure 21 ci-dessous. Nous pouvons également voir les trois autres entités que nous avons modélisées.

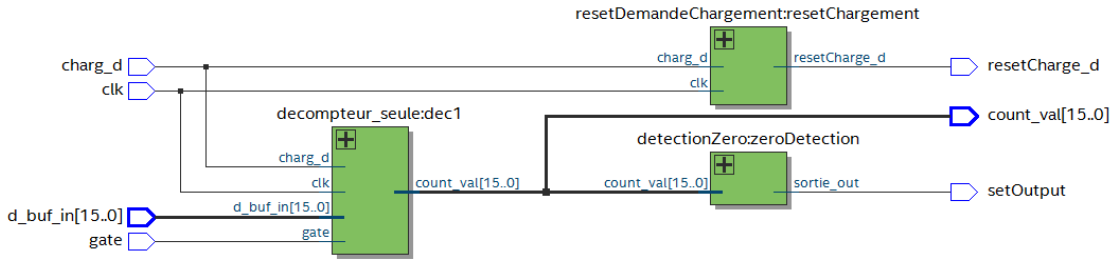


Figure 21 - Vu RTL initial du décompteur

Pour faire une étude plus précise du décompteur, nous avons examiné les matériels de chaque partie. Nous pouvons voir à la figure 22 que le décompteur utilise une registre de 16 bits pour sauvegarder sa valeur ainsi qu'une bascule pour gérer la réinitialisation de la variable charg_d. Cela donne un total de 17 bascules, comme montrent les résultats de la compilation (figure 23). Nous voyons également l'implémentation d'un additionneur de 16 bits pour le décomptage, ainsi qu'un comparateur pour la détection du 0.

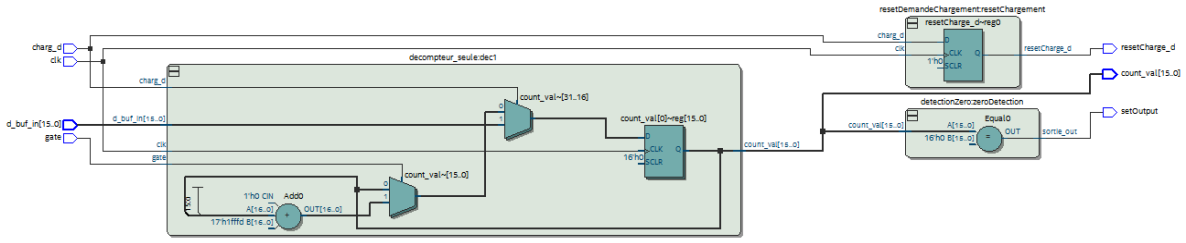


Figure 22 - Vu RTL du décompteur

Top-level Entity Name	decompteur
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	24 / 49,760 (< 1 %)
Total registers	17
Total pins	37 / 360 (10 %)

Figure 23 - Résultats de la compilation

Synthèse du timer

Pour examiner les entrées et sorties du timer, nous avons créé un bloc timer à partir de la description VHDL que nous avons écrit. Ce bloc est montré dans la figure 24 ci-dessous.

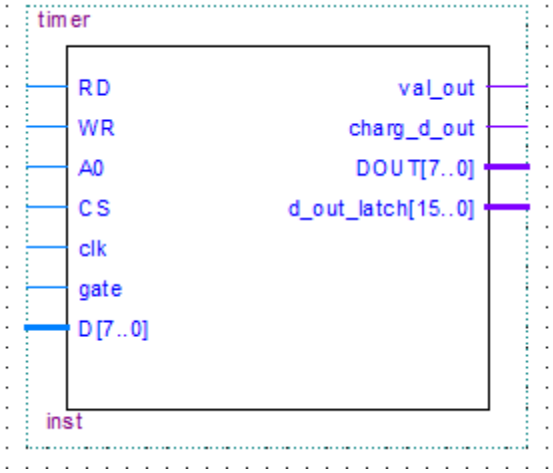


Figure 24 - Bloc du timer

Ensuite, nous avons étudié la synthèse du timer dans la vue RTL montré à la figure 25 ci-dessous. Nous pouvons voir le timer composé du dialogue CPU, du décompteur, de la gestion du buffer et les deux variables partagées en accès en écriture double, implémenté par des bascules RS asynchrones.

