Université de Bretagne Occidentale

Master 2 Informatique

Département Informatique

2020/2021

Système On-Chip

Détection de dépassement de temps d'exécution

Auteur :
William PENSEC

Auteur :
Timothé LANNUZEL

22 janvier 2021

FACULTÉ
DES SCIENCES
& TECHNIQUES

Sommaire

I Introduction

L'objectif de ce projet est de concevoir en VHDL un moniteur de temps d'exécution de tâches sur un processeur. En effet, sur un système temps réel, il est très important que les contraintes de temps soient respectées afin d'éviter tout problèmes. Le composant doit suivre l'exécution de chaques tâches et envoyer un signal d'interruption au processeur si l'une d'entre elles dépasse son échéance. La capacité maximale d'une tâche s'appelle le Worst Case Execution Time (WCET). En connaissant cette valeur, on sait si le processeur peut gérer le système ou s'il est nécessaire de le changer pour quelque chose de plus performant.

Afin de simplifier les simulations, nous avons fixé des deadlines de telle manière à ce que tout soit fini en 30 ns maximum lors de la simulation sur Vivado.

II Conception VHDL

Le projet s'est découpé en plusieurs étapes qui ont été de créer d'abord les différents modules qui composent le système puis de créer les fichiers de tests (testbench) de ces modules. La seconde étape est de regrouper ces modules afin de créer une IP sous Vivado qui pourra être utilisée ailleurs. Cette IP sera composée, comme on peut le voir sur l'image 1, du CPU, d'une mémoire, de compteurs (module chronomètre) et d'un composant permettant la communication avec le CPU par l'AXI.

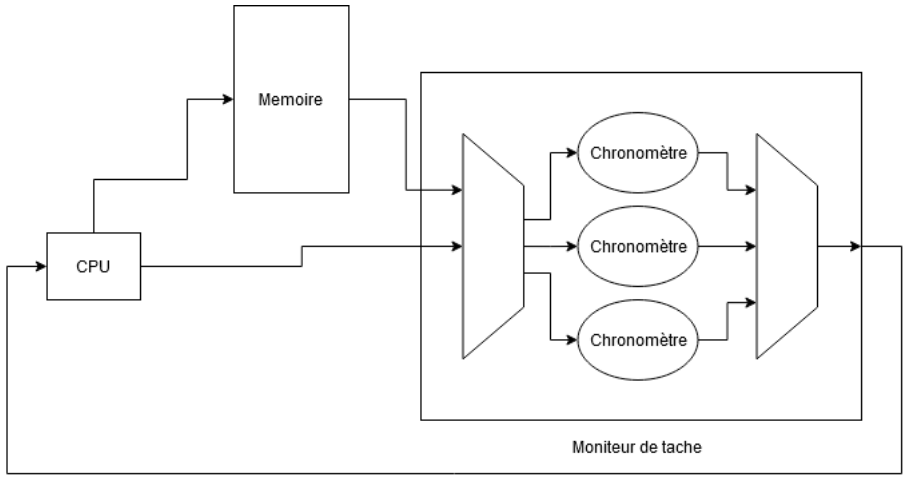

FIGURE 1 – Architecture générale

II.1 Chronomètre

L'image 2, à la page 3, représente le fonctionnement de manière schématique du module chronomètre-décompteur. Le code de cette partie est disponible dans l'archive ou sinon voir le code 1 à la page 8. Le chronomètre est lié à une horloge sur front montant `rising_edge(clk)`. Cela permet de contrôler les opérations un front sur deux pour aller un peu plus lentement. Autrement, il y a un port de démarrage/arrêt du chronomètre `startStop` qui permet comme son nom l'indique de démarrer ou stopper le module ; mais également un port afin de mettre en pause et de reprendre le timer `suspendResume`. Nous avons inclu un port de chargement `load` et de reset `reset` permettant de charger la valeur d'initialisation (valeur qui correspond à la durée du timer par exemple <10> périodes d'horloge) ou au contraire de mettre à 0 le timer de la tâche en cours.

Enfin, le dernier port qui est celui qui nous intéresse le plus est celui du `wcet`. Ce port est donc un tableau de 16 bits. C'est dans ce tableau que l'on va enregistrer la valeur du `Worst Case Execution Time` (`WCET`). C'est cette valeur qui sera chargée par le port load en mémoire et c'est cette valeur qui servira à décompter le temps avant d'envoyer si besoin l'interruption au processeur si le `WCET` arrive à 0 dans le timer.

