Final Project EELE 475

Dr.Ross Snider

Controlling a PWM Signal from Input

Kameron Kranse

Kasy Treu

Abdulaziz Alsaleh

December 10, 2015

1. Introduction

This project is able to determine the value of the pulse width modulation control word based on the seconds in the time coming from a stream of GPS data. the component we made takes the GPS stream, which is parsed by the C code, and is able to manipulate it in the component to output a value between -128 and 128 to set the pulse width modulation. This project was to prove that another component could take an input from a stream of data, and use it to manipulate the PWM of the NIOS board. The value of -128 to 128 was used so that this could easily be used to connect to a servo, and be used for other applications.

2. Overview

The major component of this design project was the custom component of the GPS parser that we added. This component did the calculations for transforming the seconds from the GPS signal to a usable value "word" that we could set to the PWM.

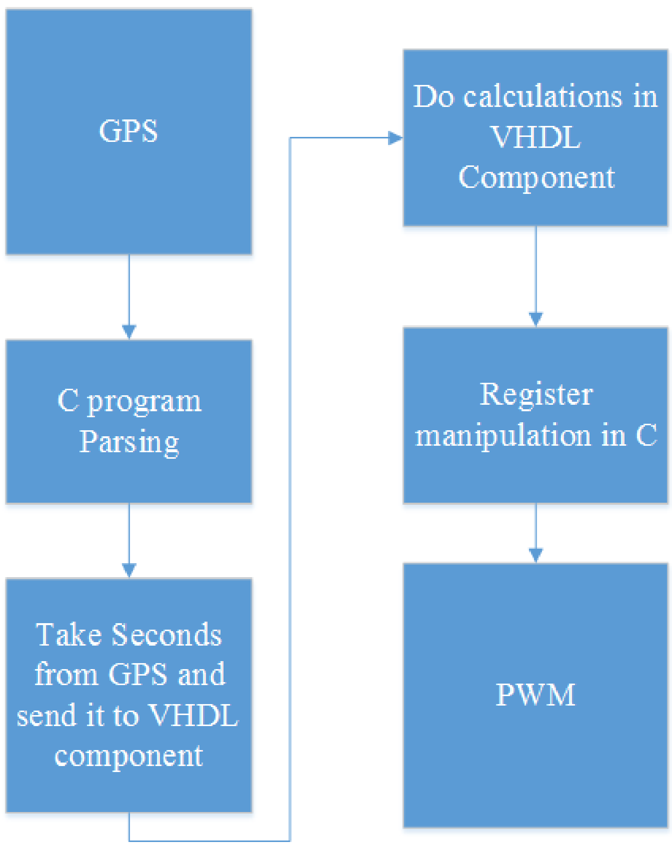

Figure 1: Block Diagram of the Project Overview

3. Hardware/Software Interface

a. Below is a Block diagram that graphically shows the memory offset of our blocks and registers.

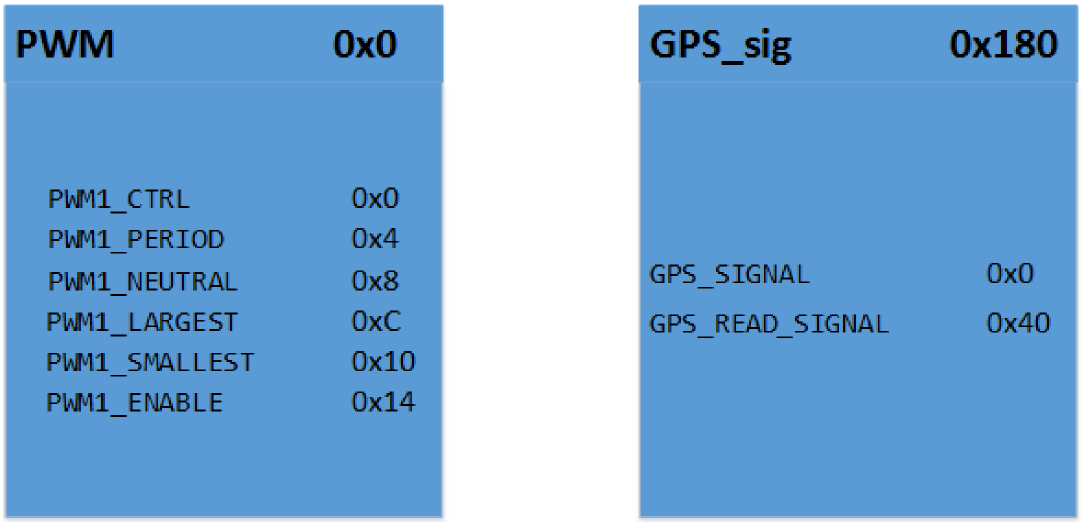

Figure 2: Register and Memory Offset

b. Below is a table that lists the registers and the information about them.

