

Labo Timer

Version 2.4 du 1er mai 2024

Guillaume Gonin & Rafael Dousse



HAUTE ÉCOLE
D'INGÉNIERIE
ET DE GESTION
DU CANTON
DE VAUD

« Le fossé séparant théorie et pratique est moins large en théorie qu'il ne l'est en pratique »

Table des matières

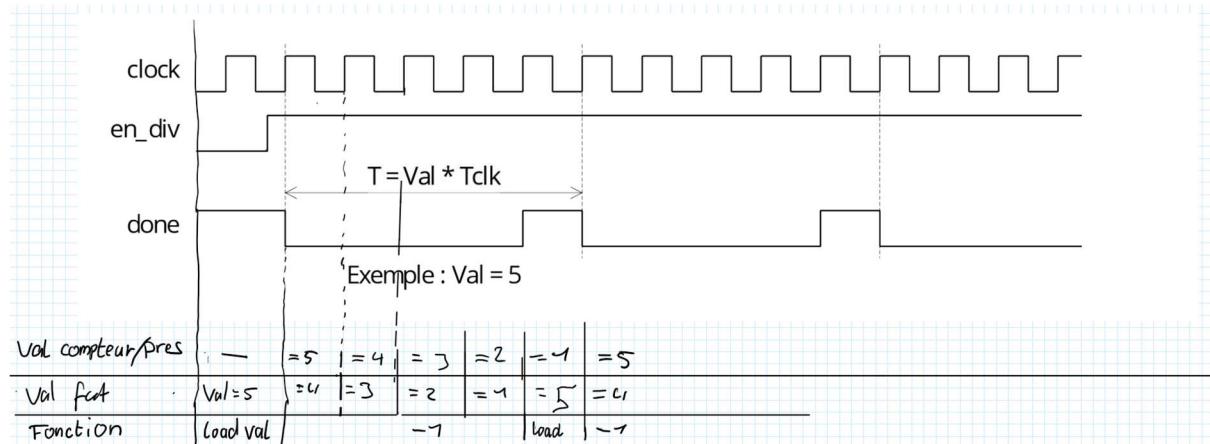
1	Introduction.....	1
2	Etapes conception.....	1
3	Schémas	2
4	Manipulation requise pour la simulation.....	3
5	Vérification	3
6	Réponses aux étapes et questions.....	4
7	Log de simulation	5
8	Compilation	7
9	Etape d'intégration et de validation sur la carte.....	7
10	Conclusion	9
11	Annexes.....	10

1 Introduction

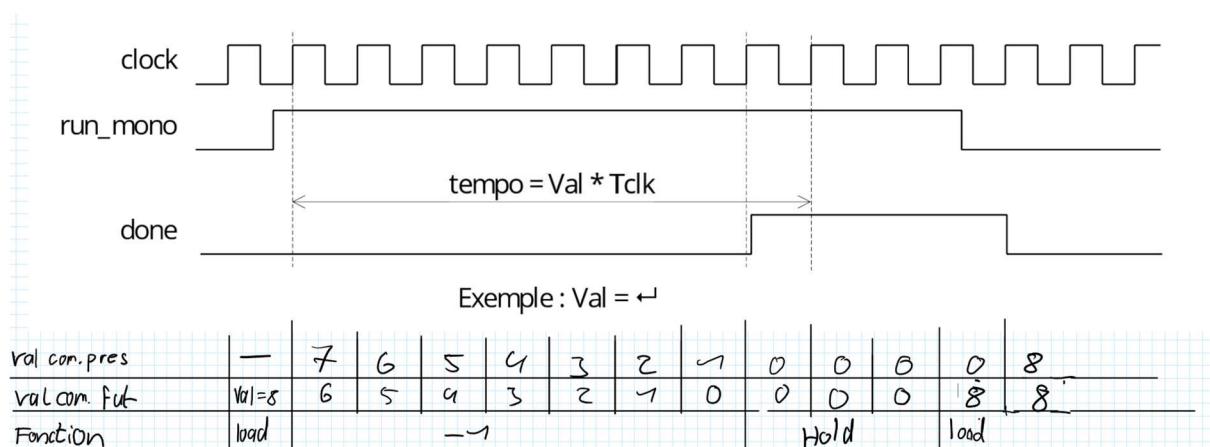
Le but principal de ce labo était de créer un timer (avec mode monostable et diviseur) en VHDL. Il nous a été demandé de créer le schéma conceptuel pour nous permettre d'apprendre davantage comment conceptualiser un système séquentiel.

2 Etapes conception

Notre première idée a été d'avoir cherché à faire compteur qui décrémente la valeur, ainsi on peut juste charger la valeur et comparer avec une valeur constante. Nous avons donc basé nos optimisations sur cette idée. Pour ce faire, nous avons fait un chronogramme avec les fonctions et les états interne au registre (q_{pres} et q_{fut}).