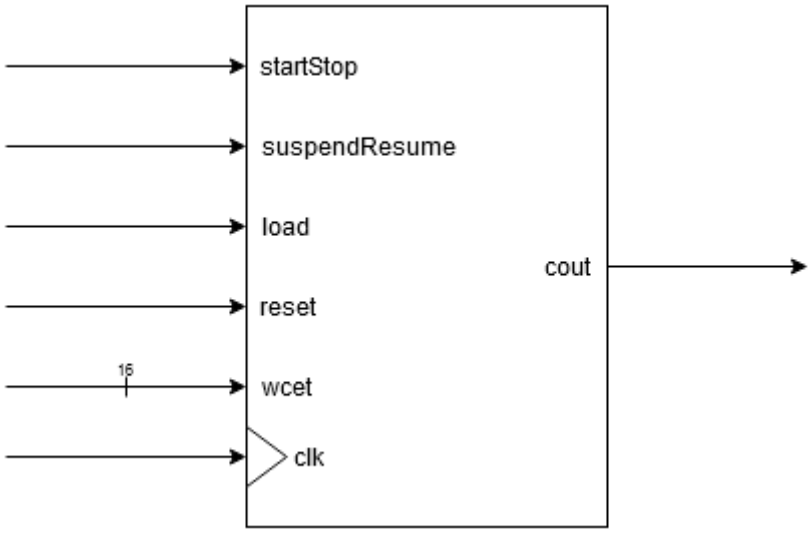

FIGURE 2 – Bloc chronomètre

II.2 TestBench Chronomètre

Le code du test bench est le code 2 à la page 9. Il s'articule de la manière suivante : tout d'abord comme d'habitude nous appellons le component à qui il

fait référence, c'est à dire le chronomètre. Puis, on crée les signaux nécessaires pour assigner des valeurs aux ports du composant. Ensuite dans l'architecture comportementale du composant test, on affecte des valeurs aux signaux. Nous avons décidé de faire une horloge avec une période de 1 ns afin d'avoir quelque chose de rapide. La valeur `startStop_ch` est initialisée à 0 et passe à 1 après 5 ns c'est à dire que le chronomètre ne démarrera qu'après 5 ns d'exécution de programme, cette valeur a été mise seulement dans un but de test mais en soit doit être initialisée à 1 lors de la création du chronomètre. La valeur `load_ch` correspondant au chargement du `WCET` en mémoire ; il est initialisé à 1, c'est à dire qu'on charge en mémoire la donnée dès qu'elle est disponible puis on passe cette valeur à 0 car on désire arrêter la mise en mémoire de la valeur afin de passer à la décrémentation. La valeur du `reset` est laissé à 0 car nous n'en avons pas besoin du tout mais si on passe cette valeur à 1 alors la donnée est mise à 0 comme convenu ! Enfin, la valeur du `wcet_ch` est initialisée à 7.

Un exemple d'exécution est proposé à l'image 4 à la page 6. On distingue sur l'image toutes les étapes citées dans le paragraphe précédent.

II.3 Moniteur de tâches

Le moniteur de tâches, comme on peut voir sur l'image 3 à la page 5, possède 1 port de sortie et 5 ports d'entrées. Comme sur le chronomètre, il y a l'horloge et le reset. Le port `id_task` correspond à l'ID sur 4 bits de la tâche en cours, par exemple, si `id_task = 0001` alors ça veut dire que la tâche en train de s'exécuter est la tâche 1. Le port `mess_task` correspond au message d'exécution, il peut prendre des valeurs entre 0 et 5. Le détail des valeurs possible est :

0 : 'stop'

1 : 'start'

2 : 'load'

3 : 'suspend'

4 : 'resume'

5 : 'reset'

Le port `wcet_task` correspond à la donnée de temps maximal d'exécution qui doit ensuite être décrémenté lors de l'exécution de la tâche par le module chronomètre. Enfin, le port de sortie `counter_interupt` est à 0 tout le temps et doit rester à 0 car s'il passe à 1 cela veut dire qu'une tâche à dépasser son temps d'exécution et donc un signal d'interruption doit être envoyé au processeur !

