ECEN 5863 - Programmable Logic Embedded System Design *Ruthvik R Chanda*
Project 1- Final Report *Shrinithi Venkatesan*

Executive Summary

Module 1- DEVELOP A MIXED-SIGNAL SYSTEM

Building on the PWM FPGA project, completed the design by using the ADC to read the PWM generated voltage and reported the results on the 7-segment LEDs.

Module 2- DISPLAY HELLO WORLD AND TOGGLE LEDS USING NIOS II AND Q SYS

Implemented a soft-core processor on programmable logic. Designed a system using Qsys (Intel FPGA Platform Designer) and its IP catalog, then imported into Quartus Prime. After compiling the FPGA project, successfully programmed the FPGA to function as a Nios II processor. Finally, developed a basic "Hello, World!" application that ran on the processor within the FPGA-based system.

Module 3- CREATE A SYSTEM ON A CHIP WITH PROGRAMMABLE HARDWARE AND A SOFT PROCESSOR

Implemented an entire SoC module by adding multiple peripherals to the Qsys system. As the lab progresses, leant how quick and easy it is to build entire systems using Altera's Qsys to configure and integrate pre-verified IP blocks.

Objectives

Module 1- DEVELOP A MIXED-SIGNAL SYSTEM

The goal of this hardware lab is to develop a working design, using most aspects of the Quartus Prime Design Flow, including Qsys.

Module 2- DISPLAY HELLO WORLD AND TOGGLE LEDS USING NIOS II AND Q SYS

The goal of this lab is to create a simple hello world application using MAX10 device as programmable hardware and Nios II soft processor.

Module 3- CREATE A SYSTEM ON A CHIP WITH PROGRAMMABLE HARDWARE AND A SOFT PROCESSOR

The goal of this Embedded Systems lab is to develop a working design, using most aspects of the Quartus Prime Design Flow, including Qsys and the NIOS II Embedded Design Suite (EDS).

Procedure

Module 1- DEVELOP A MIXED-SIGNAL SYSTEM

PWM:

We created a Quartus project to generate PWM signals using three switches for duty cycle control. We incorporated a PLL for clock generation and developed Verilog models for debouncing and PWM generation, simulating the design.

ADC:

We added an ADC module in Qsys, employing a PLL for clocking. We interfaced JTAG for debugging and established communication with other design elements using a memory-mapped bridge. ADC output was displayed on six 7-segment displays.

In **Section 1** we will prepare for the project acquiring files and other resources.
In **Section 2** we will examine the system design.
In **Section 3** we will learn how to Create a system design using Quartus Prime and Qsys.
In Section 4 we will place and route the design.
In Section 5 we will create a programming file, program the FPGA, and then test the hardware design.
`

Module 2- DISPLAY HELLO WORLD AND TOGGLE LEDS USING NIOS II AND Q SYS

Module 2 involved designing a hardware system for the DE10-Lite Development Kit using Qsys in Quartus. It included components like NIOS II Processor, memory, UART, switches, LEDs, and push-buttons. The FPGA design mapped signals to these peripherals, and the software implementation, using NIOS II Software Build Tools in Eclipse, displayed "Hello World" and allowed user-specific input based on push-button presses.

Module 3- CREATE A SYSTEM ON A CHIP WITH PROGRAMMABLE HARDWARE AND A SOFT PROCESSOR

Module 3 involved building a System-on-Chip (SoC) using the MAX10 DE10-Lite development kit with Nios 2 as the soft processor. The module consisted of two parts: In Qsys, various components like PLLs, NIOS 2 processor, RAM, Flash memory, clock bridges, LEDs, switches, timers, SDRAM controller, SPI for accelerometer, ADC module, JTAG interfaces, and system ID were added. Base addresses and interrupt priorities were assigned to each component. For software, a Board Support Package (BSP) was generated from Qsys's .sopcinfo file. A project was created in Eclipse to utilize the JTAG UART for a binary counter that displays output on LEDR[7..0] based on user input. Additionally, modifications were made to display accelerometer values on the console and LEDR[7..0] pins.

