UMONS
Université de Mons

Faculté Polytechnique

Hardware and Software

Servomotor control by DE1-SoC : tutorial

Projet
Master Ingénieur civil en Electricité

Elora AMORISON
Clelia GALVANIN

POLYTECH MONS

Sous la direction du Professeur
Carlos VALDERRAMA

Mai 2019

ACADÉMIE UNIVERSITAIRE WALLONIE-BRUXELLES

Table des matières

Table des figures

Chapitre 1

Introduction

This project is part of the course Hardware/Software Platforms for students in first master in Electrical Engineering at the Faculty of Engineering of Mons.
The purpose of this document is to teach the reader, through a tutorial, how to control a servomotor with the DE1-SoC board.

The main objectives can be describe as the following :

— To handle an entire electronical project

— To familiar with processors by manipulating a De1-SoC board and all its possibilities (switches, buttons, ...)

— To control a servomotor : the reader will be able, thanks to the Intel Quartus and ModelSim tools, to rotate the mechanical arm of a servomotor and to be able to control its position. (i.e. the angle of rotation).

— To understand the VHDL programming language by implementing functions suiting at best the needs of the application.

— To create test benches to virtually test out the driver and the application.

Chapitre 2

Definition of the application

The goal of this project is to change the position of a servomotor manually with a DE1-SoC board. The switches on the board correspond to position values. The servomotor will change its position according to the switches that will be activated.

First, the servomotor will directly adjust its position in function of the switches. In a second step, it will be necessary to add a button which has to be pressed before the servomotor updates its position. This second solution allows a better handle of the control.

To change the position value, each switch is associated with a bit : 8 bits are necessary in order to reach an position angle of 180°.

For example : "00010000" send a angle position of 16° to the servomotor

Chapitre 3

Theoretical Reminders

3.1 Servomotor

The servomotor is a motor composed of gears and a mechanical arm. It is able to move its mechanical arm with high efficiency and great precision with an angle between 0 and 180° (or between -90 and 90°) in response to an external command. This angle interval is limited due to mechanical reasons.
A servomotor looks like the figure 3.1 :

FIGURE 3.1 – Servomotor

This type of motor is controlling by a PWM signal.

3.2 PWM Signal

The Pulse Width Modulation signal (PWM) is a type of signal that can express an analogical value from a binary signal.
The proportion of time during which the signal is at a higher voltage level is called *DutyCycle* and gives a value between 0 and 1 (Figure 3.2). Its expression is given by this following relation :

$$DutyCycle = \frac{PulseWidth}{Period}$$

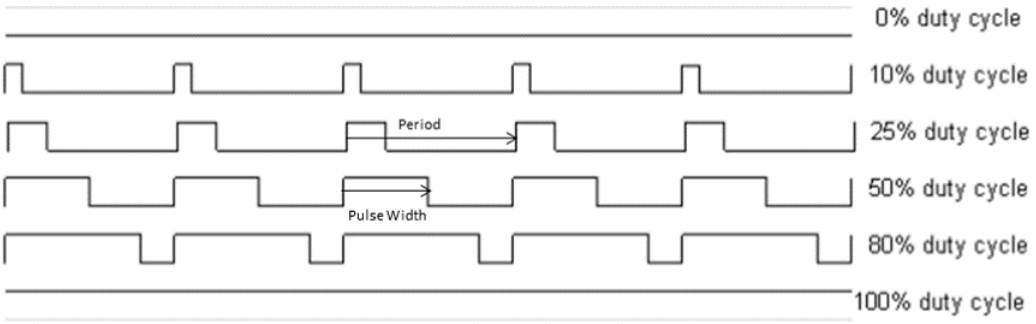

FIGURE 3.2 – DutyCycle

The length of the impulsion is limited by a minimal and maximal value.

In the case of the servomotor, the duration of the pulses is limited by minimum and maximum values. The period is $20ms$, the minimum duration is $500\mu s$ and the maximum duration is $2.5ms$.

3.3 Programming language : VHDL

VHDL is a hardware description language used in electronic design automation. VHDL comes from VHSIC (Very-High-Speed Integrated Circuits). It is used to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits.

It allows several design methodologies (behavioral, data flow, structural) while being of a very high level of abstraction in electronics. It is independent of the technology used so it allows to simulate the behaviour on computer before programming the processor.

This language has many advantages like :

— The behaviour of the required system can be described and verified before implementing into real hardware (gates and wires).

— It allows the description of a concurrent system.

— It has a full type system.

However, VHDL projects have also advantages. In fact, a VHDL project is multipurpose and portable.

Chapitre 4

Tools installation

4.1 Software

To make such an application as to control a servomotor using VHDL language, the reader will need two programs :

1. Quartus II
2. ModelSim

1. Intel Quartus Prime is a programmable logic device design software and is produced by Intel. It is called Altera Quartus II. This software includes tools to design FPGAs, SoCs and other programmable logic devices from the design to the synthesis and final simulations.

Quartus Prime includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.

FIGURE 4.1 – Quartus software

2. ModelSim is a multi-language HDL simulation environment for simulation of hardware description languages such as VHDL, Verilog and SystemC. ModelSim can be used independently, or in conjunction with Intel Quartus Prime (what we use in this case) or other softwares. Simulation is performed using the graphical user interface (GUI), or automatically using scripts.

To install the software, you'll need both the Quartus software tarfile, and the CycloneV qdz file. You have to save these both to the same directory. You can down-

load these programs on the site of Intel with the following link ; 'http ://fpgasoftware.intel.com/ ?edition=lite'.

To download ModelSim, select also the "ModelSim-Altera Starter Edition". ModelSim is a HDL Simulator and will be used to simulate signals before using the board.