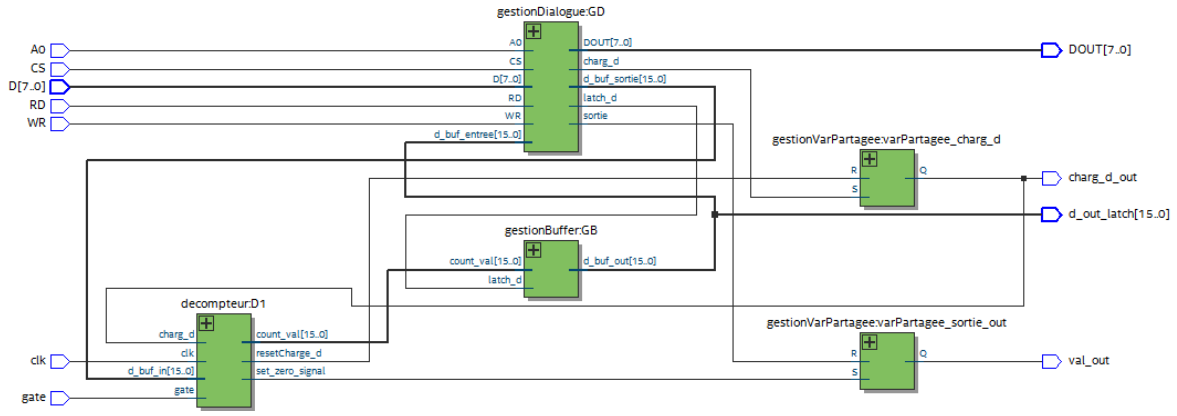


Figure 25 - Vu RTL du timer

Les résultats de la compilation (montré dans la figure 26 au-dessous) nous montre qui 50 registres et 93 LEs ont été utilisés. Le nombre de LEs utilisées est très peu comparé au nombre de LE disponibles sur le carte, mais correspond à notre projet qui n'est qu'un petit projet d'un timer.

Top-level Entity Name	timer
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	93 / 49,760 (< 1 %)
Total registers	50
Total pins	40 / 360 (11 %)

Figure 26 - - Résultats de la compilation du timer

Intégration du timer sur la plateforme DE10-Lite

Pour effectuer l'intégration du timer sur la plateforme DE10-Lite, nous avons créé une entité de niveau supérieur pour faire les assignations des LEDs, des switches et des boutons avec les entrées et sorties du timer. Pour voir la sortie du timer dans les afficheurs 7 segments, nous avons utilisé le fichier de description des afficheurs déjà utilisé en TP. Nous avons utilisé un diviseur de fréquence pour réduire la fréquence de 50MHz de la carte afin qu'elle soit suffisamment lente pour que nous puissions voir les changements. Les assignations des PINS avec les noms donnés aux éléments de la carte sont effectuées dans le script TCL du projet (pin_assignment_DE10-Lite.tcl). Le fichier de cette entité se trouve à l'annexe 11.

Ensuite, nous avons regardé le vu RTL de cette entité (figure 27 ci-dessous). Nous voyons que le timer a été correctement implémenté en ce qui concerne les signaux d'entrée et de sortie.