For module 2 and 3:

1. Configured the pin settings in Quartus for the Device Kit.
2. Utilized Qsys to construct a system based on the Nios II processor.
3. Integrated the Qsys component into your top-level design.
4. Established connections between push buttons and LEDs.
5. Compiled the hardware design.
6. Imported the Nios II-based system into Eclipse Software Build Tools.
7. Created a software project.
8. Customized a software template to enable basic IO operations.
9. Compiled the software.
10. Loaded the hardware image onto the DE10-LITE Development Kit.
11. Transffered the software executable to the DE10-LITE Development Kit.
12. Downloaded the software executable to the DE10-LITE Development Kit.
13. Verified the hardware's functionality through testing.

Module Test Results

Module 1- DEVELOP A MIXED-SIGNAL SYSTEM

PWM

Record the fmax.

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	562.11 MHz	450.05 MHz	b2v_inst|altpll_compone...o_generated|pll1|clk[1]	limit due to minimum period restriction (tmin)
2	615.01 MHz	450.05 MHz	b2v_inst|altpll_compone...o_generated|pll1|clk[0]	limit due to minimum period restriction (tmin)

`

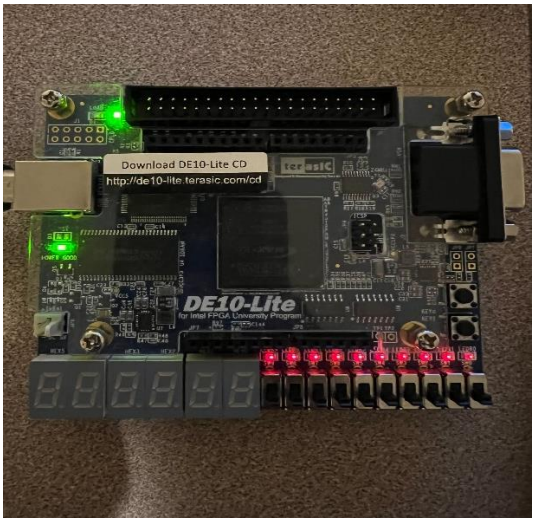
Estimate the % utilization of the FPGA logic.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Mon Oct 02 22:06:40 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	pwm_led_top
Top-level Entity Name	pwm_led_top
Family	MAX 10
Device	10M50DAF484C6GES
Timing Models	Preliminary
Total logic elements	33 / 49,760 (< 1 %)
Total registers	31
Total pins	6 / 360 (2 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	1 / 4 (25 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

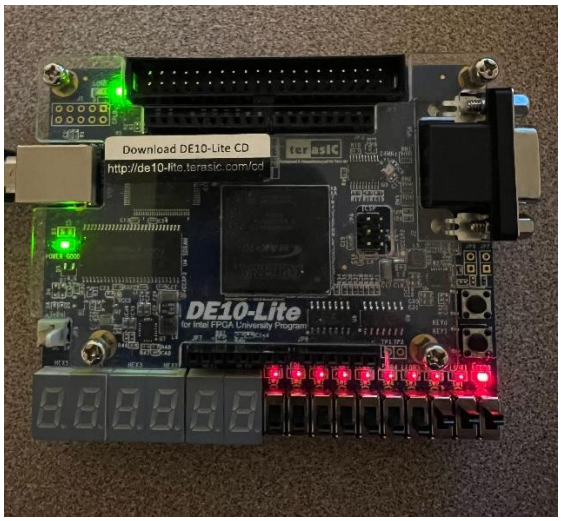
Record your observations of the board's behavior once the FPGA is programmed. Does it behave as you expected?

Ans: LEDs intensity varies using the switch inputs. When SW0,SW1,SW2 are turned on one by one, the intensity of the LED is increased. Everything is behaving as expected.

Board Output: Pic1 shows the default LED and Switches. Pic2 shows that the first 3 switches are turned on, the intensity of the first led is high.



Pic1



Pic2

ADC

Record the fmax.

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	50.51 MHz	50.51 MHz	ADC_CLK_10	
2	84.56 MHz	84.56 MHz	altera_reserved_tck	
3	470.59 MHz	450.05 MHz	inst|altpll_component|auto_generated|pll1|clk[1]	limit due to minimum period restriction (tmin)
4	568.18 MHz	450.05 MHz	inst|altpll_component|auto_generated|pll1|clk[0]	limit due to minimum period restriction (tmin)

`

Estimate the % utilization of the FPGA logic.