4.2 Hardware

To implement the application, a DE1-Soc board will be used. The detailed board (pins, switches, buttons, ...) is shown at the figure 4.2

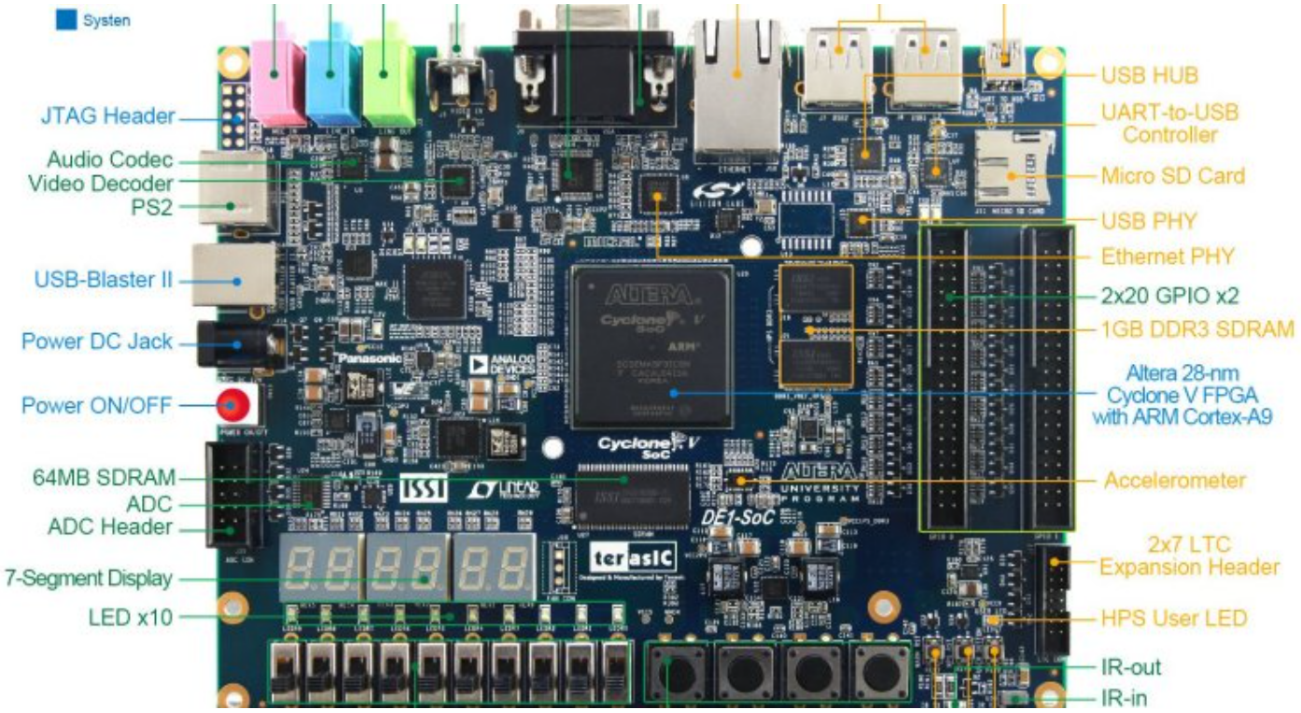

FIGURE 4.2 – DE1-Soc Board

To find this board, you can find some information on the following link to buy one ; 'https ://www.terasic.com.tw/cgi-bin/page/archive.pl ?Language=EnglishNo=836 '.

In the package, you may find the cable to supply the FPGA and an other one to connect to the computer and the software.

You may find more information on the user manual of the DE1-SoC board : "http ://www.ee.ic.ac.uk/pcheung/teaching/ee2$_{d}igital/DE1 - SoC_{U}ser_{m}anual.pdf$"

Chapitre 5

Software Development

5.1 Create a new project on Quartus II

The first thing to do is to create a new project on Quartus to create the program. The different steps are explained below.

1. Open Quartus II
2. Click on "create a new project"
3. A new window opens : (Figure 5.1)
   - You have to select the correct folder and write the name of your project.
   - Click on "Next"
   - Click on “empty project”

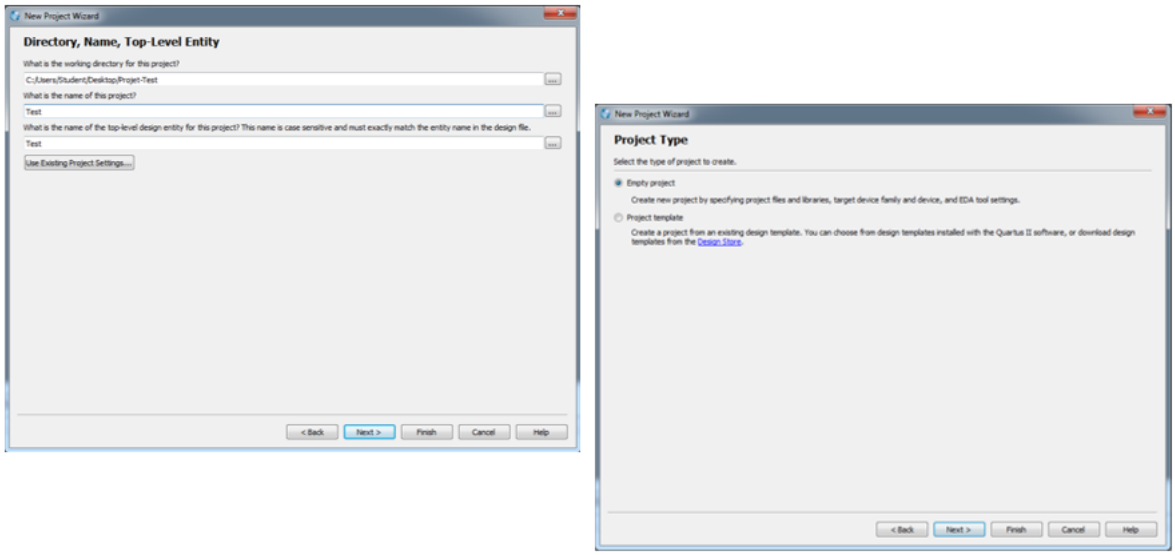

FIGURE 5.1 – Create a new project

4. A window opens called “Family and Device Settings”. Select the right family of your component and then “next”. (Figure 5.2)
   The component used in this project is a Cyclone V 5CSEMA5F31C6N.

