Embedded Systems
Processor Architecture Laboratory

DE0-Nano-SoC Lab2 - Design of a specific Programmable Interface for FPGA

Teacher :
Beuchat René

Student :
Lefebure Nicolas

17 novembre 2019

Table des matières

1 Introduction

The main purpose of this laboratory was to understand the design of a custom slave programmable interface in VHDL language on a DE0-Nano-SoC board. In particular, we implemented an Avalon slave peripheral that is capable of outputting a PWM signal. Our interface was able to modify the following specificities of the PWM :

— Period
— Duty Cycle
— Polarity

The figure 1 shows the different programmable parameters of the PWM. Period : the duration of time of one cycle. Duty cycle : percentage of time signal remains on/off during the period of the PWM signal. Polarity : logic state to define if the signal is on or off during the duty cycle.

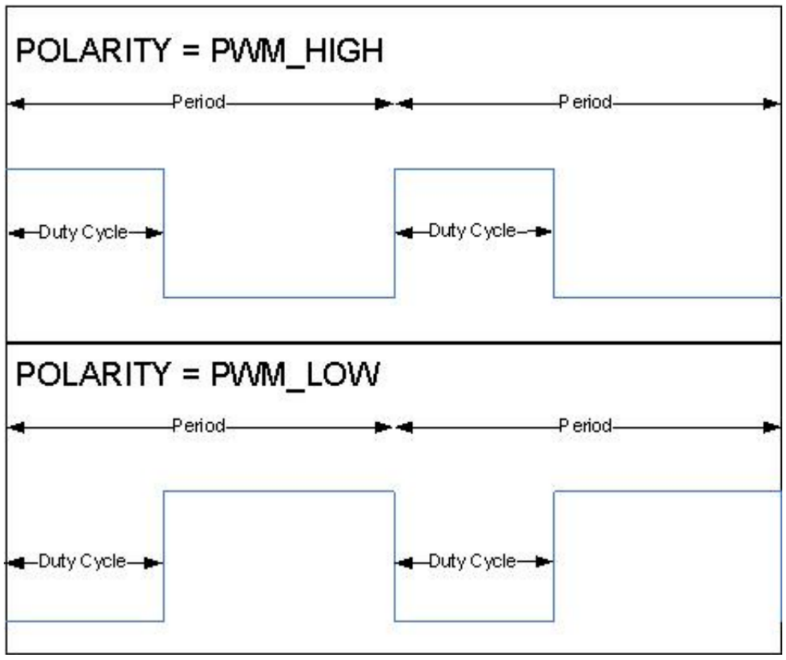

FIGURE 1: PWM specifications

The section 2 of the report explains the hardware design of the project, in particular coding of the parallel port and the PWM controller along with the NIOS II complete design. The section 3 describes the NIOS II software design.

2 Hardware Design

2.1 Quartus Prime Project

The specific parallel programmable interface is made of two parts : parallel port and PWM controller. The parallel port is used to write internal registers through Avalon bus interface and read the different sources of data by the Avalon bus interface. The PWM controller is used to generate the PWM signal.

2.1.1 Parallel port

The objective was to design a interface for an Avalon bus as a slave module. The main characteristics of the module are :

— Modification of PWM value port
— Enable/disable of the PWM
— Programmable Period, Duty cycle and Polarity

The PWM port output 'PWMa' is made of 1 bit, it can take '0' and '1' values. There is a synchronous write and read access in rising_edge of the clock with a separate bus. For your design on the Avalon bus, 1 wait-state is needed. The Avalon bus provide signals to the module :

— clk : Clock
— Reset_n : Initialization
— Adr : Address vector of 2 bits
— CS : Selection of this module
— Rd : Read access
— Wr : Write access
— WRData : Data vector of 32 bits send to the module in write access
— RDData : Data vector of 32 bits to provide by the module in read access

Then the parallel port is provided with register signals to manipulate :

— RegPeriod
— RegNewDuty
— RegPol
— PreClkEn

The figure 2 shows the parallel port. The table 1 shows the register map of the interface. The code for the parallel port is given below.