Ce chronogramme représente la fonction diviseur.



Ce chronogramme représente la fonction monostable.

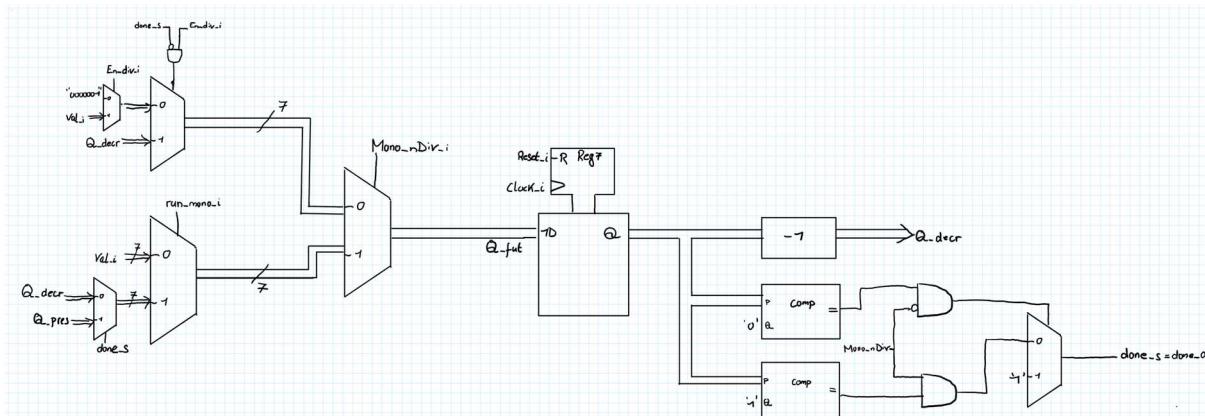
En parallèle, nous avons aussi établie la table des fonctions synchrone. On a pu ensuite faire le schéma de notre système.

Fonction synchrone : Load , Hold, décrementation

Fonction asynchrone : Reset

Mono_nDir	endir	run_mono	Pres	futur	Fonctions
0	0	—	—	val	Load
0	1	—	=1	val	Load
0	1	—	others	pres-1	-1
1	—	0	—	val	Load
1	—	1	0	pres	Hold
1	—	1	others	pres-1	-1

3 Schémas

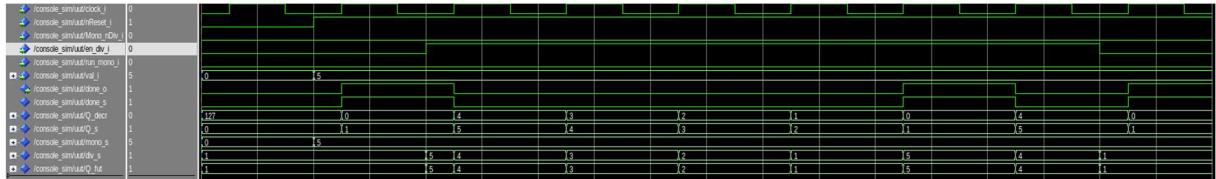


On peut remarquer l'utilisation d'un seul registre et additionneur. Néanmoins, nous avons deux comparateurs. On a décidé d'utiliser une décrémentation car ainsi nous pouvons juste charger la valeur au moment de commencer la boucle.

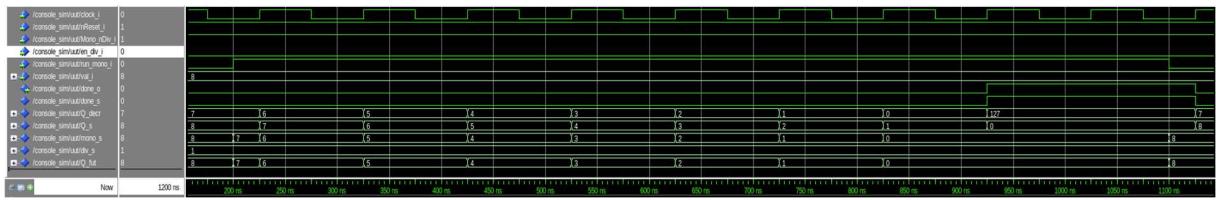
4 Manipulation requise pour la simulation

On a commencé par simuler manuellement en utilisant la console reds pour avoir une idée de la validité de notre solution. Voici les chronogrammes obtenus :

Pour le diviseur :



Pour le monostable :



NB : pour tous les chronogrammes de ce rapport le signal Q_pres est nommé Q_s ceci est dû fait que nous avons commenté le code après la création des chronogrammes et que l'on a renommé Q_s en Q_pres pour le rendre plus compréhensible.