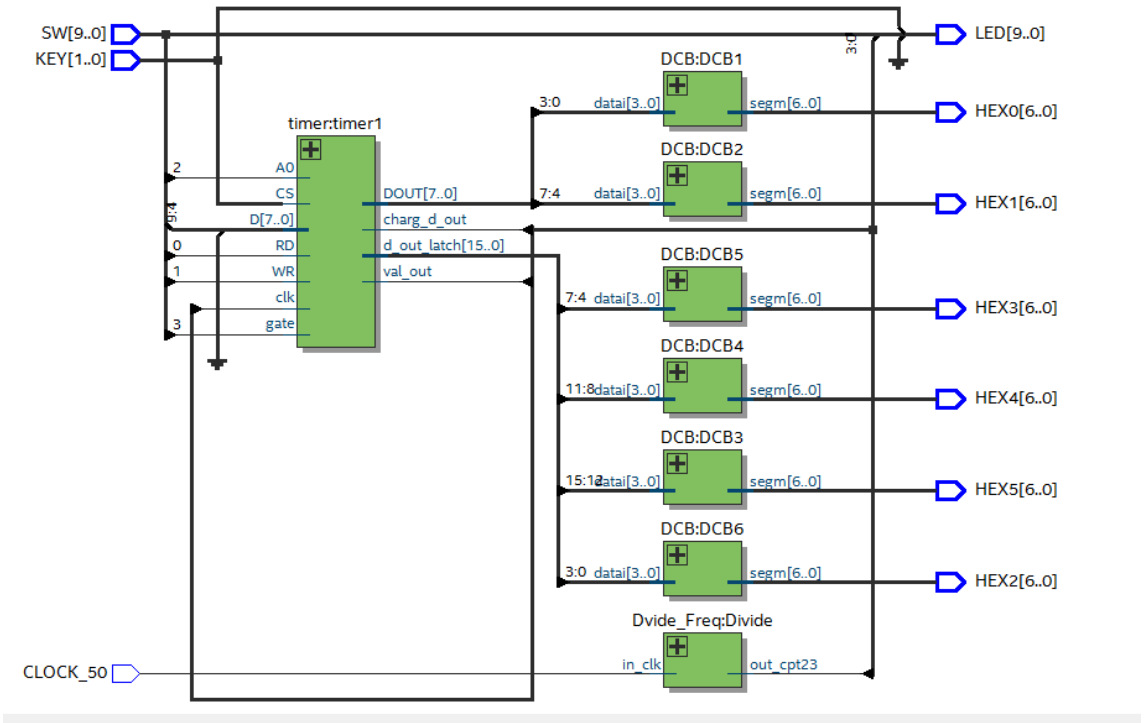


Figure 27 - Vu RTL de l'entité tête du projet

Pour conclure notre projet, nous avons chargé notre code sur la carte DE10-Lite et l'avons testé. Lors de nos tests, nous avons essayé toutes les options de configuration du timer, l'écriture d'une valeur, la lecture d'une valeur et le décomptage. Comme dans les simulations, les tests ont été concluants et le projet a bien fonctionné. La figure 28 ci-dessous montre l'assemblage de notre projet.