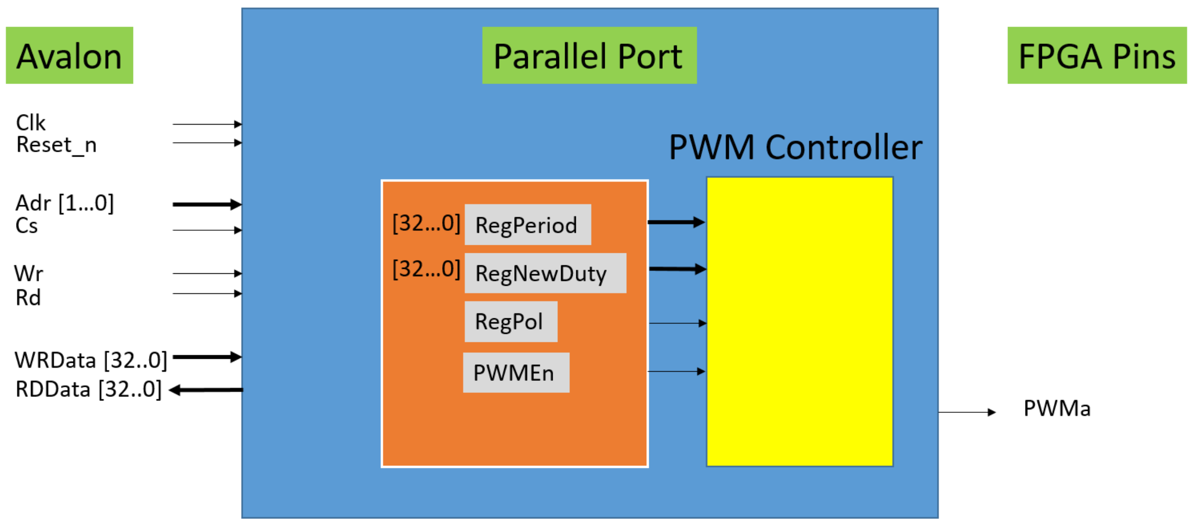

FIGURE 2: Parallel Port Module on Avalon

Table 1: Register Map

Register	Address	Read\Write	size (bits)	Reset value	Description
RegPeriod	0	R\W	32	0	Defines PWM Period
RegNewDuty	1	R\W	32	0	Defines PWM Duty cycle
PWMEn	2	R\W	1	0	Enables\Disables PWM
RegPol	3	R\W	1	[-]	Defines Polarity of PWM

2.1.2 PWM Controller

The control of the PWM is made by two processes : Counting of the clock and signal generation. The first process counts until it achieves the value of the Period. The second one assigns PWM value (0 or 1) in function of the duty cycle and the polarity. The code for the programmable interface and the top-level controller are provided at the end of the report.

2.2 Qsys System

We created a full system including a processor and memory capable of executing software. The parallel port was supported by a custom component while the rest of the software was manage by local components.

2.2.1 Custom Component

The figure 3 shows the block symbol of the custom component.

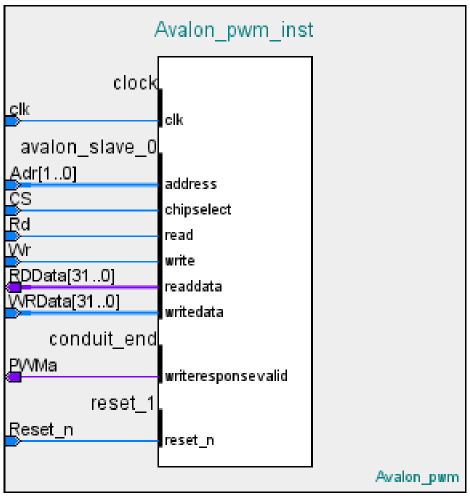

Figure 3: Block symbol of the custom component

2.2.2 Embedded Component

The rest of the components was found with the IP catalog :

— Nios II Processor
— On-Chip Memory RAM
— JTAG UART

The address map of all the components is given by the table 2.

TABLE 2: Adress Map

Component	Address	Description
Nios II Processor	0x0004_0800	Processor for operations
On-Chip Memory RAM	0x0002_0000	128kb Memory
JTAG UART	0x0004_1010	Receiver-Transmitter
Avalon PWM	0x0004_1000	Parallel port

3 Software Design

The goal of the project was to generate PWM for a certain period, duty cycle and polarity. We show three different demonstrations of PWM :

— Period = 20 ms, Duty cycle = 10ms, Polarity = 1
— Period = 15 ms, Duty cycle = 5ms, Polarity = 1
— Period = 15 ms, Duty cycle = 5ms, Polarity = 0

The figures 4 to 6 shows the PWM output of the logic analyser for the three different demonstrations.