FIGURE 3 – Bloc représentant le moniteur de tâches

II.4 TestBench Moniteur de tâches

Le testbench du moniteur de tâches sert juste à simuler le programme en créant 2 tâches (la tâche 0 et la tâche 1) et en les exécutant en regardant si le chronomètre arrive à 0 ou non. L'image 5 à la page 7 permet de voir que les 2 tâches que l'on a exécuté s'exécutent bien et que la tâche 0 ne se termine pas alors que la tâche 1 est fini en une période d'horloge.

La tâche 0 ne respecte pas son WCET car la valeur de `compteur_test` arrive à 0 au bout de 18 ns, le signal d'interruption est bien envoyé aussi car la variable `compteur` passe bien à 1 lorsque le chronomètre arrive à 0.

Les tâches sont contrôlées avec la variable `message` qui change de valeur de temps en temps pour tester un cas possible d'exécution.

III Résultats

III.1 Chronomètre

Comme on peut voir sur cette image, le signal `clk_ch` correspondant à l'horloge du CPU oscille chaque nanoseconde.

Dans l'ordre du temps, tout d'abord à 0 ns on a en mémoire le wcet de la

tâche, cette valeur est initialisée dans la variable de sortie test 'cout_test_ch' à 1 ns qui correspond au front montant de l'horloge. Après 4 ns, le `load_ch` passe à 0 ce qui permet d'arrêter la mise en mémoire du wcet en continu, si cette variable était resté à 1 le chronomètre ne se serait jamais lancé. La variable `startStop` lance l'exécution du module à 5 ns et le décompte commence aussitôt sur la sortie. Enfin, au bout de 17 ns, on voit que la variable `cout_ch` passe à 1 ce qui signifique que le chronomètre est arrivé à 0 et qu'il est nécessaire au module supérieur de lancer le signal d'interruption au processeur.

Nous avons ajouté un port de sortie `cout_test_ch` qui permet de lire les résultats plus simplement.

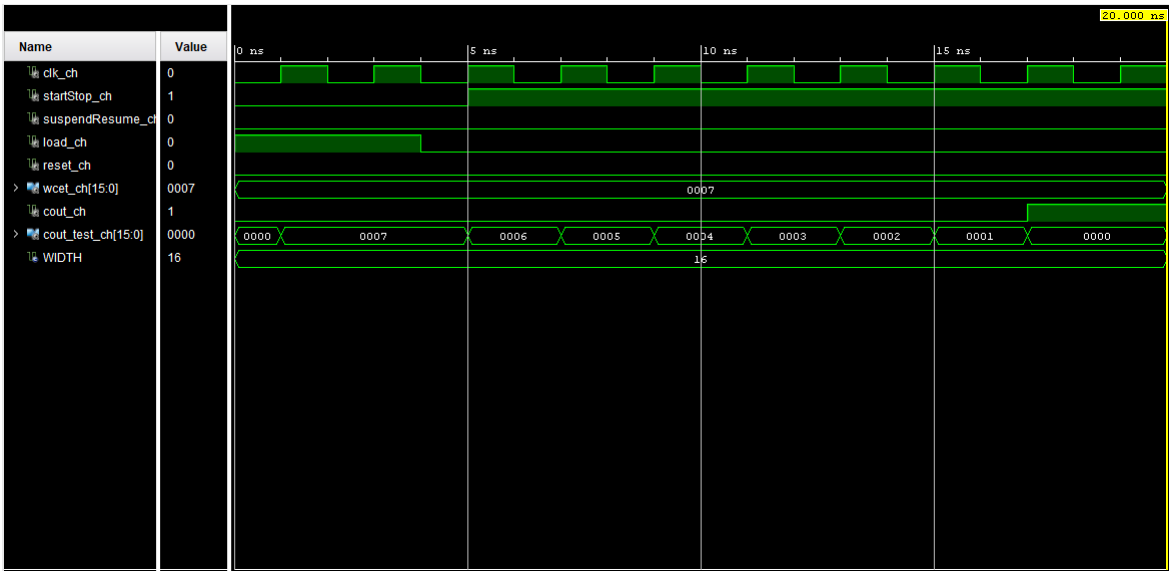

FIGURE 4 – Testbench chronomètre

III.2 Moniteur de tâches

L'image suivante correspond à l'exécution du composant `taskMonitor` avec 2 tâches en entrée possédant respectivement un WCET de 3 et de 5.