Après avoir été satisfait de nos résultats, on a obtenu le script de simulation automatique et nous avons remplacé le fichier de même nom dans le dossier src_tb (nous avons regardé le script run_timer_top_tb.tcl et comp_timer.tcl) et nous avons ensuite dû supprimer un bout d'une commande dans run_timer_top_tb.tcl pour faire fonctionner la simulation (à savoir l'argument "novopt" d'une commande qui faisait une erreur) comme nous l'a demandé l'assistant.

5 Vérification

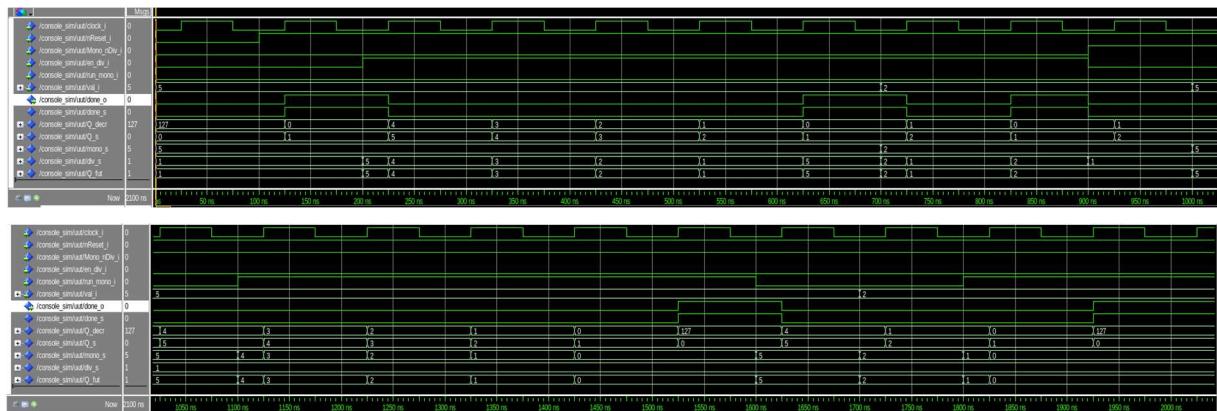
Nous avons établi le tableau suivant afin de tester notre solution :

nReset	Mono_nDiv	Run_mono	En_div	Val	Done attendu	Résultat
0	-	-	-	-	0	OK
1	0	-	0	-	1	OK
1	0	-	1	5	0 puis 1 après 4 coups de clock puis re-commence	OK
1	0	-	1	2	0 puis 1 après 1 coup de clock puis re-commence	OK
1	1	0	-	-	0	OK
1	1	1	-	5	0 puis 1 après 5 coups de clock	OK
1	1	0 (à la suite du test précédent)	-	5	1 puis 0 après 1 coups de clock	OK
1	1	1	-	2	0 puis 1 après 2 coups de clock	OK

Après test de la simulation automatique et en ayant discuté avec notre professeur, nous avons réalisé que nous avions mal interprété les exigences qui nous était demandé concernant le comportement du signal de sortie (done) qui devait, dans le cas du diviseur, et s'il n'était pas activé, être actif. Nous avons modifié notre solution ainsi que le tableau ci-dessus pour être en adéquation avec cette exigence.

De même le mode monostable contenant une petite erreur de conception car nous l'activions un coup d'horloge trop tôt nous avons donc modifié notre schéma pour tester le cas de la valeur 0 dans ce mode au lieu du cas 1.

Suite à tous cela, notre solution fut validée par la simulation automatique. Notre banc de test précédemment présentée a lui aussi été validé comme le prouve le chronogramme suivant :



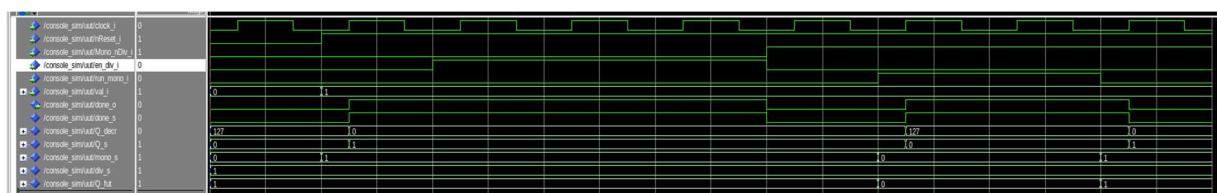
6 Réponses aux étapes et questions

"Indiquez quel est le comportement de votre solution dans le cas où Val vaut 0 et 1. Vous donnerez une explication théorique et une preuve par la simulation."