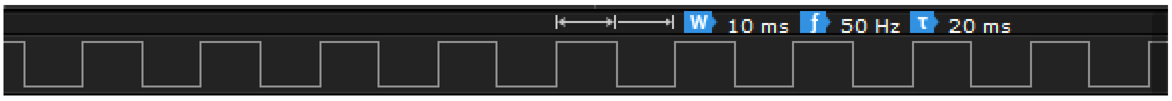

FIGURE 4: Demonstration 1

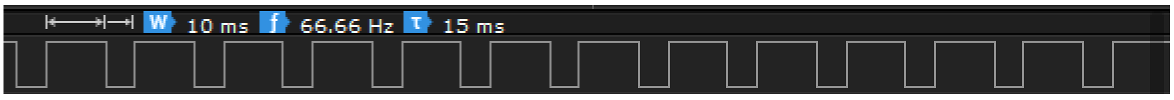

FIGURE 5: Demonstration 2

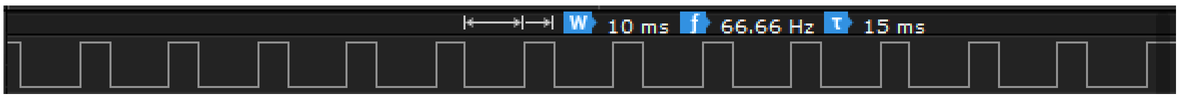

FIGURE 6: Demonstration 3

The code for the software is given at the end of the report.

C code

```c
#include <inttypes.h>
#include "system.h"
#include "stdio.h"
#include "io.h"

#define Period_1 1000000
#define Period_2 750000
#define DutyCycle_1 500000
#define DutyCycle_2 250000
#define PWMEnable 1
#define Polarity_1 1
#define Polarity_2 0
#define Demo 3

#define Period_bit 0
#define DutyCycle_bit 1
#define PWMEnable_bit 2
#define Polarity_bit 3

void PWMinit() {

  if (Demo == 1) {
    IOWR_32DIRECT(AVALON_PWM_0_BASE, Period_bit*4, Period_1);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, DutyCycle_bit*4, DutyCycle_1);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, PWMEnable_bit*4, PWMEnable);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, Polarity_bit*4, Polarity_1);
  }
  else if(Demo == 2) {
    IOWR_32DIRECT(AVALON_PWM_0_BASE, Period_bit*4, Period_2);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, DutyCycle_bit*4, DutyCycle_2);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, PWMEnable_bit*4, PWMEnable);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, Polarity_bit*4, Polarity_1);
  }
  else if(Demo == 3)  {
    IOWR_32DIRECT(AVALON_PWM_0_BASE, Period_bit*4, Period_2);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, DutyCycle_bit*4, DutyCycle_2);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, PWMEnable_bit*4, PWMEnable);
    IOWR_32DIRECT(AVALON_PWM_0_BASE, Polarity_bit*4, Polarity_2);
  }
}
int main(void) {

  printf("Hello from Nios II!\n");
  PWMinit();

  while(1)

  return 0;
}
```

VHDL Code : Parallel Port

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Lab22 is
port
        (
                clk     :  IN STD_LOGIC;
                Reset_n :  IN STD_LOGIC;
                Adr     :  IN STD_LOGIC_VECTOR(1 downto 0);
                CS      :  IN STD_LOGIC;
                Rd      :  IN STD_LOGIC;
                Wr      :  IN STD_LOGIC;
                WRData  :  IN STD_LOGIC_VECTOR(31 downto 0);
                RDData  :  OUT STD_LOGIC_VECTOR(31 downto 0);
                PWMa    :  OUT STD_LOGIC
        );
end Lab22;


architecture comp of Lab22 is

signal RegPeriod    :  unsigned(31 downto 0);      --Reg. Periode of PWM
signal RegNewDuty   :  unsigned(31 downto 0);      --Reg. Duty cylce
signal RegPol       :  std_logic;                  --Reg. Polarity
signal CntPWM       :  unsigned(31 downto 0);      --Counter for PWM
signal PWMEn        :  std_logic;                  --PWM enable Begin

Begin


-----------------------------------------------
-----------------------------------------------
---------------- Parallel port ----------------
-----------------------------------------------
-----------------------------------------------

-- Process to write internal registers through Avalon bus interface
-- Synchronous access in rising_edge of clk
-- Addresses allows to select write registers if CS and Wr activated