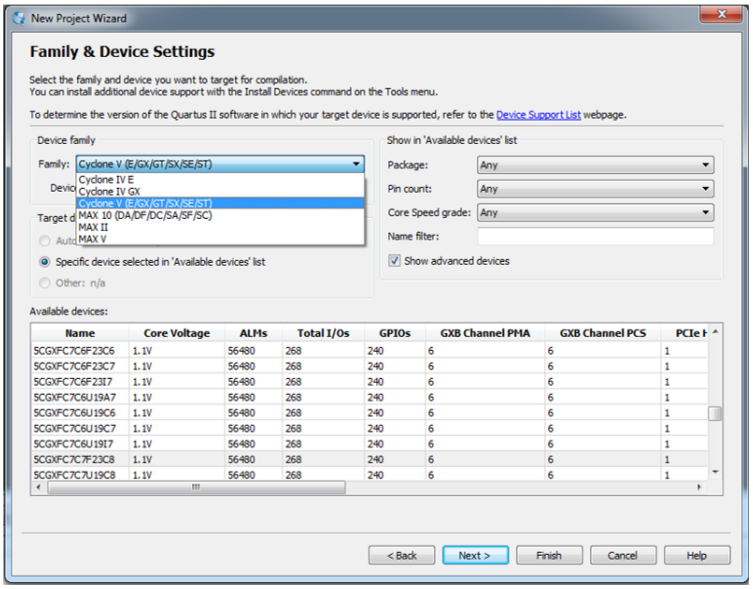

FIGURE 5.2 – Window “Family and Device Settings”

5. Now that the project is created, you have to add files to it. There are two choices : to add an already created file or to create a new file.
   To create a new file, select the VHDL Type File on the menu. (Figure 5.3)

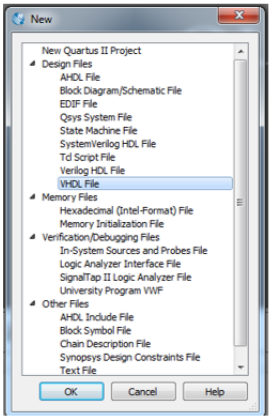

FIGURE 5.3 – Create a new VHDL file

6. You can add an existing file at your project. To do that, go to "New project Wizard" and add the name of the existing file you want to add. (Figure 5.4)

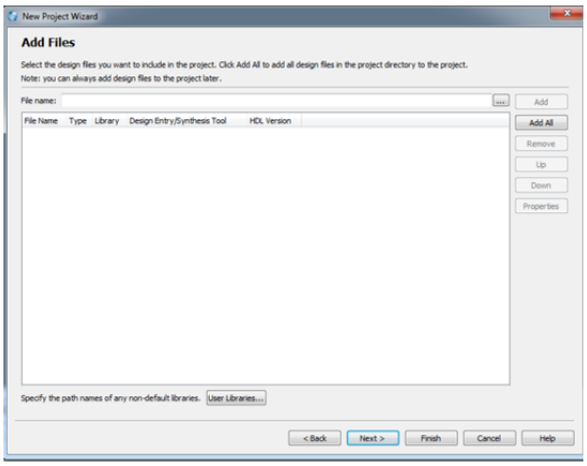

FIGURE 5.4 – Add an existing file

5.2 VHDL Code on Quartus II

5.2.1 Main codes

The program is divided in several files to be clearer :

1. **The clock at 64kHz.**(Figure A.1)
   The PWM control signal has two characteristics : its frequency (= 20ms) and its Pulse Width (between a minimal value of 0.5ms and a maximal value of 2.5ms). We want the servomotor to take 128 positions. So, we need a frequency divider of 64kHz.
2. **The PWM control for the servomotor.** (Figure A.2)
   There are three inputs : the clock, the reset, the position signal (called *pos*) and one output which is the servomotor control signal (called *servo*). With a 64kHz clock, we know that we will have 1ms each 64 iterations. But we want a frequency of 20ms so we multiply 20*64. We implement a counter from 0 to 1279 in the code and it is called *cnt*. The minimal value of the width pulse is 0.5ms so we have to add an offset of 32 to make it equivalent.
   The following code build a square wave with different width pulses. The block diagram at Figure 5.5 show the structure of the PWM control.
3. **The combination of the 64kHz clock and the servomotor.** (Figure A.3)
   The PORT MAP is used to stick together the 64kHz clock and the servomotor control component. This is what we call the *Driver* and it will be simulate with a test bench. So as the control the servomotor correctly. The schematic diagram of the application is shown below. (Figure 5.6)
4. **The implementation of the interface.**
   This file is used to link the IN and OUT signals to the pins, button and switches of the board.(Figure A.4)