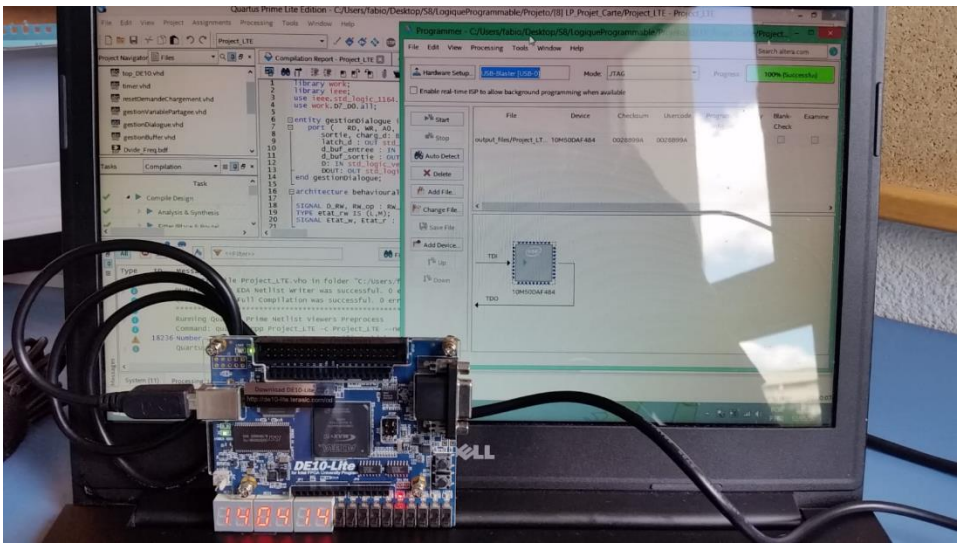


Figure 28 - Implémentation dans la carte DE10-Lite

Conclusion

Dans ce projet, nous avons étudié le développement d'un timer Intel 8254 qui était introduit en TD. Nous avons notamment utilisé le réseau de Petri que nous avons développé pour intégrer le système dans des circuits logiques programmables. Cela impliquait l'utilisation d'une machine à états pour simplifier la mise en œuvre de l'entité principale. Ce projet nous a également permis d'étudier l'implémentation de variables partagées en VHDL. Les variables partagées étaient nécessaires en raison des opérations de lecture et d'écriture de plusieurs entités. Cependant, ils n'étaient utilisés que dans les cas où une entité effectuait une lecture suivie d'une écriture. Les variables partagées ont été modélisées à l'aide de bascules RS et des signaux normaux ont été utilisés pour représenter d'autres variables. Après la mise en œuvre, nous avons étudié la synthèse de notre projet dans l'environnement Quartus et nous avons utilisé ces informations pour examiner les différents composants matériels générés par notre projet. Cela nous a permis de confirmer que les messages attendus avaient été utilisés. Par exemple, nous avons vérifié que des bascules étaient utilisées pour implémenter les variables partagées. Enfin, nous avons programmé la carte avec notre code VHDL. Cela impliquait de coupler les entrées et les sorties de notre code avec les entrées et les sorties physiques de la carte. Ensuite, nous avons vérifié que la fonctionnalité avait été correctement implémentée et que le timer fonctionnait comme prévu.

Annexes

Annexe 1 - Code du paquet “D7_D0_package.vhd”

```
1   library work;
2   library ieee;
3   use ieee.std_logic_1164.all;
4
5   package D7_D0 IS
6           TYPE RW_type IS (Latch, Least, Most, LeastMost);
7           function RW_convert (signal D : std_logic_vector(7 downto 0)) return RW_type;
8   end D7_D0;
9
10  package body D7_D0 is
11      function RW_convert (signal D : std_logic_vector(7 downto 0)) return RW_type is
12      variable rw: RW_type; -- local variable
13      variable RW_temp : std_logic_vector(1 downto 0);
14      begin
15          RW_temp(0) := D(4);
16          RW_temp(1) := D(5);
17          case RW_temp is
18              when "00"   => rw := Latch;
19              when "01"   => rw := Least;
20              when "10"   => rw := Most;
21              when OTHERS => rw := Leastmost;
22          end case;
23          return rw;  -- return min value
24      end;
25  end D7_D0;
```