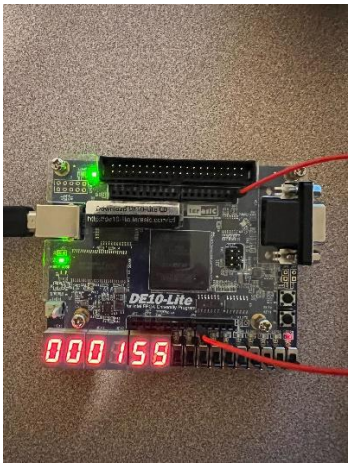
Flow Summary	
<<Filter>>	
Flow Status	Successful - Mon Oct 02 22:36:24 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	pwm_led_top
Top-level Entity Name	pwm_led_top
Family	MAX 10
Device	10M50DAF484C6GES
Timing Models	Preliminary
Total logic elements	4,743 / 49,760 (10 %)
Total registers	3233
Total pins	50 / 360 (14 %)
Total virtual pins	0
Total memory bits	270,176 / 1,677,312 (16 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	2 / 4 (50 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	1 / 2 (50 %)

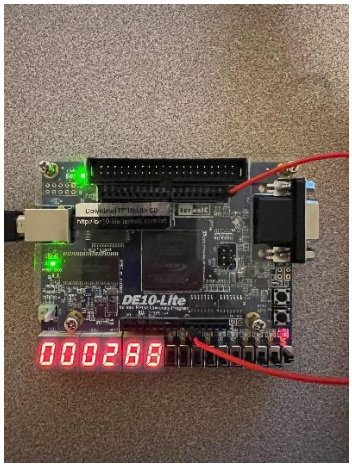
S

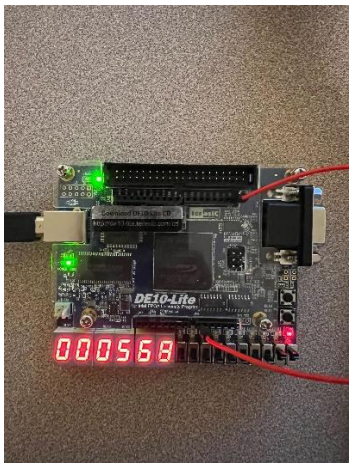
Record your observations of the board's behavior once the FPGA is programmed. Does it behave as you expected? Is this a good voltmeter? What could you change in either the board hardware or FPGA logic to make it perform better?

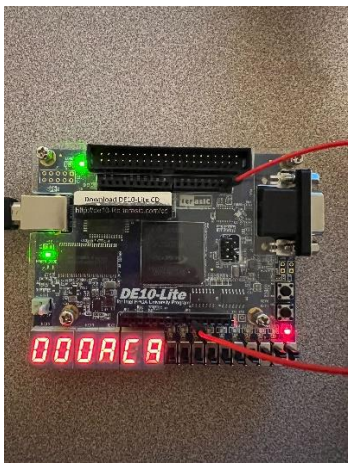
Ans: LEDs intensity varies using the switch inputs and prints the ADC values on the 7segment display. When SW0,SW1,SW2 are turned on one by one, the intensity of the LED is increased with increasing ADC values. Everything is behaving as expected. This is an average voltmeter. As it only shows an ADC value and not the voltage values on the 7-segment display. Changing the FPGA logic by converting the ADC values to voltage would make this perform better.

Board Output:









`

Module 2- DISPLAY HELLO WORLD AND TOGGLE LEDS USING NIOS II AND Q SYS

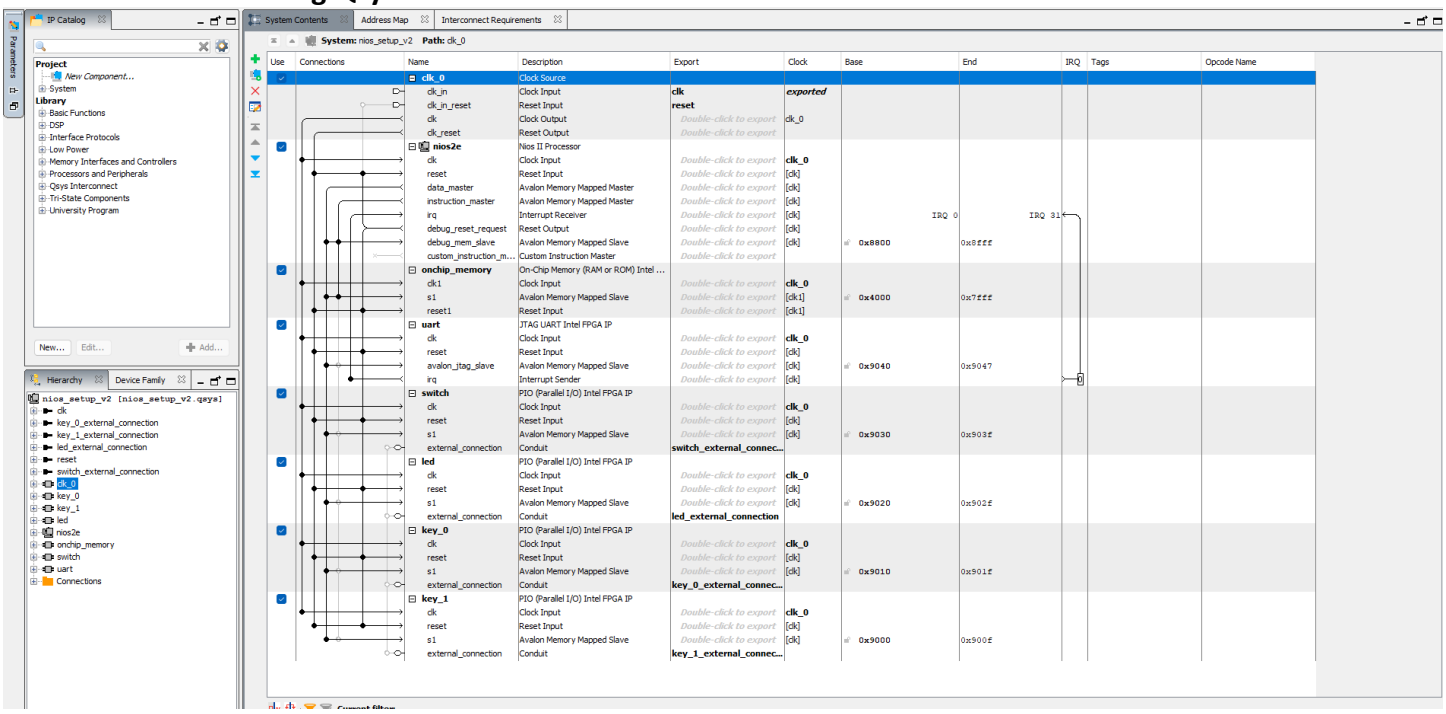
Record the fmax

	Fmax	Restricted Fmax	Clock Name
1	115.85 MHz	115.85 MHz	altera_reserved_tck

Estimate the % utilization of the FPGA logic.

Flow Status	Successful - Fri Sep 29 18:49:40 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	hello_world_lab
Top-level Entity Name	hello_world
Family	MAX 10
Device	10M50DAF484C6GES
Timing Models	Preliminary
Total logic elements	1,643 / 49,760 (3 %)
Total registers	892
Total pins	7 / 360 (2 %)
Total virtual pins	0
Total memory bits	142,336 / 1,677,312 (8 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