L'ordre d'exécution par rapport au temps est tout d'abord le message est à 0 donc l'état de la tâche 0 comme indiqué avec l'ID à 0 est sur stop, à 1 ns (période de l'horloge), le WCET de la tâche 0 est initialisé à 0 car l'instruction de chargement 'load' n'a pas été reçu. Cette instruction arrivera au bout de 2 ns. Le troisième message reçu est le 'start' à 3 ns mais le programme ne commencera qu'au tick suivant c'est à dire 4 ns. Le décompte commence donc de 3 puis passe à 2 avant d'être stoppé à 7 ns pour laisser la main à la tâche 1 qui s'exécute de 10 ns à 12 ns et puis la tâche 0 reprend jusqu'à arriver à 0 au bout de 17 ns et donc d'envoyer

le signal d'interruption immédiatement au processeur. Nous pouvons voir que la tâche 0 reprend bien à 2 et non pas au départ donc on a bien une sauvegarde de la valeur et un chargement correct sans le reset.

Nous avons ajouté un port de sortie `compteur_test` qui prend la valeur de `cout_test_ch` qui permet de lire les résultats de la tâche courante plus simplement.

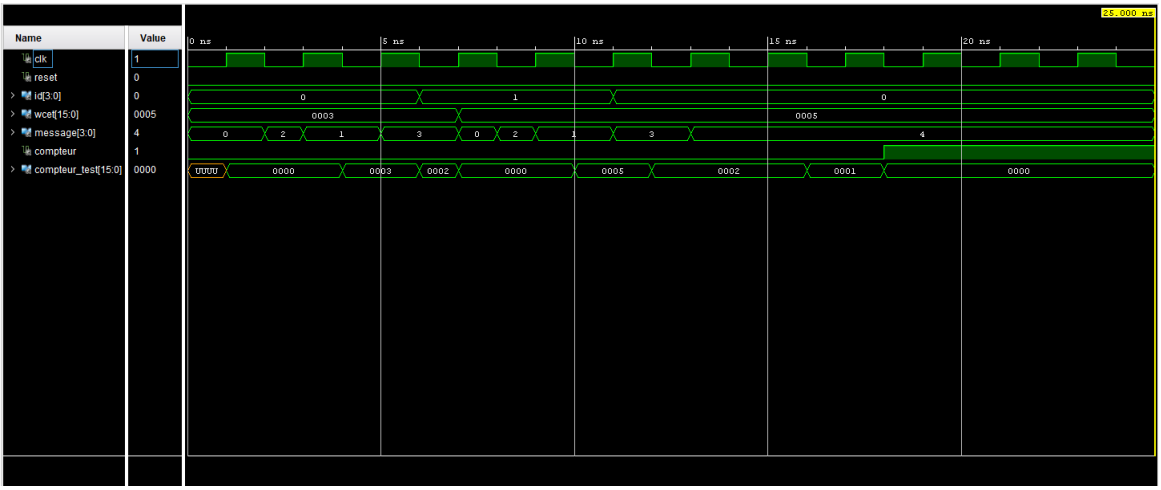

FIGURE 5 – Testbench moniteur de tâches

IV Continuité du projet

La première étape serait de réaliser l'IP sur Vivado avec les branchements AXI et ensuite de générer le block design complet

La deuxième étape serait de générer le fichier 'wrapper' dont le rôle est de connecter les Entrées/Sorties sur les broches physiques du FPGA.

La troisième étape serait de lancer la synthèse et implémentation du programme Vivado, créer le bitstream.

Enfin, la dernière étape serait la réalisation d'un programme en C dans le but d'interfacer ce même programme et le code VHDL afin de générer les tâches et envoyer ces id, messages et autres données au code moniteur de tâches puis au chronomètre. Et enfin, exécuter le code sur un FPGA afin de tester son bon fonctionnement.