Annexe 2 - Code d'entité "gestionDialogue.vhd"

```
1   library work;
2   library ieee;
3   use ieee.std_logic_1164.all;
4   use work.D7_D0.all;
5
6   entity gestionDialogue is
7           port (  RD, WR, A0, CS: IN std_logic;
8                   sortie, charg_d: BUFFER std_logic;
9                   latch_d : OUT std_logic;
10                  d_buf_entree : IN std_logic_vector(15 downto 0);
11                  d_buf_sortie : OUT std_logic_vector(15 downto 0);
12                  D: BUFFER std_logic_vector(7 downto 0));
13  end gestionDialogue;
14
15  architecture behavioural of gestionDialogue is
16
17  SIGNAL D_RW, RW_op : RW_type := Least;
18  TYPE etat_rw IS (L,M);
19  SIGNAL Etat_w, Etat_r : etat_rw := L;
20
21  BEGIN
22          PROCESS(CS)
23          BEGIN
24                  if( charg_d = '1' AND sortie = '1') then
25                          charg_d <= '0'; sortie <= '0';
26                  elsif( charg_d = '1' AND sortie = '0') then
27                          charg_d <= '0';
28                  elsif( charg_d = '0' AND sortie = '1') then
29                          sortie <= '0';
30                  elsif(FALLING_EDGE(CS)) then
31                          --Reception controle
32                          if(WR = '1' and RD = '0' and A0 = '1') then
33                                  --set value of RW_signal
34                                  D_RW <= RW_convert(D);
35                                  if(D_RW = Latch) then
36                                          latch_d <= '1';
```

```
                        else
                                RW_op <= D_RW;  --RW component of DP octet
                                latch_d <= '0';
                                sortie <= '1';
                                -- testMSB
                                if (D_RW = Most) then
                                        Etat_r <= M;
                                        Etat_w <= M;
                                else
                                        Etat_r <= L;
                                        Etat_w <= L;
                                end if;
                        end if;
                --Reception data
                elsif(WR = '1' and RD = '0' and A0 = '0') then
                        sortie <= '1';
                        if(etat_w = L) then
                                --d_out buffer lsb
                                d_buf_sortie(7 DOWNTO 0) <= D;
                                -- test_etat_recLSB
                                if(RW_op = LeastMost) then
                                        Etat_w <= M;
                                else
                                        d_buf_sortie(15 DOWNTO 8) <= "00000000";
                                        charg_d <= '1';
                                end if;
                        else
                                --d_out buffer mlsb
                                d_buf_sortie(15 DOWNTO 8) <= D;
                                -- test_etat_recMSB
                                if(RW_op = LeastMost) then
                                        Etat_w <= L;
                                        charg_d <= '1';
                                else
                                        d_buf_sortie(7 DOWNTO 0) <= "00000000";
                                        charg_d <= '1';
                                end if;
                        end if;

                        -- Envoi data
                        elsif(WR = '0' and RD = '1' and A0 = '0') then
                                if(etat_r = L) then
                                        --d_out buffer lsb
                                        D <= d_buf_entree(7 DOWNTO 0);
                                else
                                        --d_out buffer msb
                                        D <= d_buf_entree(15 DOWNTO 8);
                                end if;
                                -- test_etat_envoi
                                if(RW_op = LeastMost) then
                                        if(Etat_r <= L) then
                                                Etat_r <= M;
                                        else
                                                Etat_r <= L;
                                                latch_d <= '0';
                                        end if;
                                else
                                        latch_d <= '0';
                                end if;
                        end if;
                end if;
        end process;
end behavioural;
```

Annexe 3 - Code d'entité "gestionBuffer.vhd"

```
1   library ieee;
2   use ieee.std_logic_1164.ALL;
3
4   ENTITY gestionBuffer IS
5   PORT (  latch_d : IN std_logic;
6           count_val : IN std_logic_vector (15 DOWNTO 0);
7           d_buf_out : OUT std_logic_vector (15 DOWNTO 0));
8   END ENTITY;
9
10  ARCHITECTURE behavior OF gestionBuffer IS
11  BEGIN
12          PROCESS (latch_d, count_val) IS
13          BEGIN
14                  IF (latch_d = '0') THEN
15                          d_buf_out <= count_val;
16                  END IF;
17          END PROCESS;
18  END behavior;
```

Annexe 4 - Code du décompteur_mode0_niveau1

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   ENTITY decompteur IS
6           PORT(   charg_d : IN std_logic;                                                 -- s'il faut charger une valeur
7                   d_buf_in : IN std_logic_vector(15 DOWNTO 0);                            -- la valeur a charger dans le composant
8                   gate : IN std_logic;                                                    -- on decompte ou pas
9                   clk : IN std_logic;                                                     -- l'horloge
10                  C_OUT_VAL : BUFFER std_logic_vector(15 DOWNTO 0) := "0000000000000100"; -- la sortie du decompteur
11                  setZero : BUFFER std_logic := '0';                                      -- buffer qui permet mis a zero la sortie
12                  resetCharg_d : BUFFER std_logic := '0');                                -- buffer qui permet mis a zero le charg_d
13  END decompteur;
```

