To: Prof. Ross Snider

From: Kameron Kranse, Kasy Treu, and Abdulaziz Alsaleh

Regarding: Lab #8 – PWM LAB

Date Performed: November 24, 2015

Adding Custom Hardware to the NIOS II Soft Processor and Memory Mapping it on the Avalon Interconnect Fabric

***Summary**:*

This lab experiment goal was to use the Pulse Width Modulation (PWM) component that was created in VHDL, add it to Qsys IDE, create three PWM controllers on the Avalon bus and output them onto the oscilloscope within the programming IDE. Also show that shadow registers were properly implemented through using them.

***Introduction**:*

A PWM module was created and three of these modules were added to the system. These modules would create a single pulse of a desired width repeating at a desired interval. The output of the system was read through an oscilloscope to determine if the output was correct. Also shadow registers were added, which were used to mimic given registers in the system. These would allow us to read information in some registers without accessing those specific registers themselves.

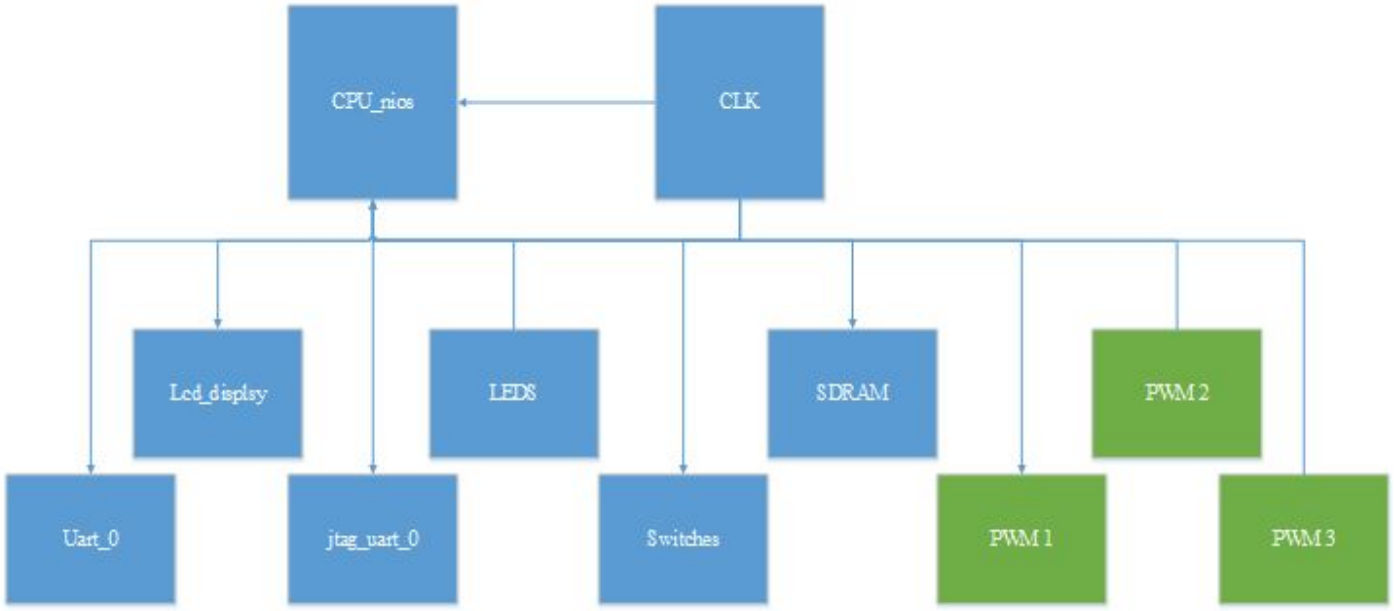

Figure 1: Block Diagram of System

***Body**:*

The first step we did was create an avalon wrapper in order to build a bus for communication with a peripheral device. Next we created a PWM (pulse width modulation) module. The VHDL in order to create these modules was provided. These modules would output a pulse of a given width and period. The avalon wrapper needed to write to registers that will control the PWM module. These registers' offsets were determined based on the clock speed that was 50 Mhz in the system that we were using. Six registers were used to hold control words for the PWM module. Upon a write these control words may not have been a full 32 bits long so they were right shifted having the upper end of the 32 bits ignored. However, on a read the trailing bits on the beginning needed to be changed to either 0 or 1 depending on the situation. These registers were originally read only, we altered those registers to be read and write capable,

and then created a second pair of “shadow” registers that would copy the value of the original register and store it there, these were read only registers in the event that writing to the original registers ended with an undesirable result. The register values could be read from both locations.