Block Description Table

Register Name **Register Type**	**Block_Offset (in bytes)**	**Register_Offset (in bytes)**	**What happens on a**	
			Write?	**Read?**
Control Registers				
PWM_CTRL std_logic_vector	0x0	0x0	Controls the PWM	Returns current value
PWM_ENABLE std_logic	0x0	0x20	Turns on PWM	Returns current value
GPS_SIGNAL std_logic_vector	0x180	0x00	Write a value for calculations to be performed on	NA

GPS_READ_SIGNAL std_logic_vector	0x180	0x40	NA	Returns calculation
Initialization Registers				
PWM_PERIOD std_logic_vector	0x0	0x4	Sets the period of PWM in clock cycles	Returns current value
PWM_NEUTRAL std_logic_vector	0x0	0x8	Sets the middle PWM width in clock cycles	Returns current value
PWM_LARGEST std_logic_vector	0x0	0xC	Sets the largest PWM width in clock cycles	Returns current value
PWM SMALLEST std_logic_vector	0x0	0x10	Sets the smallest PWM width in clock cycles	Returns current value

Figure 3: Block Description Table

c. Below is the bit mappings and descriptions for the registers used in our project.

PWM_CTRL

	MSB PWM_CTRL (Block_Offset = 0x0, Register_Offset=0x0) LSB							
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 4: Register Description for PWM_CTRL

This register controls the PWM pulse width.

PWM_ENABLE

Register Description

	MSB PWM_ENABLE (Block_Offset = 0x0, Register_Offset=0x14) LSB

Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 5: Register Description for PWM_ENABLE

This register enables the PWM.

GPS_SIGNAL

Register Description

	MSB	GPS_SIGNAL (Block_Offset = 0x180, Register_Offset=0x0)						LSB
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 6: Register Description for GPS_SIGNAL

This register holds GPS Time value from the software

GPS_READ_SIGNAL

Register Description

	MSB	GPS_READ_SIGNAL (Block_Offset = 0x180, Register_Offset=0x40)					LSB	
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 7: Register Description for GPS_SIGNAL

This register takes in the calculated control word from hardware, and is read from the C code.

PWM_PERIOD

Register Description

	MSB	PWM_PERIOD (Block_Offset = 0x0, Register_Offset=0x4)					LSB	
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 8: Register Description for PWM_PERIOD

This register sets the period of the PWM (in clock cycles).

PWM_NEUTRAL

Register Description

	MSB	PWM_NEUTRAL (Block_Offset = 0x0, Register_Offset=0x8)						LSB
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 9: Register Description for PWM_NEUTRAL

This register sets the middle width of the PWM (in clock cycles).

PWM_LARGEST

Register Description

	MSB	PWM_LARGEST (Block_Offset = 0x0, Register_Offset=0xC)						LSB
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 10: Register Description for PWM_LARGEST

This register sets the largest width of the PWM (in clock cycles).

PWM_SMALLEST

Register Description

	MSB	PWM_SMALLEST (Block_Offset = 0x0, Register_Offset=0x10)						LSB
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 11: Register Description for PWM_SMALLEST

This register sets the smallest width of the PWM (in clock cycles).

d. A description of what your software does with each of these registers.

Before the code runs, it sets the register values for the minimum and maximum pulse width to -128 and 127, respectively. Then, the C code reads the data stream from the GPS data, and parses it out to find the seconds of the current time. It then sends the current seconds variable back into the register of the GPS_Sig. There, the component manipulates the data and sends it to a read only register, GPS_READ_Sig. The C code then reads out from that register with the manipulated value and set to the PWM_CTRL register to operate the PWM at a different pulse width based upon the seconds of the GPS.

4. Conclusion

~~**Make a conclusion and relate your experience regarding the creation and execution of your project**~~

The project initially was challenging as we needed to understand how to create multiple components that were able to communicate with each other through registers/shared information. We could create the component we needed easily, but it proved difficult to get our components to interact with each other correctly. After many failed attempts at getting information to flow between our components and its software, we correctly wired our QSYS system and it started to function the way we wanted it to. In this lab, we essentially combined the last two exercises we have in class, by combining the GPS parser, the PWM module, and the concept of shadow registers into this project. Getting information to flow freely back and forth between our C code and hardware was the most difficult part of this process. Improper implementation resulted in an inability to connect our components together, which resulted in hours of debugging for our VHDL. When we finally connected the our hardware up correctly, the next largest problem we encountered was the math manipulation in our hardware. We overcame this obstacle by performing shifts in the registers to emulate the mathematical functions we needed. After a few more hours of brainstorming and debugging, we finally resulted with a functional project that did exactly what we wanted it to do.