```
15  ARCHITECTURE behaviour OF decompteur IS
16          BEGIN
17
18          PROCESS(clk)
19          BEGIN
20                  IF rising_edge(clk) THEN
21                          IF charg_d = '1' THEN           --demande de chargement
22                                  C_OUT_VAL <= d_buf_in;
23                                  resetCharg_d <= '1';    -- reset la variable charge_d
24                          ELSE
25                                  IF gate = '1' THEN -- cas decomptage
26                                          C_OUT_VAL <= C_OUT_VAL - 1;
27                                          IF (C_OUT_VAL = "0000000000000000") THEN
28                                                  setZero <= '1';
29                                          END IF;
30                                  END IF;
31                          END IF;
32                          -- Remis a zero
33                          IF resetCharg_d = '1' THEN
34                                  resetCharg_d <= '0';
35                          END IF;
36                          -- Remis a zero
37                          IF setZero = '1' THEN
38                                  setZero <= '0';
39                          END IF;
40                  END IF;
41          END PROCESS;
42  END behaviour;
```

Annexe 5 - Code de l'entité "décompteur_seule.vhd"

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  ENTITY decompteur_seule IS
6  PORT(   charg_d : IN std_logic;                                          -- Demande de chargement
7          d_buf_in : IN std_logic_vector(15 DOWNTO 0);                     -- Valeur de chargement
8          gate : IN std_logic;                                             -- Decompte ou faire rien
9          clk : IN std_logic;                                              -- Signal clock
10         count_val : BUFFER std_logic_vector(15 DOWNTO 0) := "0000000000000110"); -- Valeur de la sortie
11 END decompteur_seule;
12
13 ARCHITECTURE behaviour OF decompteur_seule IS
14         BEGIN
15         PROCESS(clk)
16         BEGIN
17                 IF rising_edge(clk) THEN
18                         -- Cas demande de chargement
19                         IF charg_d = '1' THEN
20                                 count_val <= d_buf_in;
21                         ELSE
22                                 IF gate = '1' THEN -- Cas decompter
23                                         count_val <= count_val - 1;
24                                 END IF;
25                         END IF;
26                 END IF;
27         END PROCESS;
28  END behaviour;
```

Annexe 6 - Code de l'entité "resetDemandeChargement.vhd"

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  ENTITY resetDemandeChargement IS
5  PORT(   clk : IN std_logic;
6          charg_d : IN std_logic;
7          resetCharge_d: OUT std_logic := '0');
8  END resetDemandeChargement;
9
10 ARCHITECTURE behaviour OF resetDemandeChargement IS
11 BEGIN
12         PROCESS(clk)
13         BEGIN
14                 IF (rising_edge(clk)) THEN
15                         IF(charg_d = '1') THEN
16                                 resetCharge_d <= '1';
17                         ELSE
18                                 resetCharge_d <= '0';
19                         END IF;
20                 END IF;
21         END PROCESS;
22  END behaviour;
```

Annexe 7 - Code de l'entité "detectionZero.vhd"

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   ENTITY detectionZero IS
5   PORT(   sortie_out : OUT std_logic := '0';              -- Signal de la sortie defaut a 0
6           count_val : IN std_logic_vector(15 DOWNTO 0));  -- Valeur du decompteur
7   END detectionZero;
8
9   ARCHITECTURE behaviour OF detectionZero IS
10  BEGIN
11          PROCESS(count_val)
12          BEGIN
13                  IF(count_val = "0000000000000000") THEN
14                          sortie_out <= '1';
15                  ELSE
16                          sortie_out <= '0';
17                  END IF;
18          END PROCESS;
19   END behaviour;
```