V Code

V.1 Chronomètre

```
1  ----------------------------------------------------------------------------------
2  -- Engineer: Timothé LANNUZEL & William PENSEC
3  -- Create Date: 03.01.2020 16:01:00
4  -- Module Name: chronometer - Behavioral
5  -- Project Name: Détection de dépassement de temps d'exécution
6  -- Revision: 1.2
7  ----------------------------------------------------------------------------------
8
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 use IEEE.STD_LOGIC_UNSIGNED.ALL;
13 use IEEE.NUMERIC_STD.ALL;
14
15 entity chronometer is
16     generic(
17         WIDTH : integer := 16
18     );
19
20     Port (
21         clk : in std_logic;
22         startStop : in std_logic;
23         suspendResume : in std_logic;
24         load : in std_logic;
25         reset : in std_logic;
26         wcet : in std_logic_vector(WIDTH - 1 downto 0);
27         cout : out std_logic;
28         cout_test : out std_logic_vector(WIDTH - 1 downto 0)
29     );
30 end chronometer;
31
32 architecture Behavioral of chronometer is
33     signal curr_value : std_logic_vector(WIDTH - 1 downto 0) := (
       others => '0');
34 begin
35     cout_test <= curr_value;
36
37     compteur : process(clk)
38     begin
39         if rising_edge(clk) then
40             if load = '1' then
41                 curr_value <= wcet;
42             elsif reset = '1' then
43                 curr_value <= (others => '0');
44             elsif startStop = '1' and suspendResume = '0' and
       unsigned(curr_value) /= 0 then  -- 1 start | 0 resume
45                 curr_value <= std_logic_vector(unsigned(curr_value) -
        1);
```

```
46              end if;
47          end if;
48
49      end process;
50
51      test : process(curr_value)
52      begin
53          if startStop = '1' and unsigned(curr_value) = 0 then
54              cout <= '1';
55          else
56              cout <= '0';
57          end if;
58          if reset = '1' then
59              cout <= '0';
60          elsif load = '1' then
61              cout <= '0';
62          end if;
63      end process;
64
65  end Behavioral;
```

Listing 1 – Chronomètre

V.2 TestBench Chronomètre

```
1   ----------------------------------------------------------------------------------
2   -- Engineer: Timothé LANNUZEL & William PENSEC
3   -- Create Date: 05.01.2020 17:21:00
4   -- Module Name: chronometer_tb - Behavioral
5   -- Project Name: Détection de dépassement de temps d'exécution
6   -- Revision: 1.0
7   ----------------------------------------------------------------------------------
8
9
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.STD_LOGIC_UNSIGNED.ALL;
13  use IEEE.NUMERIC_STD.ALL;
14
15  entity chronometer_tb is
16  --  Port ( );
17  end chronometer_tb;
18
19  architecture Behavioral of chronometer_tb is
20      component chronometer is
21          generic(
22              WIDTH : integer := 16
```

```
23          );
24          Port(
25              clk : in std_logic;
26              startStop : in std_logic;
27              suspendResume : in std_logic;
28              load : in std_logic;
29              reset : in std_logic;
30              wcet : in std_logic_vector(WIDTH - 1 downto 0);
31              cout : out std_logic;
32              cout_test : out std_logic_vector(WIDTH - 1 downto 0)
33          );
34      end component chronometer;
35
36      constant WIDTH : integer := 16;
37      signal clk_ch : std_logic := '0';
38      signal startStop_ch : std_logic;
39      signal suspendResume_ch : std_logic;
40      signal load_ch : std_logic;
41      signal reset_ch : std_logic;
42      signal wcet_ch : std_logic_vector(WIDTH - 1 downto 0);
43      signal cout_ch : std_logic;
44      signal cout_test_ch : std_logic_vector(WIDTH - 1 downto 0);
45
46  begin
47      clk_ch <= not clk_ch after 1 ns;
48      startStop_ch <= '0', '1' after 5 ns;
49      suspendResume_ch <= '0'; -- always active
50      load_ch <= '1', '0' after 4 ns; -- 0 to stop data's loading
51      reset_ch <= '0'; -- never reseted
52      wcet_ch <= std_logic_vector(to_unsigned(7, 16));
53
54      iut : entity work.chronometer(Behavioral)
55      Port map(
56          clk => clk_ch,
57          startStop => startStop_ch,
58          suspendResume => suspendResume_ch,
59          load => load_ch,
60          reset => reset_ch,
61          wcet => wcet_ch,
62          cout => cout_ch,
63          cout_test => cout_test_ch
64      );
65  end Behavioral;
```