FIGURE 5.5 – Block Diagram of the servomotor control

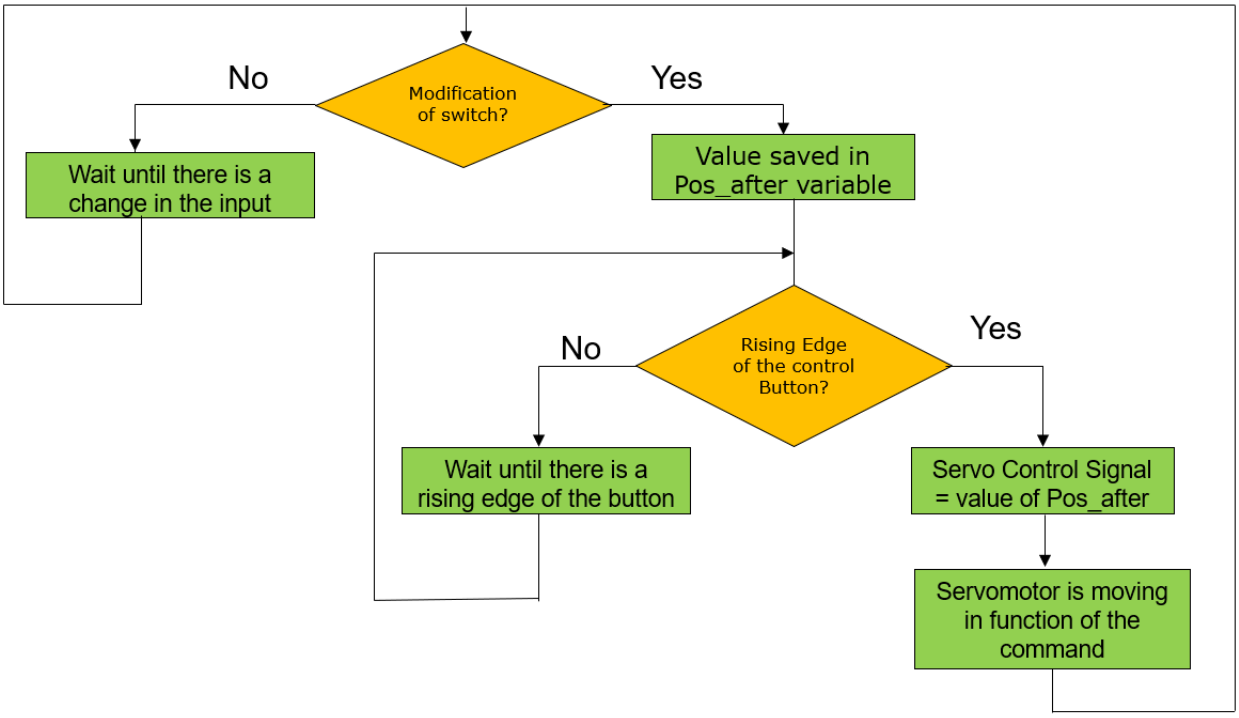

FIGURE 5.6 – Block Diagram of the Behaviour of our program

The names of the ports are listed and the user can take a look at the list. (Figure 5.7) This code defines the application via a Process. The position is controlled by 8 switches. It sends a binary value to the position signal which will send a angle position value to the servomotor. To clearly see the binary position value sent, there is a LED connected to each switch. So when a switch is on, the LED is also on.

Afterwards, the control signal is updated only when we go through the process. To go inside the process, we have to press a certain button. So the servomotor is not directly changing its position when we modify the switches. This file is made from the application and the driver. We will test it with a test bench.

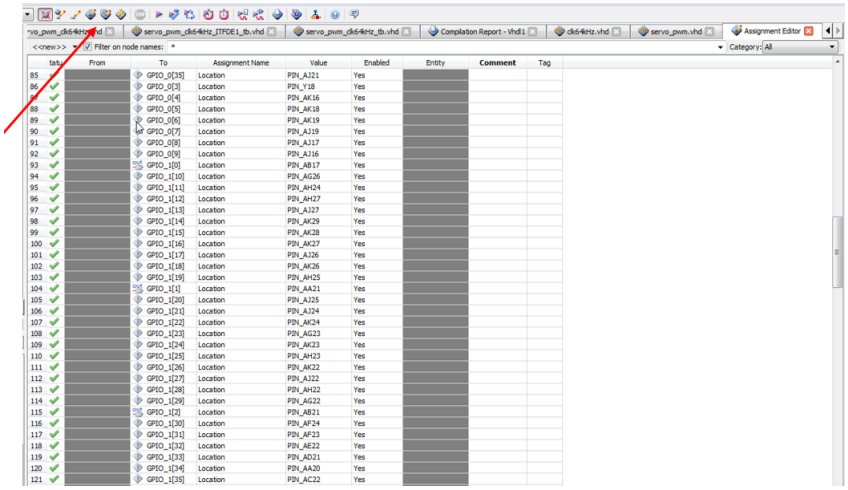

FIGURE 5.7 – Pins of the board

5.2.2 Test Benches

To test the program, test benches are made. Test benches are then compiled in ModelSim to visualise the signals and to verify that they correspond to what we expect. Two test benches are made : one for the driver (Code at figure B.1) and another for the driver and the application (Code at figure B.2 and B.3).

5.3 Simulations of Test Benches on ModelSim

In order to simulate the test benches, you have to go on the menu "Assignments". Select the concerned test bench file you made and change its type file in "VHDL Test Bench".

To launch ModelSim, compile your codes and then click on Run Simulation > RTL Viewer. ModelSim will open automatically. Once in ModelSim,

- Click on "Compile" and select the files you want to simulate and those related to it.
- Click on the concerned test bench file and right click on "Simulate".
- Drag and drop the signals you want to appear in the simulation window.
- Run the simulation.

Remark : Pay attention that the document related to the test bench is in the top position on Quartus II. To do this, you just have to press the right click on the concerned file in Quartus and select "On set top file".

5.3.1 Driver Simulation

For the simulation, we define arbitrarily some values for the variable «pos» which is the input signal corresponding to the position. The angle position signal will take first 0°, 40°, 80°and 120°.
For each position sent, we can observe the control signal modifying its pulse width which becomes bigger and bigger. (Figure 5.8)

For a position command sent of 80°, the pulse width value is 1749440 ns.(Figure 5.9) The result is what it is expected because the pulse width must be between 0.5ms and 2.5ms and when the angle position increases, the pulse width also increases. We also can note that the frequency of the PWM signal is correctly at 20ms and the minimal and maximal values are in the same range that the theorical values.