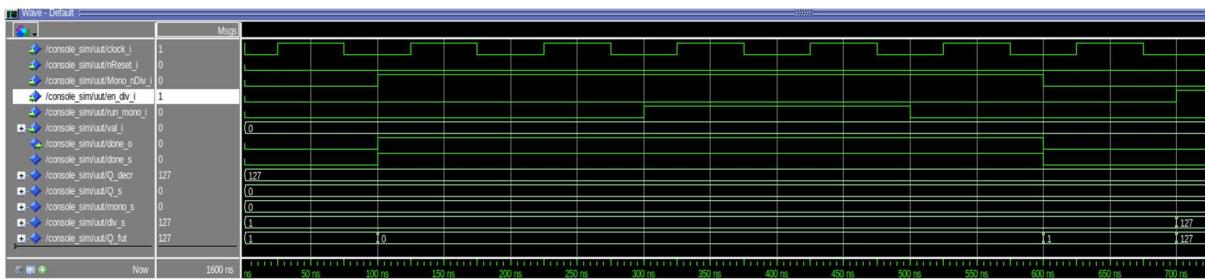
Dans un premier temps val sera chargée dans le système puis Q sera égal à 0 ou 1. Le cas de 1 fera dans la fonction diviseur (et si elle est activée car sinon juste charge 1 et done = 1) donner la valeur 1 à done car on test si Q = 1 ce qui sera toujours le cas (car ensuite on va recharger la valeur à 1). Dans la fonction monostable, on aura en premier lieu un coup de clock avec done à 0 puis Q = 0 et donc done = 1.

Dans le cas du 0, on aura dans la fonction monostable, la même chose que la fonction diviseur au cas 1, c'est-à-dire un done toujours égal à 1. Pour la fonction diviseur, on aura done égal à 0 car la valeur du compteur n'est pas égale à 0 et donc une décrémentation, vu que c'est un nombre non signé sur 7 bits on va passer de 0 à 127.

Le cas 1 :



Le cas 0 :



7 Log de simulation

```

vsim work.timer_top_tb

# Start time: 19:16:53 on Apr 26,2024

# ** Note: (vsim-8009) Loading existing optimized design _opt

# Loading std.standard

# Loading std.textio(body)

# Loading ieee.std_logic_1164(body)

# Loading ieee.numeric_std(body)

# Loading work.timer_top_tb(struct)#1

# Loading work.timer_top(timer)#1

# Loading work.timer_top_tester(test_bench)#1

# ** Warning: (vsim-8713) Suspect default binding for component instance 'tst'.

# Entity has no generics at all; the component has generic:

#      mode_normal_optionnel

# Time: 0 ns Iteration: 0 Instance: /timer_top_tb/tst File: ../src_tb/timer_top_tb.vhd Line: 100
# ** UI-Msg: (vish-4014) No objects found matching '/timer_top_tb/tst/Erreur_s'.
# Error in macro ./../run_timer_top_tb.tcl line 37
# ** UI-Msg: (vish-4014) No objects found matching '/timer_top_tb/tst/Erreur_s'.
# while executing
# "add wave -divider Testbench"
run -all

# ** Note: Debut simulation

# Time: 0 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Warning: NUMERIC_STD."=". metavalue detected, returning FALSE
# Time: 0 ns Iteration: 0 Instance: /timer_top_tb/uut
# ** Warning: NUMERIC_STD."=". metavalue detected, returning FALSE
# Time: 0 ns Iteration: 0 Instance: /timer_top_tb/uut
# ** Warning: NUMERIC_STD."=". metavalue detected, returning FALSE
# Time: 0 ns Iteration: 1 Instance: /timer_top_tb/uut
# ** Warning: NUMERIC_STD."=". metavalue detected, returning FALSE

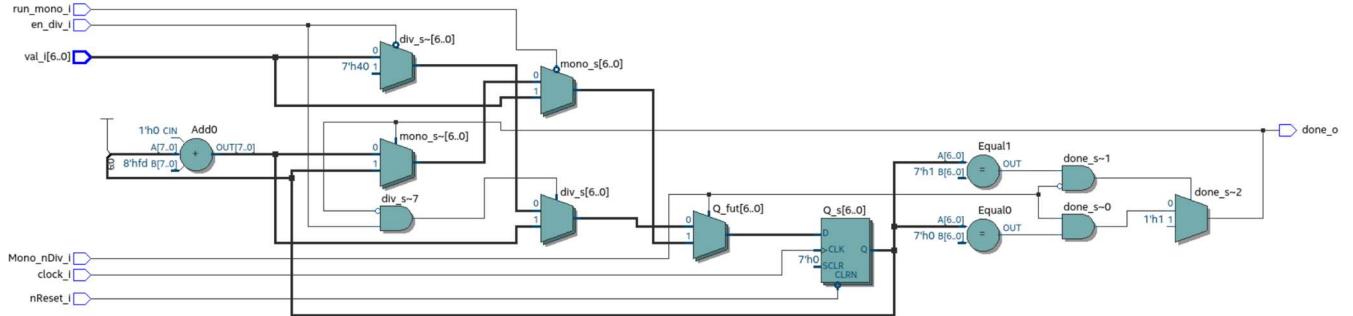
```

```
# Time: 0 ns Iteration: 1 Instance: /timer_top_tb/uut
# ** Note: Done pas verifie
# Time: 98 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 148 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 198 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 248 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 298 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 348 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 398 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 86998 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 87048 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 87098 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 87148 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: Done pas verifie
# Time: 87198 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Error: Lors verif Sortie, asynch
# Time: 128898 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Error: Lors verif Sortie, synch
# Time: 128948 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: >-----
#      >> Nombre d'erreurs detectees = 0
#      >>-----
# Time: 128952 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: >----->>
#      >>           >>
#      >>   Bravo votre systeme fonctionne   >>
#      >>           >>
```

```
#      >>----->>
# Time: 128952 ns Iteration: 0 Instance: /timer_top_tb/tst
# ** Note: >>Fin de la simulation
# Time: 128952 ns Iteration: 0 Instance: /timer_top_tb/tst
```