WrReg: process(clk, Reset_n)                                                                  -- Write by
        begin
                if Reset_n = '0' then
                        RegPeriod <= (others => '0');
                        RegNewDuty <= (others => '0');
                        PWMEn <= '0';
                elsif rising_edge(clk) then
                        if (CS = '1') and (Wr = '1') then
                        case Adr is
                                when "00" =>
                                        RegPeriod <= unsigned(WRData);
                                when "01" =>
                                        RegNewDuty <= unsigned(WRData);
                                when "10" =>
                                        PWMEn <= WRData(0);
                                when "11" =>
                                        RegPol <= WRData(0);
                                when others =>
                                        null;
                                end case;
                        end if;
                end if;
        end process WrReg;

-- Process to read the different sources of data by the Avalon bus interface
RdReg: process(Cs, Rd, Adr, RegPeriod, RegNewDuty, PWMEn, RegPol)
        begin
                if rising_edge(clk) then
                        RDData <= (others => '0');
                        if (CS = '1') and (Rd = '1') then
                                case Adr is
                                        when "00" =>
                                                RDData <= std_logic_vector(RegPeriod);
                                        when "01" =>
                                                RDData <= std_logic_vector(RegNewDuty);
                                        when "10" =>
                                                RDData(0) <= PWMEn;
                                        when "11" =>
                                                RDData(0) <= RegPol;
                                        when others =>
                                                RDData <= (others => '0');
                                end case;
                        end if;
                end if;
        end process RdReg;


-----------------------------------------------
-----------------------------------------------
--------------- PWM Controller ----------------
-----------------------------------------------
-----------------------------------------------

-- Process to generate PWM

-- Process to count until Period value
CountingPWM: process(clk, Reset_n)
begin
        if Reset_n = '0' then
                CntPWM <= (others => '0');
        elsif rising_edge(clk) then
                if (CntPWM < (RegPeriod - 1)) then
                        CntPWM <= CntPWM + 1;
                else
                        CntPWM <= (others => '0');
                end if;
        end if;
end process CountingPWM;
```

```
-- Process to assign PWM value and polarity
Pulse: process(clk, Reset_n)
begin
        if Reset_n = '0' then
                PWMa <= '0';
        elsif rising_edge(clk) then
                if (PWMEn = '1') then
                        if (RegPol = '0') then
                                if (RegNewDuty > CntPWM) then
                                        PWMa <= '1';
                                else
                                        PWMa <= '0';
                                end if;
                        elsif (RegPol = '1') then
                                if (RegNewDuty > CntPWM) then
                                        PWMa <= '0';
                                else
                                        PWMa <= '1';
                                end if;
                        end if;
                end if;
        end if;
end process Pulse;


End comp;
```

VHDL Code : Top-level Controller

```
-- #############################################################################
-- DE0_Nano_SoC_top_level.vhd
--
-- BOARD         : DE0-Nano-SoC from Terasic
-- Author        : Sahand Kashani-Akhavan from Terasic documentation
-- Revision      : 1.1
-- Creation date : 11/06/2015
--
-- Syntax Rule : GROUP_NAME_N[bit]
--
-- GROUP : specify a particular interface (ex: SDR_)
-- NAME  : signal name (ex: CONFIG, D, ...)
-- bit   : signal index
-- _N    : to specify an active-low signal
-- #############################################################################

library ieee;
use ieee.std_logic_1164.all;

entity DE0_Nano_SoC_top_level is
    port(

        FPGA_CLK1_50     : in    std_logic;

        -- KEY
        KEY_N            : in    std_logic_vector(1 downto 0);

        -- LED
        LED              : out   std_logic_vector(7 downto 0);


        -- GPIO_0
        GPIO_0           : inout std_logic_vector(35 downto 0)

    );
end entity DE0_Nano_SoC_top_level;

architecture rtl of DE0_Nano_SoC_top_level is
                component system is
                port (
                        avalon_pwm_0_conduit_end_writeresponsevalid : out std_logic;
                        clk_clk                                     : in  std_logic := 'X';
                        reset_reset_n                               : in  std_logic := 'X'
                );
        end component system;

begin
u0 : component system
                port map (
                        avalon_pwm_0_conduit_end_writeresponsevalid => GPIO_0(0), -- avalon_pwm_0_c
                        clk_clk                                     => FPGA_CLK1_50,
                        reset_reset_n                               => KEY_N(0)
                );
```

```
end;
```