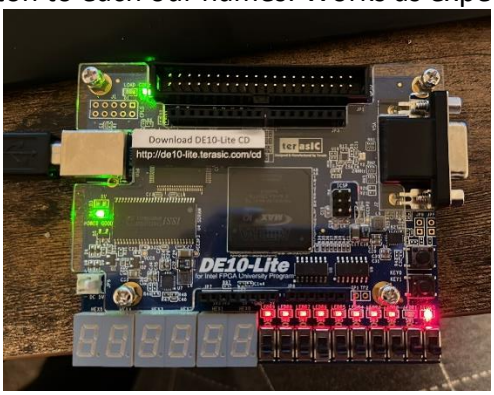
Connections made using Qsys

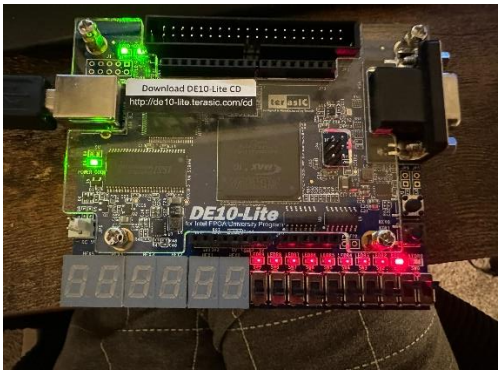


`

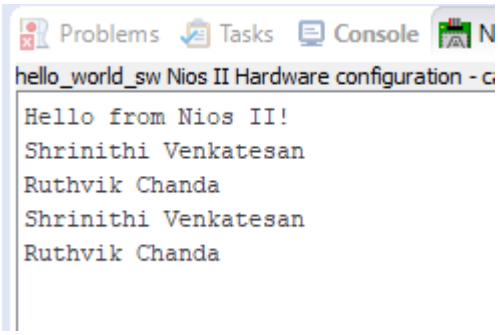
Record your observations of the board's behavior once the FPGA is programmed. Does it behave as you expected?

So, when we program the file, firstly the sw[0] is configured for led[0]. In nios software part, we configure sw[1] to toggle led[1]. Configured the push buttons 0 and 1 and it will display our names when pressed. One button to each our names. Works as expected.





Board displaying LED outputs



Console output from the software after configuring pushbuttons

Answer the following

a. Which component is the Instruction master of the NIOS II connected to? What is the need for this and why it is connected only to one component (slave)?

The Instruction Master of the NIOS II is connected to the On-Chip Memory in this configuration. This connection is crucial because it allows the NIOS II processor to fetch and execute program instructions. The On-Chip Memory acts as the RAM (Random Access Memory) for the processor, holding the program's code section, which contains the executable instructions.

b. How are the LED and the switch connected using S/W?

Led 0 and Sw0 use Inverted Logic and is implemented using nios2(software).

c. How are the LED and the switch connected using H/W?

The LED 1 and Sw1 are connected that when the Sw1 is turned on, the LED1 is ON. Programmed through hardware(.sof)

`

Module 3- CREATE A SYSTEM ON A CHIP WITH PROGRAMMABLE HARDWARE AND A SOFT PROCESSOR

Record the fmax. What is the highest speed clock used in the design?

Ans: Highest Clock Speed from the 7 clocks is 315.76Mhz

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax
1	50.38 MHz	50.38 MHz
2	77.26 MHz	77.26 MHz
3	104.7 MHz	104.7 MHz
4	108.42 MHz	108.42 MHz
5	113.11 MHz	113.11 MHz
6	288.35 MHz	250.0 MHz
7	315.76 MHz	250.0 MHz

Estimate the % utilization of the FPGA logic.