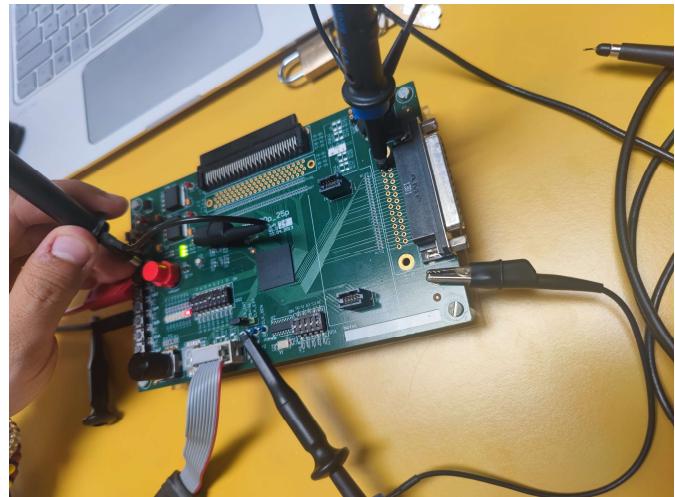
8 Compilation

Voici la vue RTL donné par Quartus :

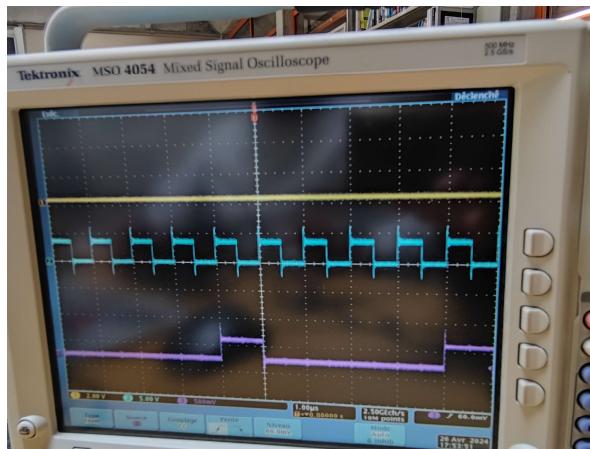


Une fois ces étapes complétées, nous pouvons vérifier l'intégration sur la carte et contrôler si notre minuterie fonctionne correctement. Pour comprendre la connexion des boutons et les réglages du programme, il est nécessaire de consulter les fichiers `max_top.vhd` et `maxv_top_pin_assignment.qsf`. Ces documents indiquent que pour saisir une valeur de minuterie, il est possible d'utiliser les interrupteurs situés juste au-dessus du JTAG. Les interrupteurs 0 à 6 sélectionnent la valeur, tandis que l'interrupteur 7 choisit le mode diviseur ou monostable. Les boutons, situés sur le côté gauche de la carte, activent les fonctions `en_div` et `run_mono` : le bouton 1 pour `en_div` et le bouton 8 pour `run_mono`. Le bouton de réinitialisation se trouve sur le bouton rouge de la carte, la fréquence d'horloge sur une broche nommée `CLK_INT`, et enfin, la sortie `done` sur la LED_0, à côté des interrupteurs. Il est également à noter que le signal `done` est redirigé vers le connecteur 25 broches, position 1, ce qui permet de prendre des mesures à l'oscilloscope et d'observer les résultats.