Concluding Section:

In conclusion, even though only one PWM was created and used, the switches for the PWM worked properly and designated the correct timing for the modulation. Also, when reading and writing to registers the value that was written to a register could be read back from both the original register and it’s 64 bit offset shadow register.

Appendix

```
/*
 * pwm.c
 *
 *  Created on: Nov 3, 2015
 *      Author: Kasy Treu, Kameron Kranse, Abdulaziz Alsaleh
 */

#define PWM1_BASE_ADDRESS    0x01001100
#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 PWM1_READ_BASE_ADDRESS    0x01001140
#define PWM1_READ_CTRL          ((volatile int*) PWM1_READ_BASE_ADDRESS)
#define PWM1_READ_PERIOD        ((volatile int*) (PWM1_READ_BASE_ADDRESS + 4))
#define PWM1_READ_NEUTRAL       ((volatile int*)(PWM1_READ_BASE_ADDRESS +
8))
#define PWM1_READ_LARGEST       ((volatile int*)(PWM1_READ_BASE_ADDRESS +
12))
#define PWM1_READ_SMALLEST      ((volatile int*)(PWM1_READ_BASE_ADDRESS +
16))
#define PWM1_READ_ENABLE        ((volatile int*) (PWM1_READ_BASE_ADDRESS +
20))

#define Switches          (volatile int *) 0x010011c0
#define LEDs            (int *) 0x010011d0

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

int main() {

        *PWM1_NEUTRAL = 0x000124f8;
        *PWM1_LARGEST = 0x000186a0;
```

```
        *PWM1_SMALLEST = 0x0000c350;
//      *PWM1_ENABLE = 1;
        *PWM1_PERIOD = 0x0f4240;

        printf("Write to CTRL: -5\n");
        *PWM1_CTRL = -5;
        printf("CTRL: %d\n", *PWM1_CTRL);
        printf("CTRL READ: %d\n", *PWM1_READ_CTRL);

        printf("Write to PERIOD: 1\n");
        *PWM1_PERIOD = 1;
        printf("PERIOD: %d\n", *PWM1_PERIOD);
        printf("PERIOD READ: %d\n", *PWM1_READ_PERIOD);

        printf("Write to NEUTRAL: 2\n");
        *PWM1_NEUTRAL = 2;
        printf("NEUTRAL: %d\n", *PWM1_NEUTRAL);
        printf("NEUTRAL READ: %d\n", *PWM1_READ_NEUTRAL);

        printf("Write to LARGEST: 3\n");
        *PWM1_LARGEST = 3;
        printf("LARGEST: %d\n", *PWM1_LARGEST);
        printf("LARGEST READ: %d\n", *PWM1_READ_LARGEST);

        printf("Write to SMALLEST: 4\n");
        *PWM1_SMALLEST = 4;
        printf("SMALLEST: %d\n", *PWM1_SMALLEST);
        printf("SMALLEST READ: %d\n", *PWM1_READ_SMALLEST);

        printf("Write to ENABLE: 1\n");
        *PWM1_ENABLE = 5;
        printf("ENABLE: %d\n", *PWM1_ENABLE);
        printf("ENABLE READ: %d\n", *PWM1_READ_ENABLE);

//      while (1) {
//              switch (*Switches) {
//
//              case 1:
//                      *PWM1_ENABLE = 0x01;
```

```
//                      *PWM1_CTRL = -128;
//                      *LEDs = 1;
//                      break;
//
//              case 2:
//                      *PWM1_ENABLE = 0x01;
//                      *PWM1_CTRL = 0;
//                      *LEDs = 2;
//                      break;
//
//              case 4:
//                      *PWM1_ENABLE = 0x01;
//                      *PWM1_CTRL = 127;
//                      *LEDs = 4;
//                      break;
//              default:
//                      *PWM1_ENABLE = 0x00;
//                      *LEDs = 0;
//                      break;
//              }
//
//
//      }

        return 0;
}
```