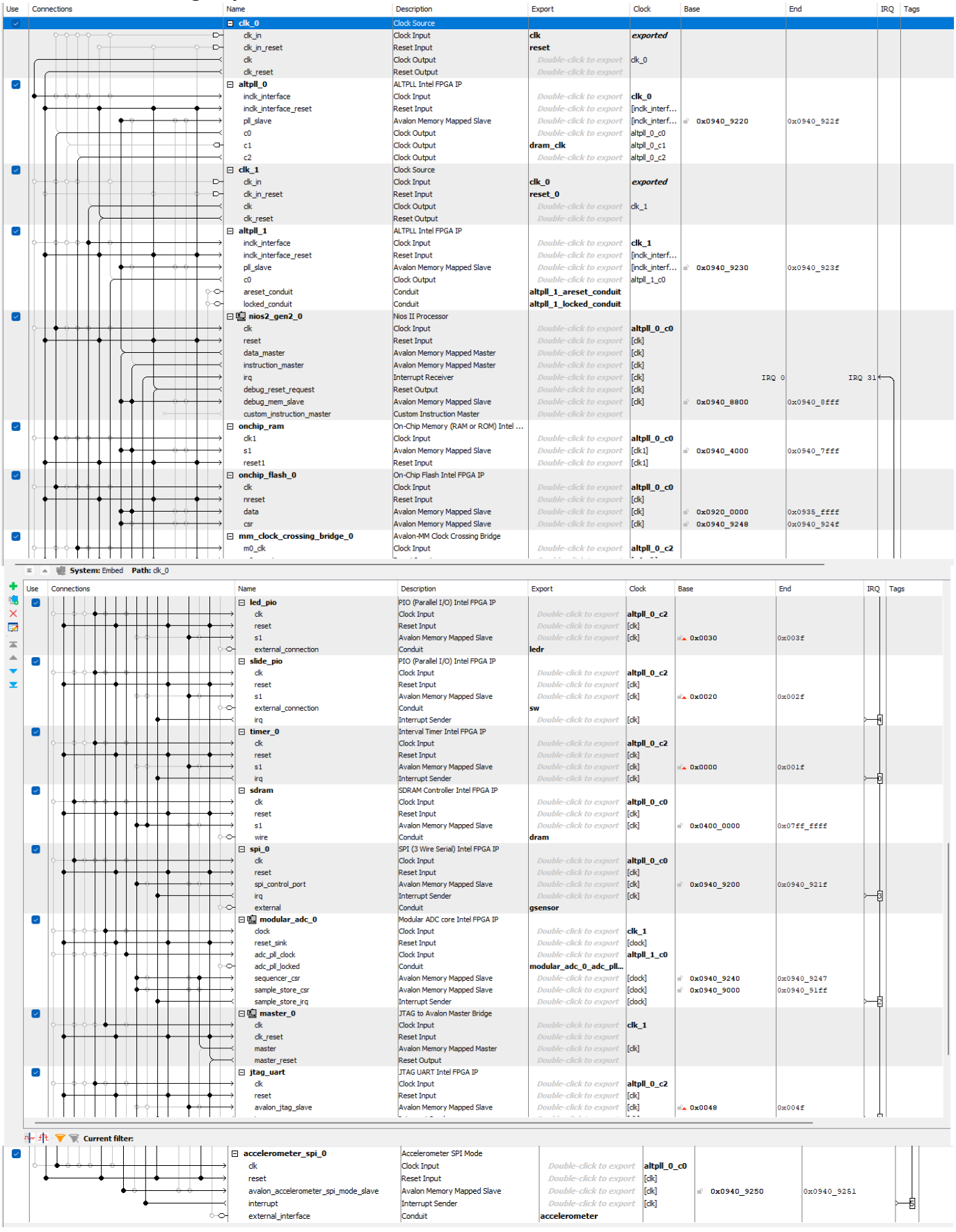
Flow Summary

<<Filter>>

Flow Status	Successful - Tue Oct 03 00:02:48 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Embed
Top-level Entity Name	DE10_LITE_Default
Family	MAX 10
Device	10M50DAF484C6GES
Timing Models	Preliminary
Total logic elements	11,476 / 49,760 (23 %)
Total registers	7358
Total pins	185 / 360 (51 %)
Total virtual pins	0
Total memory bits	759,864 / 1,677,312 (45 %)
Embedded Multiplier 9-bit elements	6 / 288 (2 %)
Total PLLs	3 / 4 (75 %)
UFM blocks	1 / 1 (100 %)
ADC blocks	1 / 2 (50 %)