Listing 2 – TestBench chronomètre

V.3 Moniteur de tâches

```
1  ----------------------------------------------------------------------------------
2  -- Engineer: Timothé LANNUZEL & William PENSEC
3  -- Create Date: 10.12.2020 16:22:12
4  -- Module Name: taskMonitor - Behavioral
5  -- Project Name: Détection de dépassement de temps d'exécution
6  -- Revision: 1.2
7  ----------------------------------------------------------------------------------
8
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 use IEEE.STD_LOGIC_UNSIGNED.ALL;
13 use IEEE.NUMERIC_STD.ALL;
14
15 entity taskMonitor is
16     Port (
17         clk_in : in std_logic;
18         reset_in : in  std_logic;
19
20         id_task : in std_logic_vector(3 downto 0);
21         wcet_task : in std_logic_vector(15 downto 0);
22         mess_task : in std_logic_vector(3 downto 0);
23
24         counter_interupt : out std_logic;
25         counter_interupt_test: out std_logic_vector(16 - 1 downto 0)
26     );
27 end taskMonitor;
28
29 architecture Behavioral of taskMonitor is
30
31 ---------------------------------------------------------
32 ------------------------Signal---------------------------
33 ---------------------------------------------------------
34     --signaux pour les chronos
35     signal sigstartStop :  std_logic_vector(3 downto 0);
36     signal sigsuspendResume :  std_logic_vector(3 downto 0);
37     signal sigload :  std_logic_vector(3 downto 0);
38     type register_array is array ( 3 downto 0 ) of std_logic_vector(
       15 downto 0 );
39     signal sigtaskwcet : register_array;
40
41     signal curChrono: std_logic_vector(3 downto 0) := (others => '0')
       ;  -- pour connaitre sur quel chrono on est
42     signal interrupt_timer : std_logic_vector(3 downto 0);
43
44     signal curcounter_interupt_test : register_array;
45     signal currTaskId : integer :=0;
46     signal sigReset  :  std_logic_vector(3 downto 0);
```

```
47
48  ------------------------------------------------------------
49  -----------------------Components---------------------------
50  ------------------------------------------------------------
51      component chronometer
52          generic(
53              WIDTH : integer := 16
54          );
55          Port (
56              clk : in std_logic;
57              startStop : in std_logic;
58              suspendResume : in std_logic;
59              load : in std_logic;
60              reset : in std_logic;
61              wcet : in std_logic_vector(WIDTH - 1 downto 0);
62              cout : out std_logic;
63              cout_test : out std_logic_vector(WIDTH - 1 downto 0)
64          );
65      end component;
66
67  begin
68      ------------------------------------------------------------
69      -----------------------Port map-----------------------------
70      ------------------------------------------------------------
71      generate_chrono : for i in 0 to 3 generate
72          instChrono : entity work.chronometer(Behavioral)
73              port map(
74                  clk => clk_in,
75                  startStop => sigstartStop(i),
76                  suspendResume => sigsuspendResume(i),
77                  load => sigload(i),
78                  reset => sigReset(i),
79                  wcet => sigtaskwcet(i),
80                  cout => interrupt_timer(i),
81                  cout_test => curcounter_interupt_test(i)
82
83              );
84      end generate;
85
86      tache : process(clk_in)
87      begin
88                --Changement du WCET
89          if reset_in = '1' then
90              loop1 : for i in 0 to 3 LOOP
91                      sigstartStop(i) <= '0';     --STOP
92                      sigload(i) <= '0';
93                      sigsuspendResume(i) <= '0';
94                      sigReset(i) <= '1';
95              END LOOP loop1;
```

```
96          else
97              case conv_integer ( mess_task ) is          --Changement
    de message
98                  when 0 =>
99                      sigstartStop(currTaskId) <= '0';    --STOP
100                     sigload(currTaskId) <= '0';
101                     sigsuspendResume(currTaskId) <= '0';
102                     sigReset(currTaskId) <= '0';
103                 when 1 =>
104                     sigstartStop(currTaskId) <= '1';    --START
105                     sigload(currTaskId) <= '0';
106                 when 2 =>
107                     sigtaskwcet(currTaskId) <= wcet_task;
108                     sigload(currTaskId) <= '1';         --LOAD
109                 when 3 =>
110                     sigsuspendResume(currTaskId) <= '1';--SUSPEND
111                 when 4 =>
112                     sigsuspendResume(currTaskId) <= '0';--RESUME
113                 when 5 =>
114                     sigReset(currTaskId) <= '1';        --RESET
115                     sigstartStop(currTaskId) <= '0';
116                     sigload(currTaskId) <= '0';
117                     sigsuspendResume(currTaskId) <= '0';
118                 when others =>
119                     --ne rien faire
120             end case;
121         end if;
122         if reset_in = '1' then
123              counter_interupt <= '0';
124         elsif interrupt_timer(currTaskId) = '1' then
125             counter_interupt <= '1';
126         else
127             counter_interupt <= '0';
128         end if;
129         counter_interupt_test <= curcounter_interupt_test(currTaskId)
    ;
130     end process;
131
132     tache2 : process(id_task)
133     begin
134         currTaskId <= to_integer (unsigned(id_task));
135
136     end process;
137
138 end Behavioral;
```