Pour valider notre programme, nous avons donc pris en compte trois signaux : celui de l'horloge, le signal `done` sur la broche 1 du connecteur 25 broches, et les boutons 1 et 8 pour activer notre programme. Voici une image du montage :



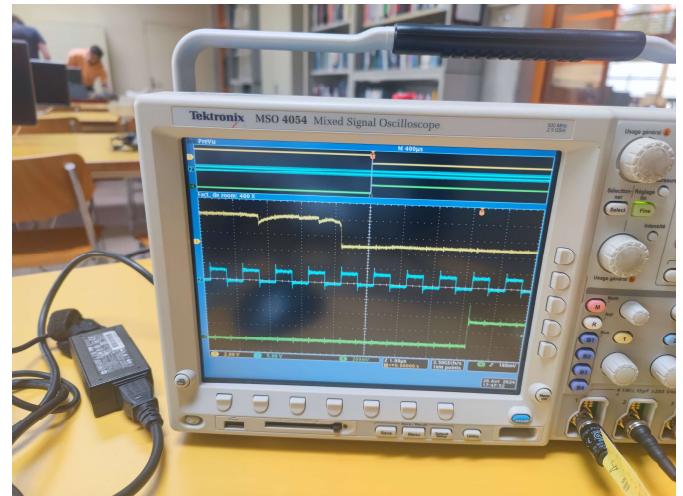
Et les 2 images suivantes sont les signaux observés sur l'oscilloscope.



Le signal jaune est le enable du bouton 1 et 8.

Le signal bleu est le clock.

Le signal violet sur la première image est le done du mode diviseur et le signal vert de la deuxième image est le done du mode monostable.



10 Conclusion

Nous avons pu expérimenter un grand nombre de concept au cours de ce labo. Nous avons particulièrement apprécié devoir chercher la solution la plus optimale (bien que ce ne fut pas explicitement demandé) et nous sommes satisfait du résultat (particulièrement l'utilisation d'un seul registre).

11 Annexes

timer_top.vhd

```
-----
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- Fichier      : timer_top.vhd
--
-- Description   :
--
-- Auteurs       : Rafael Dousse, Guillaume Gonin
-- Date          : 01.05.2024
-- Version       : 3.2
--
-- Utilise       : Manipulation Timer pour cours CSN
--
--



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

entity timer_top is
    port(
        clock_i      : in  std_logic; -- signale de l'horloge
        nReset_i     : in  std_logic; -- reset asynchrone (actif bas)
        Mono_nDiv_i  : in  std_logic; -- entrée de choix du mode
        en_div_i     : in  std_logic; -- enable du mode diviseur
        run_mono_i   : in  std_logic; -- enable du mode monostable
        val_i         : in  std_logic_vector(6 downto 0); -- valeur pour
notre timer
        done_o        : out std_logic -- signal de sortie (valeur dépend de
la fonction)
    );
end timer_top ;

architecture timer of timer_top is

    signal done_s : std_logic; -- valeur de sortie
    signal Q_pres : unsigned(6 downto 0); -- état présent (du registre
interne)
    signal Q_decr : unsigned(6 downto 0); -- état présent - 1
    signal mono_s : std_logic_vector( 6 downto 0); -- état futur pour le
monostable
    signal div_s : std_logic_vector( 6 downto 0); -- état futur pour le
diviseur
    signal Q_fut : std_logic_vector( 6 downto 0); -- état futur

begin

    -- définition des fonctions diviseur et monostable
    div_s <= std_logic_vector(Q_decr) when ((en_div_i = '1') and (done_s =
'0')) else
        val_i;
    mono_s <= val_i when run_mono_i = '0' else
        std_logic_vector(Q_pres) when done_s = '1' else
        std_logic_vector(Q_decr);

    -- on utilise la bonne fonction
    Q_fut <= mono_s when Mono_nDiv_i = '1' else div_s;

    -- notre registre (Flip Flop D)
```

```

process(nReset_i, clock_i)
begin
    if (nReset_i = '0') then
        Q_pres <= (OTHERS => '0');
    elsif rising_edge(clock_i) then
        Q_pres <= unsigned(Q_fut);
    end if;
end process;

-- utilisation des propriétés de unsigned pour décrire notre
additionneur (val-1)
Q_decr <= Q_pres - 1;
-- définition de la fonction done
done_s <= '1' when Q_pres = 1 or en_div_i = '0' else '0';
done_o <= done_s;
end timer;