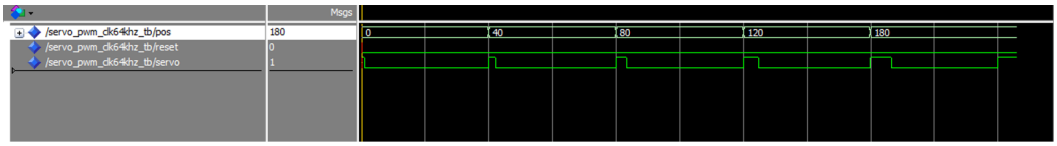

FIGURE 5.8 – Driver Simulation

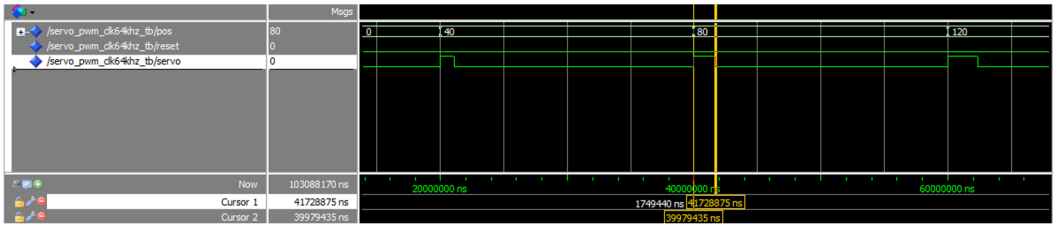

FIGURE 5.9 – Driver Simulation

5.3.2 Driver and Application Simulations

This simulation represents what actually happens in practice. There are two phases of operation to our project. First, we change the positions of the switches and store this value in a variable. Then, after pressing on the button, the value stored in the variable is sent to the servomotor and it changes its position. In the test bench, this is what we do :

1. We change the values of switches but nothing changes (which is normal because the value is stored in a variable but not sent to the output)
2. We wait 3 x PWM half period and we press the button.
3. We wait for 1 additional PWM half period before releasing the button.
4. When the button is released, the switches value go to Ipos-after.
5. PWM value is then modified as soon as the next impulse.

Remark : We have here left arbitrary times in order to correctly observe the phenomena of the process according to the steps of the process. In practice, the choice of the time to leave is free. We notice that the PWM value increases according to the angle.

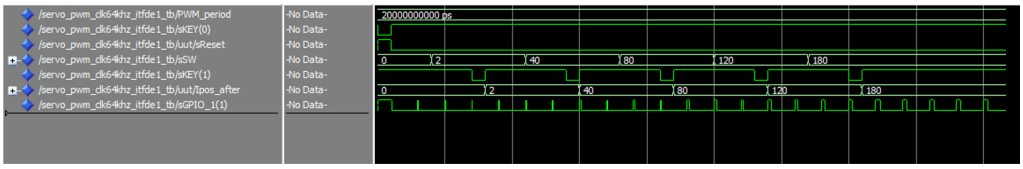

FIGURE 5.10 – Driver and Application Simulation

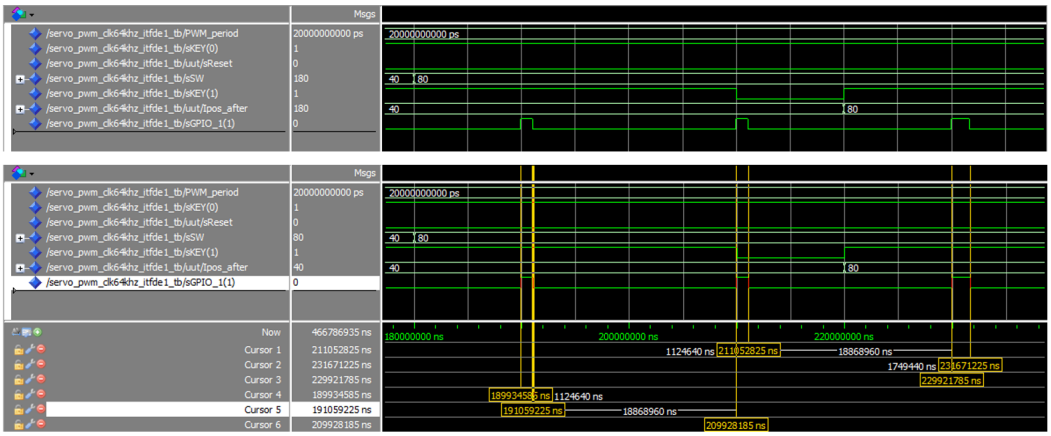

FIGURE 5.11 – Driver and Application Simulation

Chapitre 6

Hardware Development

6.1 How put the code on the processor ?

To test the application on the servomotor, the program has to be put on the DE1-SoC with the following steps :

— Connect the board to the computer with the USB cable provided for this purpose with the board ;
— Click on Program Device (Figure 6.1) ;

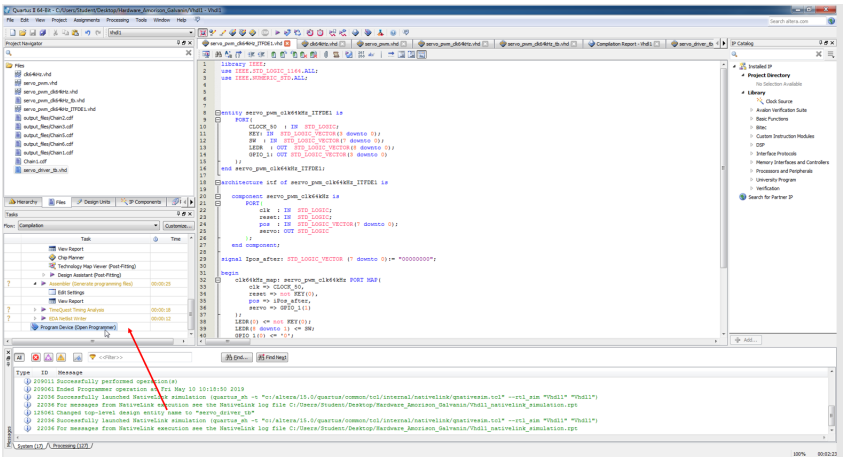

FIGURE 6.1 – Quartus II environment

— A new window will open, click on “Hardware Setup” and choose your hardware (Figure 6.2) ;

— Click on “Auto Detect” and choose the correct device (Figure 6.3) ;
— Then on “Add File” and select your own file ;
— Finally, click on “Start” and wait for the loading of your code.