Listing 3 – Moniteur de tâches

V.4 TestBench Moniteur de tâches

```
1  ----------------------------------------------------------------------------------
2  -- Engineer: Timothé LANNUZEL & William PENSEC
3  -- Create Date: 19.01.2021 12:09:40
4  -- Module Name: taskMonitor_tb - Behavioral
5  -- Project Name: Détection de dépassement de temps d'exécution
6  -- Revision: 1.2
7  ----------------------------------------------------------------------------------
8
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 use IEEE.STD_LOGIC_UNSIGNED.ALL;
13 use IEEE.NUMERIC_STD.ALL;
14
15 entity taskMonitor_tb is
16 --  Port ( );
17 end taskMonitor_tb;
18
19 architecture Behavioral of taskMonitor_tb is
20     component taskMonitor is
21         Port (
22             clk_in : in std_logic;
23             reset_in : in  std_logic;
24
25             id_task : in std_logic_vector(3 downto 0);
26             wcet_task : in std_logic_vector(15 downto 0);
27             mess_task : in std_logic_vector(3 downto 0);
28
29             counter_interupt : out std_logic;
30             counter_interupt_test: out std_logic_vector(15 downto 0)
31
32         );
33     end component taskMonitor;
34
35     signal clk : std_logic := '0';
36     signal reset : std_logic;
37     signal id : std_logic_vector(3 downto 0);
38     signal wcet : std_logic_vector(15 downto 0);
39     signal message : std_logic_vector(3 downto 0);
40     signal compteur : std_logic;
41     signal compteur_test: std_logic_vector(15 downto 0);
42
43 begin
44     clk <= not clk after 1 ns;
45     reset <= '0', '1' after 29 ns;
```

```
46      id <=    std_logic_vector(to_unsigned(0,4)),                    --
    tache 0
47               std_logic_vector(to_unsigned(1,4)) after 6 ns ,        --
    tache 1
48               std_logic_vector(to_unsigned(0,4)) after 11 ns;        --
    tache 0
49
50      wcet <= std_logic_vector(to_unsigned(3, 16)),                   --
    tache 0: 3
51               std_logic_vector(to_unsigned(5, 16)) after 7 ns;      --
    tache 1: 5
52
53      message <=  std_logic_vector(to_unsigned(0,4)),                 --
    stop
54                  std_logic_vector(to_unsigned(2,4)) after 2 ns,      --
    load
55                  std_logic_vector(to_unsigned(1,4)) after 3 ns,      --
    start
56                  std_logic_vector(to_unsigned(3,4)) after 5 ns,      --
    suspend
57                  std_logic_vector(to_unsigned(0,4)) after 7 ns,      --
    stop
58                  std_logic_vector(to_unsigned(2,4)) after 8 ns,      --
    load
59                  std_logic_vector(to_unsigned(1,4)) after 9 ns,      --
    start
60                  std_logic_vector(to_unsigned(3,4)) after 11 ns,     --
    suspend
61                  std_logic_vector(to_unsigned(4,4)) after 13 ns;     --
    resume
62
63      iut : entity work.taskMonitor(Behavioral)
64      Port map(
65           clk_in => clk,
66           reset_in => reset,
67           id_task => id,
68           wcet_task => wcet,
69           mess_task => message,
70           counter_interupt => compteur,
71           counter_interupt_test =>compteur_test
72      );
73
74  end Behavioral;
```

Listing 4 – TestBench moniteur de tâches