Appendix

C CODE:

```
/*
 * pwm.c
 *
 *  Created on: Nov 3, 2015
 *      Author: Kasy Treu, Kameron Kranse, Aziz
 */
#define PWM1_BASE_ADDRESS    0x01001180
#define GPS_SIGNAL         (int *) 0x01001000
#define GPS_READ_SIGNAL     (int *) 0x01001040
#define PWM1_CTRL          ((volatile int*) PWM1_BASE_ADDRESS)
#define PWM1_PERIOD        ((volatile int*) (PWM1_BASE_ADDRESS + 4))
#define PWM1_NEUTRAL       ((volatile int*)(PWM1_BASE_ADDRESS + 8))
#define PWM1_LARGEST       ((volatile int*)(PWM1_BASE_ADDRESS + 12))
#define PWM1_SMALLEST      ((volatile int*)(PWM1_BASE_ADDRESS + 16))
#define PWM1_ENABLE        ((volatile int*) (PWM1_BASE_ADDRESS + 20))

#define Switches          (volatile int *) 0x010011c0
#define LEDs               (int *) 0x01001250

#include<stdio.h>
#include <stdlib.h>

int main() {

	*PWM1_NEUTRAL = 0x000124f8;
	*PWM1_LARGEST = 0x000186a0;
	*PWM1_SMALLEST = 0x0000c350;
	*PWM1_ENABLE = 1;
	*PWM1_PERIOD = 0x0f4240;

	char c;
	char dataSet[80];
	while (1) {
			c = getchar();
			if (c == '$') {
				c = getchar();
				if (c == 'G') {
		dataSet[0] = c; //if the first character in the line after the $ is a G, store to array
					c = getchar();
					if (c == 'P') {
	dataSet[1] = c; //if the first character in the line after G is P, store to next array spot
```

```
                    c = getchar();
                    if (c == 'G') {
dataSet[2] = c; //if the first character in the line after P is G, store to next array spot
                        c = getchar();
                        if (c == 'G') {
dataSet[3] = c; //if the first character in the line after G is G, store to next array spot
                            c = getchar();
                            if (c == 'A') {
dataSet[4] = c; //if the first character in the line after G is A, store to next array spot

                                int i = 5;
                                while (c != '*') {
                        c = getchar(); // put all values in a char[] to XOR
                        dataSet[i] = c;
                        i++;
                      }


                    char myarray[2] = { dataSet[10], dataSet[11] };
                    int time;


            sscanf(myarray, "%d", &time);

                // Send Seconds to GPS signal
                *GPS_SIGNAL = time;
                *PWM1_CTRL = *GPS_READ_SIGNAL;

                *LEDs = time;
                 printf(" %d\n",  time);
                 printf(" %d\n",  *GPS_READ_SIGNAL);
                            }
                        }
                    }
                }
            }
        }
}

return 0;
}
```

VHDL for GPS_Sig:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity GPS_sig is
	port (
		clk				: in std_logic;
		reset_n			: in std_logic;
		gps_signal		: in std_logic_vector(31 downto 0);
		avs_s1_write		: in std_logic;
		avs_s1_read				: in std_logic;
		avs_s1_address		: in std_logic_vector(4 downto 0);
		avs_s1_writedata	: in std_logic_vector(31 downto 0);
		avs_s1_readdata	: out std_logic_vector(31 downto 0)
	);
end GPS_sig;


architecture behavior of GPS_sig is

	signal readdata			: STD_LOGIC_VECTOR (31 downto 0);
	signal wre : std_logic;  -- write enable
	signal re : std_logic;   -- read  enable
	signal addr : std_logic_vector(4 downto 0);  -- register address
	signal gps_signal2		: std_logic_vector(31 downto 0);
	signal temp  : std_logic_vector(31 downto 0);
	signal temp2 : std_logic_vector(31 downto 0);
	signal gps_output :std_logic_vector(31 downto 0);

begin
	wre  <= avs_s1_write;
	re   <= avs_s1_read;
	addr <= avs_s1_address;

  process (clk)
	begin
		if rising_edge(clk) and wre='1' and addr="00000" then
			gps_output <=  avs_s1_writedata;
		end if;
	end process;

	process (clk)
	begin
		if rising_edge(clk) and re='1' then
```

```
                    case addr is
                        when "00000" | "10000" =>
                            readdata <= gps_signal2;
                        when others => readdata <= x"00000000";
                    end case;
                end if;
            end process;

process(clk)
 begin
 if gps_output < "000000000000000000000000111101" then
            if gps_output < "00000000000000000000000000100000" then
                    temp2 <= (gps_output(29 downto 0) & "00"); --multiply by 4

                    temp <= "1111111111111111111111111" & temp2(6 downto 0);
            elsif gps_output < "00000000000000000000000000110000" then
                    temp2 <= (gps_output(29 downto 0) & "00"); --multiply by 4
                    temp <= "00000000000000000000000000" & temp2(5 downto 0);
            else
                    temp2 <= (gps_output(29 downto 0) & "00"); --multiply by 4
                    temp <= "0000000000000000000000000" & temp2(6 downto 0);
            end if;
                    gps_signal2 <= temp;
 end if;
end process;

            avs_s1_readdata <= readdata;


            end architecture;
```