Annexe 8 - Code du décompteur_mode0_niveau2

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   ENTITY decompteur IS
5   PORT(   charg_d : IN std_logic;                                                 -- Demande de chargement
6           d_buf_in : IN std_logic_vector(15 DOWNTO 0);                            -- Valeur de chargement
7           gate : IN std_logic;                                                    -- Decompte ou faire rien
8           clk : IN std_logic;                                                     -- Signal clock
9           count_val : BUFFER std_logic_vector(15 DOWNTO 0) := "0000000000000110"; -- Valeur de la sortie
10          set_zero_signal : OUT std_logic:='0';                                   -- Zero au debout
11          resetCharge_d : OUT std_logic := '0');                                  -- Zero au debout
12  END decompteur;
13
14  ARCHITECTURE behaviour OF decompteur IS
15  --component declarations
16  component detectionZero is
17  PORT(   sortie_out : OUT std_logic := '0';              -- Signal de la sortie defaut a 0
18          count_val : IN std_logic_vector(15 DOWNTO 0));  -- Valeur de la sortie
19  END component;
20
21  component resetDemandeChargement is
22  PORT(   clk : IN std_logic;
23          charg_d : IN std_logic;
24          resetCharge_d: OUT std_logic := '0');
25  END component;
26
27  component decompteur_seule is
28  PORT(   charg_d : IN std_logic;                                                 -- Demande de chargement
29          d_buf_in : IN std_logic_vector(15 DOWNTO 0);                            -- Valeur de chargement
30          gate : IN std_logic;                                                    -- Decompte ou faire rien
31          clk : IN std_logic;                                                     -- Signal clock
32          count_val : BUFFER std_logic_vector(15 DOWNTO 0) := "0000000000000110"); -- Valeur de la sortie
33  END component;
34          BEGIN
35                  zeroDetection : detectionZero PORT MAP(set_zero_signal, count_val);
36                  resetChargement : resetDemandeChargement PORT MAP(clk, charg_d, resetCharge_d);
37                  dec1: decompteur_seule PORT MAP(charg_d, d_buf_in, gate, clk, count_val);
38   END behaviour;
```

Annexe 9 - Code de la bascule RS (Set-Reset)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   ENTITY gestionVarPartagee IS
5           PORT(   S: IN std_logic;
6                   R: IN std_logic;
7                   Q: BUFFER std_logic);
8   END gestionVarPartagee;
9
10  ARCHITECTURE behaviour OF gestionVarPartagee IS
11
12          SIGNAL notQ : STD_LOGIC;
13
14          BEGIN
15                  Q <= R nor notQ;
16                  notQ <= S nor Q;
17   END behaviour;
```

Annexe 10 - Code du timer

```
1   library work;
2   library ieee;
3   use ieee.std_logic_1164.all;
4
5   ENTITY timer IS
6           PORT (  RD, WR, A0, CS, clk, gate : IN std_logic;
7                   val_out, charg_d_out : OUT std_logic;
8                   D : IN std_logic_vector(7 downto 0);
9                   DOUT : OUT std_logic_vector(7 downto 0);
10                  d_out_latch : OUT std_logic_vector(15 downto 0));
11  end timer;
12
13  architecture behavioural of timer is
14
15  --component declarations
16  component gestionBuffer is
17          PORT (  latch_d : IN std_logic;
18                  count_val : IN std_logic_vector (15 DOWNTO 0);
19                  d_buf_out : OUT std_logic_vector (15 DOWNTO 0));
20  end component;
21
22  component decompteur is
23          PORT(   charg_d : IN std_logic;                                          -- Demande de chargement
24                  d_buf_in : IN std_logic_vector(15 DOWNTO 0);                     -- Valeur de chargement
25                  gate : IN std_logic;                                             -- Decompte ou faire rien
26                  clk : IN std_logic;                                              -- Signal clock
27                  count_val : BUFFER std_logic_vector(15 DOWNTO 0) := "0000000000000110"; -- Valeur de la sortie
28                  set_zero_signal : OUT std_logic:='0';                            -- Zero au debout
29                  resetCharge_d : OUT std_logic:='0');                             -- Zero au debout
30  end component;
31
```

```
component gestionDialogue is
        PORT(   RD, WR, A0, CS: IN std_logic;
                sortie, charg_d: BUFFER std_logic;
                latch_d : OUT std_logic;
                d_buf_entree : IN std_logic_vector(15 downto 0);
                d_buf_sortie : OUT std_logic_vector(15 downto 0);
                D: IN std_logic_vector(7 downto 0);
                DOUT: OUT std_logic_vector(7 downto 0));
end component;

component gestionVarPartagee is
        PORT(   S: IN std_logic;
                R: IN std_logic;
                Q: BUFFER std_logic);
end component;

--Signal declarations
SIGNAL charg_d_set, charg_d_reset, charg_d_out_timer, sortie_out_set, sortie_out_reset, latch_d_timer: std_logic;
SIGNAL count_val_timer, d_buf_in_timer, d_buf_out_timer: std_logic_vector (15 DOWNTO 0);
SIGNAL sortie_out_timer: std_logic := '0';

begin
        d_out_latch <= d_buf_out_timer;
        charg_d_out <= charg_d_out_timer;
        val_out <= sortie_out_timer;

--Port maps
GD : gestionDialogue PORT MAP(RD, WR, A0, CS, sortie_out_reset, charg_d_set, latch_d_timer, d_buf_out_timer, d_buf_in_timer, D, DOUT);

varPartagee_charg_d: gestionVarPartagee PORT MAP(charg_d_set, charg_d_reset, charg_d_out_timer);
varPartagee_sortie_out: gestionVarPartagee PORT MAP(sortie_out_set, sortie_out_reset, sortie_out_timer);

SB : gestionBuffer PORT MAP(latch_d_timer, count_val_timer, d_buf_out_timer);
D1 : decompteur PORT MAP(charg_d_out_timer, d_buf_in_timer, gate, clk, count_val_timer, sortie_out_set, charg_d_reset);

end behavioural;
```