```

timer_top_tester.vhd

```

-----
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- Fichier      : Timer_top_tester_Test_Bench.vhd
--
-- Description   :
--
-- Auteur        : Messerli
-- Date          : 07.12.2010
-- Version       : 0.0
--
-- Utilise       : Ce fichier est genere automatiquement par le logiciel
--                  : \\"HDL Designer Series HDL Designer\\".
--
-- | Modifications |
-----
-- Ver  Auteur  Date      Description
-- 1.0  GAA     03.12.2015  Correction de la partie monostable :
--                         for J in 1 to Val_v-2 loop ==>
--                         for J in 1 to Val_v-1 loop.
-- 1.1  FCC     22.11.2016  Adapter au nouvel énoncé 2016.
-- 1.2  EMI     02.12.2016  Modifier modulo Val_v a 128
-----

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

entity Timer_top_tester is
    port(
        Done_obs      : in  std_logic;
        Clock_sti     : out std_logic;
        En_Div_sti    : out std_logic;
        Mono_nDiv_sti : out std_logic;
        run_mono_sti  : out std_logic;
        Val_sti        : out std_logic_vector (6 downto 0);
        nReset_sti    : out std_logic
    );
    -- Declarations
end Timer_top_tester ;

architecture Test_Bench of Timer_top_tester is

```

```

constant Periode_c    : time := 100 ns;
constant Pulse_c      : time := 4 ns;  --duree impulsion sur erreur, ...
constant Tp_c          : time := 5 ns;  --temps de propagation

--signaux de base pour la simulation sequentiel (HORLOGE)
signal Sim_End_s : boolean := false;
signal Horloge_s : Std_Logic;           -- signal d'horloge
--signal Debut_Cycle : Std_Logic;       -- indique debut cycle horloge

--signaux de detection des erreurs (check), propre a l'application
signal Erreur_s : Std_Logic := '0';
shared variable Nbr_Err_v : Integer;

--signaux intermediaires pour la simulation, propre a l'application
signal Reset_sti : Std_Logic;
signal Done_ref : Std_Logic;

-----
-- Procedure permettant plusieurs cycles d'horloge
-- Le premier appel de la procedure termine le cycle precedent si celui-ci
-- n'etait pas complet (par exemple : si on a fait quelques pas de
-- simulation non synchronises avant, reset asynchrone, combinatoire,
...)

-----
procedure cycle (nombre_de_cycles : Integer := 1) is
begin
    for i in 1 to nombre_de_cycles loop
        wait until Falling_Edge(Horloge_s);
        wait for 2 ns; --assigne stimuli 2ns apres flanc montant
    end loop;
end cycle;

begin
-----
--Connexion signaux description a verifier
-----
Clock_sti <= Horloge_s;
nReset_sti <= not Reset_sti;
-----
--Process de generation de l'horloge
-----

process
begin
    while not Sim_End_s loop
        Horloge_s <= '1', '0' after Periode_c/2;
        wait for Periode_c;
    end loop;
    wait;
end process;

-----
-- Debut des pas de simulation
-----

process is
    variable Attente_v,Val_v : Natural;

    procedure test_reset_asynch is
    begin
        -- Test de la remise a zero asynchrone
        Reset_sti      <= '1';
        Mono_nDiv_sti <= 'X';

```

```

En_Div_sti      <= 'X';
run_mono_sti   <= 'X';
Val_sti         <= (others => '0');
Done_ref        <= '-'; --depend valeur initialisation cpt !
cycle(2);
Mono_nDiv_sti <= '1';
En_Div_sti     <= '1';
run_mono_sti   <= '1';
cycle(2);
Reset_sti      <= '0';
En_Div_sti     <= '0';
run_mono_sti   <= '0';
end test_reset_asynch;

begin

Nbr_Err_v := -2; -- initialise compteur d'erreur
                  --tester genere deux erreurs volontaires
                  -- total final = 0

report "Debut simulation";

test_reset_asynch;

-- Test du fonctionnement en diviseur
Mono_nDiv_sti <= '0';
wait until Rising_Edge(Horloge_s);
Done_ref <= '1'; --Done est a '1' au repos
Val_v:=2;
for I in 1 to 10 loop
    Val_sti <= Std_Logic_Vector(To_Unsigned(Val_v, Val_sti'length));
    cycle;
    --active le mode diviseur
    En_Div_sti <= '1';
    wait until Rising_Edge(Horloge_s);
    Done_ref <= '0';                                --Done doit etre desactive
    for J in 1 to Val_v-1 loop                      --attente Done inactif
        wait until Rising_Edge(Horloge_s);
    end loop;
    Done_ref <= '1';                                --Done actif pendant une
periode
        wait until Rising_Edge(Horloge_s); --attente d'une periode
        Done_ref <= '0';                                --Done doit etre desactive
        for J in 1 to Val_v-1 loop                      --attente Done inactif
            wait until Rising_Edge(Horloge_s);
        end loop;
        Done_ref <= '1';                                --Done actif pendant une
periode
        wait until Rising_Edge(Horloge_s); --attente d'une periode
        Done_ref <= '0';                                --Done doit etre desactive
        cycle;
        --desactive le mode diviseur
        En_Div_sti <= '0';
        wait until Rising_Edge(Horloge_s);
        Done_ref <= '1'; --Done est a '1' au repos
        --calcul prochaine valeur pour Val, pseudo aleatoire
        Val_v := ((7 * Val_v)/3)mod 128;   -- mod EMI: modulo 128 pour 7
bits
        cycle(3);
    end loop;

-- Test du fonctionnement en monostable
Mono_nDiv_sti <= '1';
Done_ref <= '-'; --etat de Done pas changer lors changement de mode
wait until Rising_Edge(Horloge_s);
Val_v := 2;