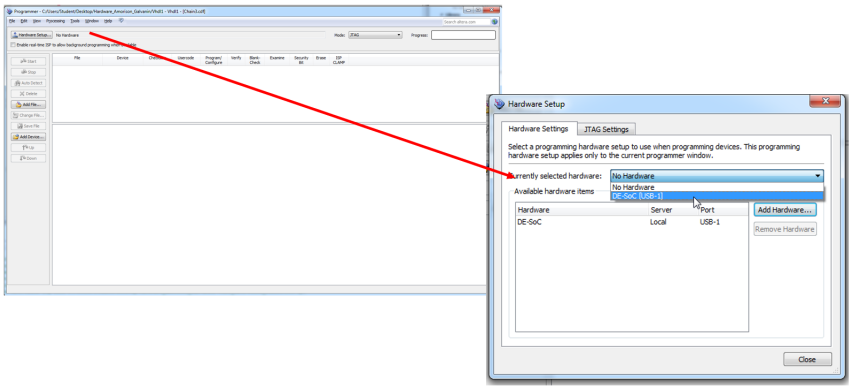

FIGURE 6.2 – Hardware Setup

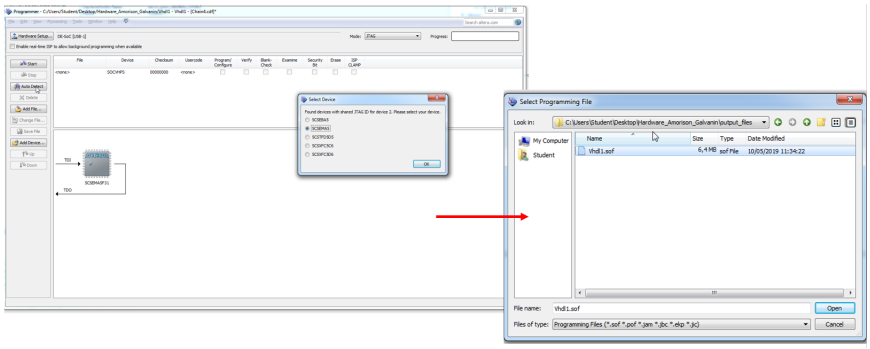

FIGURE 6.3 – Hardware selection

You can now use the board to verify that your code is doing what you want. You can check the behaviour of the program with two different ways : by connecting the board to an oscilloscope to observe the waveform of the signals or by connecting it to the servomotor and testing its changes of positions.

Chapitre 7

Final Results

7.1 Test on oscilloscope

To see the results on an oscilloscope, you have to connect the oscilloscope and the board. You can connect the cable of the oscilloscope with the pin corresponding to the output signal. Don't forget to connect the grounds with each other.
The signals waveform observed are normally similar to the simulations performed on ModelSim : Figure 7.1.

7.2 Test on servomotor

7.2.1 How to connect a servomotor to the board ?

A servomotor has three cables :

— The yellow wire must be connected to the output signal. The pin that send the output signal on the board is GPIO1(1).

— The red wire must be connected to the supply (here 5V) and it corresponds on the board to the GPIO1(10) pin.

— The brown (or black) wire must be connected to the ground and it is on the board the GPIO1(11) pin.

The ground and the 5V supply are given in the user manual of the processor.
The connections are shown in the Figure 7.2 and 7.3.

7.3 Results

You can watch our video to see the results with the following link :
"https ://www.youtube.com/watch ?v=HVyE35rkSaQt=5s"

We hope this tutorial has helped you. Let's control servomotor !