Annexe 11 - Code d'entité tête de la carte

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY top_DE10 IS
        PORT ( CLOCK_50 : IN std_logic;
                        SW : IN std_logic_vector(9 downto 0);
                        KEY : IN std_logic_vector(1 downto 0);
                       LED : OUT  std_logic_vector(9 downto 0);
                       HEX0, HEX1, HEX2, HEX3, HEX4, HEX5 : OUT std_logic_vector(6 downto 0));
--      PORT ( HEX00,HEX01,HEX02,HEX03,HEX04,HEX05,HEX06 : OUT std_logic);
END top_DE10;

ARCHITECTURE arch_top OF top_DE10 IS
--Components declaration
COMPONENT Dvide_Freq
        PORT (in_clk : IN std_logic; out_cpt23 : OUT std_logic);
END COMPONENT;

COMPONENT DCB is
        port (datai : IN std_logic_vector (3 downto 0);
                segm : OUT std_logic_vector (6 downto 0) );
end COMPONENT;

COMPONENT timer IS
        PORT (  RD, WR, A0, CS, clk, gate : IN std_logic;
                val_out, charg_d_out : OUT std_logic;
                D : IN std_logic_vector(7 downto 0);
                DOUT : OUT std_logic_vector(7 downto 0);
                d_out_latch : OUT std_logic_vector(15 downto 0));
END COMPONENT;
-- signals declaration
SIGNAL CS, RD, WR, A0, gate, CLK, val_out, charg_d_out: std_logic;
SIGNAL D, DOUT: std_logic_vector(7 downto 0);
SIGNAL d_out_latch: std_logic_vector(15 downto 0);

```

```
BEGIN
--Inputs
        CS <= KEY(0);
        RD <= SW(0);
        WR <= SW(1);
        A0 <= SW(2);
        gate <= SW(3);
        D <= SW(9 downto 4)&"00";

-- Display LEDs
        LED(0) <= RD;
        LED(1) <= WR;
        LED(2) <= A0;
        LED(3) <= gate;
        LED(4) <= CS;
        LED(7) <= charg_d_out;
        LED(8) <= val_out;
        LED(9) <= clk;
-- Other LEDs off
        LED(6 downto 5) <= "00";

-- Divide input frequency 50 MHz by 2 exp 24
Divide : Dvide_Freq PORT MAP ( in_clk => CLOCK_50, out_cpt23 => clk);

-- Display 7 segments
-- LSB of DOUT  on Seg 0
        DCB1 : DCB PORT MAP (datai => DOUT(3 downto 0), segm => HEX0);
-- MSB of DOUT  on Seg 1
        DCB2 : DCB PORT MAP (datai => DOUT(7 downto 4), segm => HEX1);
-- d_buf_out on Seg 2 to 5
        DCB3 : DCB PORT MAP (datai => d_out_latch(15 downto 12), segm => HEX5);
        DCB4 : DCB PORT MAP (datai => d_out_latch(11 downto 8), segm => HEX4);
        DCB5 : DCB PORT MAP (datai => d_out_latch(7 downto 4), segm => HEX3);
        DCB6 : DCB PORT MAP (datai => d_out_latch(3 downto 0), segm => HEX2);
-- Instance of timer
        timer1: timer PORT MAP (RD, WR, A0, CS, clk, gate, val_out, charg_d_out, D, DOUT, d_out_latch);
END arch_top;
```

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITE BRETAGNE LOIRE

Cti Commission des Titres d'Ingénieur

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR, DE LA RECHERCHE ET DE L'INNOVATION