```

```

for I in 1 to 10 loop
    Val_sti <= Std_Logic_Vector(To_Unsigned(Val_v, Val_sti'length));

    cycle(2);

    --test monostable sans rearmement
    run_mono_sti <= '1';                                --arme le monostable
    wait until Rising_Edge(Horloge_s);
    Done_ref <= '0';                                     --Done doit etre desactive
    for J in 1 to Val_v-1 loop                          --attente delai monostable
        wait until Rising_Edge(Horloge_s);
    end loop;
    Done_ref <= '1';                                     --Done doit s'activer
    cycle(3);
    run_mono_sti <= '0';
    wait until Rising_Edge(Horloge_s);
    Done_ref <= '0';

    --test monostable avec rearmement
    Attente_v := Val_v/3 + 1;   --calcul d'une attente pour le
rearmement
    cycle(2);
    run_mono_sti <= '1';                                --demarrage delai du monostable
    wait until Rising_Edge(Horloge_s);
    Done_ref <= '0';                                     --Done doit etre desactive
    cycle(Attente_v);                                  --attendre un delai
    run_mono_sti <= '0';                                --rearme le monostable
    cycle;
    run_mono_sti <= '1';                                --demarrage delai du monostable
    wait until Rising_Edge(Horloge_s);
    for J in 1 to Val_v-1 loop                          --attente delai monostable
        wait until Rising_Edge(Horloge_s);
    end loop;
    Done_ref <= '1';                                     --Done doit s'activer
    cycle(3);                                         --Done reste actif
    run_mono_sti <= '0';
    wait until Rising_Edge(Horloge_s);
    Done_ref <= '0';

    --calcul prochaine valeur pour Val
    Val_v := (((7 * Val_v)/3)mod 31)+1;
    cycle(3);
end loop;

--2 erreurs provoquées: verification envoi message d'erreur et
activation signal Erreur
cycle(10);
Done_ref <= '1';
cycle;
Done_ref <= '0';

report ">>-----" & LF &
        "      >> Nombre d'erreurs detectees = " &
integer'image(Nbr_Err_v) & LF &
        "      >>-----";
if Nbr_Err_v = 0 then
    report ">>----->>" & LF &
            "      >>      >>" & LF
&
            "      >>  Bravo votre systeme fonctionne  >>" & LF
&
            "      >>      >>" & LF
&
            "      >>----->>";
elsif Nbr_Err_v < 0 then

```

```

        report ">>----->>" & LF &
                "           >> Tres bizarre, vous n'avez pas les deux erreurs
volontaire du TB!!!  >>" & LF &
                "           >>----->>";
    else
        report ">>----->>" & LF &
                "           >> Courage, vous avez encore quelques erreurs a
corriger     >>" & LF &
                "           >>----->>";
    end if;
    report ">>Fin de la simulation";

Sim_End_s <= true;
wait; --Attente infinie, stop la simulation

end process;

-- 
-- Process de verification
--

process
begin
    while not Sim_End_s loop
        wait until Falling_Edge(Horloge_s);
        wait for (Periode_c/2)- 2 ns;
        --Verification asynchrone
        if Done_ref = '-' then
            report"Done pas verifie";
        elsif Done_ref /= Done_obs then
            Erreur_s <= '1', '0' after Pulse_c;
            Nbr_Err_v := Nbr_Err_v + 1;
            report"Lors verif Sortie, asynch"
                severity ERROR;
        end if;
        wait until Rising_Edge(Horloge_s);
        wait for (Periode_c/2)- 2 ns;
        --Verification synchrone
        if Done_ref = '-' then
            report"Done pas verifie";
        elsif Done_obs /= Done_ref then
            Erreur_s <= '1', '0' after Pulse_c;
            Nbr_Err_v := Nbr_Err_v + 1;
            report"Lors verif Sortie, synch"
                severity ERROR;
        end if;
    end loop;
    wait;
end process;

end architecture Test_Bench;

```