FIGURE 7.1 – PWM waveform

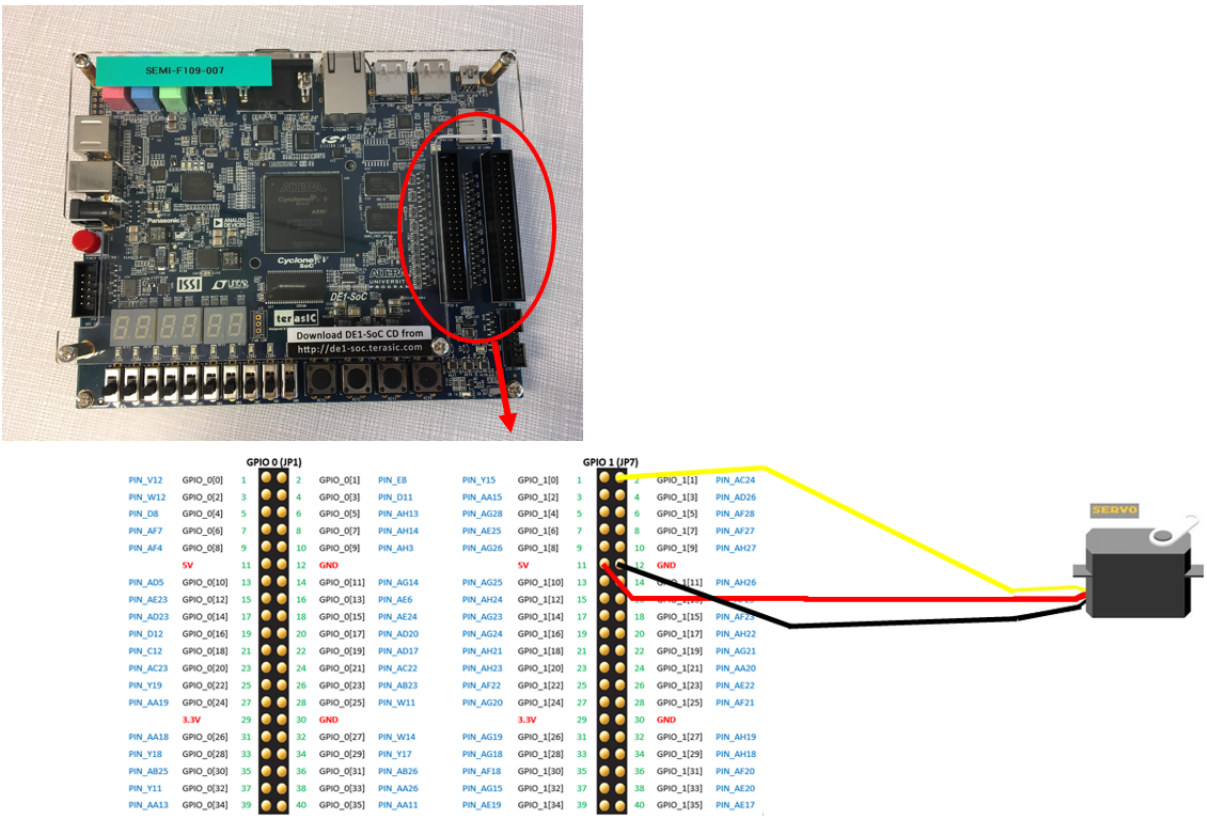

FIGURE 7.2 – Connections between the board and the servomotor

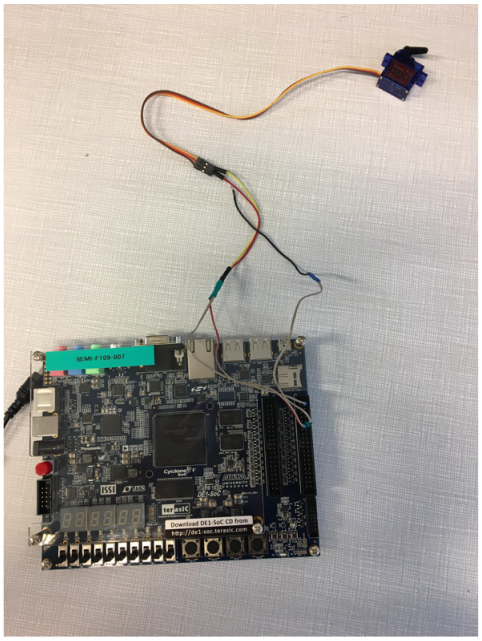

FIGURE 7.3 – Connections between the board and the servomotor

Annexe A

Main Codes

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity clk64kHz is
    Port (
        clk    : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end clk64kHz;

architecture Behavioral of clk64kHz is
    signal temporal: STD_LOGIC;
    signal counter : integer range 0 to 780 := 0;
begin
    freq_divider: process (reset, clk) begin
        if (reset = '1') then
            temporal <= '0';
            counter  <= 0;
        elsif rising_edge(clk) then
            if (counter = 780) then
                temporal <= NOT(temporal);
                counter  <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    clk_out <= temporal;
end Behavioral;
```

FIGURE A.1 – Clk64kHz Code

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity servo_pwm is
    PORT (
        clk   : IN  STD_LOGIC;
        reset : IN  STD_LOGIC;
        pos   : IN  STD_LOGIC_VECTOR(7 downto 0);
                  servo : OUT STD_LOGIC
    );
end servo_pwm;

architecture Behavioral of servo_pwm is
    -- Counter, from 0 to 1279.
    signal cnt : unsigned(10 downto 0);
    -- Temporal signal used to generate the PWM pulse.
    signal pwmi: unsigned(8 downto 0);
begin
    -- Minimum value should be 0.5ms.
    pwmi <= unsigned('0' & pos) + 32;
    -- Counter process, from 0 to 1279.
    counter: process (reset, clk) begin
        if (reset = '1') then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            if (cnt = 1279) then
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;
    -- Output signal for the servomotor.
    servo <= '1' when (cnt < pwmi) else '0';
end Behavioral;
```

FIGURE A.2 – Servomotor Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5
6
7
8  entity servo_pwm_clk64kHz is
9      PORT(
10         clk  : IN  STD_LOGIC;
11         reset: IN  STD_LOGIC;
12         pos  : IN  STD_LOGIC_VECTOR(7 downto 0);
13         servo: OUT STD_LOGIC
14     );
15 end servo_pwm_clk64kHz;
16
17 architecture Behavioral of servo_pwm_clk64kHz is
18     COMPONENT clk64kHz
19         PORT(
20             clk    : in  STD_LOGIC;
21             reset  : in  STD_LOGIC;
22             clk_out: out STD_LOGIC
23         );
24     END COMPONENT;
25
26     COMPONENT servo_pwm
27         PORT (
28             clk   : IN  STD_LOGIC;
29             reset : IN  STD_LOGIC;
30             pos   : IN  STD_LOGIC_VECTOR(7 downto 0);
31             servo : OUT STD_LOGIC
32         );
33     END COMPONENT;
34
35     signal clk_out : STD_LOGIC := '0';
36
37
38 begin
39     clk64kHz_map: clk64kHz PORT MAP(
40         clk => clk,
41                 reset => reset,
42                 clk_out => clk_out
43     );
44
45     servo_pwm_map: servo_pwm PORT MAP(
46         clk => clk_out,
47                 reset => reset,
48                 pos => pos,
49                 servo => servo
50     );
51 end Behavioral;
```

FIGURE A.3 – Servomotor + Clk64kHz Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5
6
7
8  entity servo_pwm_clk64kHz_ITFDE1 is
9      PORT(
10         CLOCK_50  : IN  STD_LOGIC;
11         KEY: IN  STD_LOGIC_VECTOR(3 downto 0);
12         SW  : IN  STD_LOGIC_VECTOR(7 downto 0);
13         LEDR  : OUT  STD_LOGIC_VECTOR(8 downto 0);
14         GPIO_1: OUT STD_LOGIC_VECTOR(3 downto 0)
15     );
16 end servo_pwm_clk64kHz_ITFDE1;
17
18 architecture itf of servo_pwm_clk64kHz_ITFDE1 is
19
20         component servo_pwm_clk64kHz is
21                 PORT(
22                     clk  : IN  STD_LOGIC;
23                          reset: IN  STD_LOGIC;
24                          pos  : IN  STD_LOGIC_VECTOR(7 downto 0);
25                          servo: OUT STD_LOGIC
26                 );
27         end component;
28
29 signal Ipos_after: STD_LOGIC_VECTOR (7 downto 0):= "00000000";
30 signal sReset: STD_LOGIC;
31 begin
32          sReset <= not KEY(0);
33     clk64kHz_map: servo_pwm_clk64kHz PORT MAP(
34         clk => CLOCK_50,
35                   reset => sReset,
36                   pos => iPos_after,
37                   servo => GPIO_1(1)
38     );
39     LEDR(0) <= not KEY(0);
40          LEDR(8 downto 1) <= SW;
41          GPIO_1(0) <= '0';
42          GPIO_1(2) <= '1';
43
44          bouton : process(KEY(1)) begin
45                         if rising_edge(KEY(1)) then
46                                 iPos_after <= SW;
47                         end if;
48         end process;
49
50
51 end;
```

FIGURE A.4 – Interface Code

Annexe B

Test Benches Codes

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--Test Bench Driver


ENTITY servo_pwm_clk64kHz_tb IS
END servo_pwm_clk64kHz_tb;

ARCHITECTURE behavior OF servo_pwm_clk64kHz_tb IS
    -- Unit under test.
    COMPONENT servo_pwm_clk64kHz
        PORT(
            clk   : IN  std_logic;
            reset : IN  std_logic;
            pos   : IN  std_logic_vector(7 downto 0);
            servo : OUT std_logic
        );
    END COMPONENT;

    -- Inputs.
    signal clk  : std_logic := '0';
    signal reset: std_logic := '0';
    signal pos  : std_logic_vector(7 downto 0) := (others => '0');
    -- Outputs.
    signal servo : std_logic;
    -- Clock definition.
    constant clk_period : time := 10 ns;
BEGIN
    -- Instance of the unit under test.
    uut: servo_pwm_clk64kHz PORT MAP (
        clk => clk,
        reset => reset,
        pos => pos,
        servo => servo
    );

   -- Definition of the clock process.
   clk_process :process begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;

    -- Stimuli process.
    stimuli: process begin
        reset <= '1';
        wait for 50 ns;
        reset <= '0';
        wait for 50 ns;
        pos <= "00000000";
        --wait for 20 ms;
                wait until servo='1';
        pos <= "00101000";
        wait until servo='1';
        pos <= "01010000";
        wait until servo='1';
        pos <= "01111000";
        wait until servo='1';
        pos <= "10110100";
        wait;
    end process;
END;
```

FIGURE B.1 – Driver Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--Test Bench Application


ENTITY servo_pwm_clk64kHz_ITFDE1_tb IS
END servo_pwm_clk64kHz_ITFDE1_tb;


architecture Behavioral of servo_pwm_clk64kHz_ITFDE1_tb is

    COMPONENT servo_pwm_clk64kHz_ITFDE1

                        PORT(
                                CLOCK_50  : IN  STD_LOGIC;
                                KEY: IN  STD_LOGIC_VECTOR(3 downto 0);
                                SW  : IN  STD_LOGIC_VECTOR(7 downto 0);
                                LEDR  : OUT  STD_LOGIC_VECTOR(8 downto 0);
                                GPIO_1: OUT STD_LOGIC_VECTOR(3 downto 0)
    );
         END COMPONENT;


        signal sCLOCK_50  : std_logic := '0';
        signal sKEY : STD_LOGIC_VECTOR(3 downto 0) := "1111";
        signal sSW  : std_logic_vector(7 downto 0) := (others => '0');
    -- Outputs.
    signal sGPIO_1 : STD_LOGIC_VECTOR(3 downto 0) := "0000";
         signal sLEDR  : std_logic_vector(8 downto 0) := (others => '0');
    -- Clock definition.
    constant clk_period : time := 10 ns; --A tester : mettre une clock de 12 µs
         constant PWM_period : time := 20 ms;

BEGIN

             -- Instance of the unit under test.
    uut: servo_pwm_clk64kHz_ITFDE1 PORT MAP (
        ClOCK_50 => sClOCK_50,
        KEY => sKEY,
        SW => sSW,
        LEDR => sLEDR,
                  GPIO_1 => sGPIO_1
    );

   -- Definition of the clock process.
   clk_process :process begin
        sClOCK_50 <= '0';
        wait for clk_period/2;
        sClOCK_50 <= '1';
        wait for clk_period/2;
   end process;
```

FIGURE B.2 – Driver and Application Test Bench

```
53
54
55
56          stimuli: process begin
57          sKEY(0) <= '0'; --Reset à 1 comme not KEY dans le code
58          wait for PWM_period/2; --Attendre un temps plus long et plus cohérent avec notre PWM_period
59          sKEY(0) <= '1'; --Reset à 0 idem
60          wait for 3*PWM_period/2; --
61
62                    sSW <= "00000010";    -- ON change la position des pos mais le PWM ne devrait pas changer
63                    wait for 3*PWM_period/2; -- ON observe que le PWM ne change pas
64
65                    sKEY(1) <= '0';       -- On appuie sur le bouton
66                    wait for PWM_period/2;
67                    sKEY(1) <= '1'; -- On lache le bouton -> ipos after change
68                    wait for 3*PWM_period/2; -- On attend, à la prochaine PWM_period on voit le PWM changer
69
70          sSW <= "00101000"; --40
71                    wait for 3*PWM_period/2;
72
73                    sKEY(1) <= '0';
74                    wait for PWM_period/2;
75                    sKEY(1) <= '1';
76                    wait for 3*PWM_period/2;
77
78          sSW <= "01010000"; --80
79                    wait for 3*PWM_period/2;
80
81                    sKEY(1) <= '0';
82                    wait for PWM_period/2;
83                    sKEY(1) <= '1';
84                    wait for 3*PWM_period/2;
85
86          sSW <= "01111000";      --120
87                    wait for 3*PWM_period/2;
88
89                    sKEY(1) <= '0';
90                    wait for PWM_period/2;
91                    sKEY(1) <= '1';
92                    wait for 3*PWM_period/2;
93
94          sSW <= "10110100";--180
95                    wait for 3*PWM_period/2;
96
97                    sKEY(1) <= '0';
98                    wait for PWM_period/2;
99                    sKEY(1) <= '1';
100                   wait for 3*PWM_period/2;
101         wait;
102     end process;
103
104
105 END;
```

FIGURE B.3 – Driver and Application Test Bench (2)