`

Connections made using Qsys

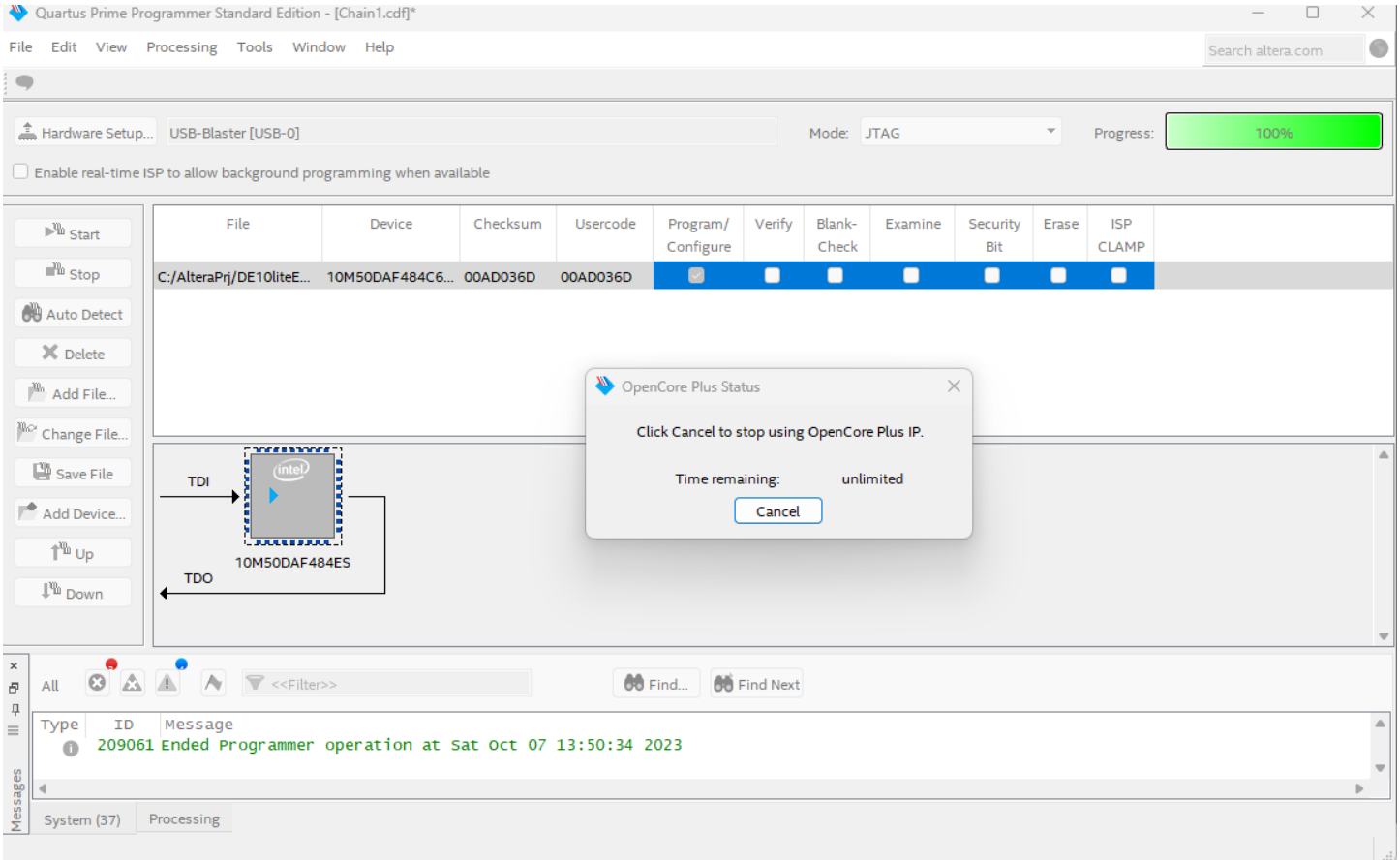


`

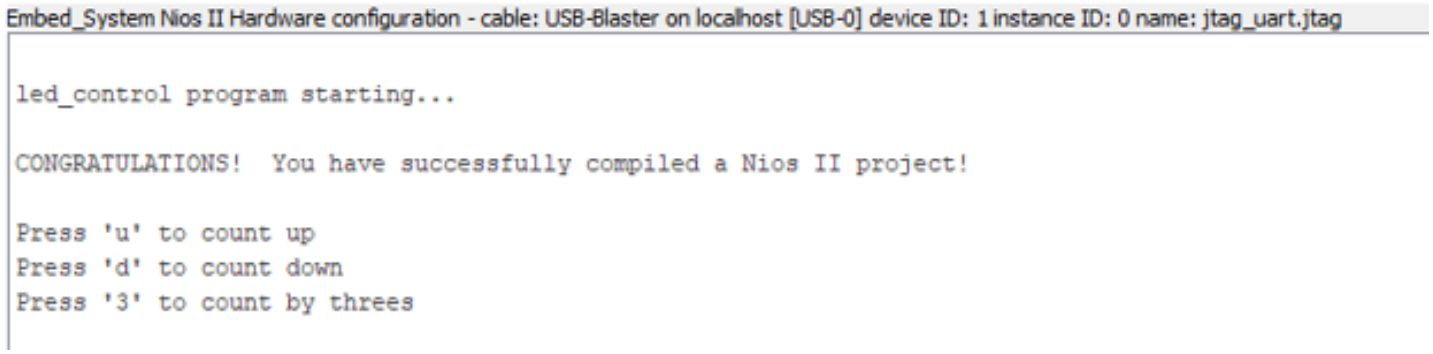
Record your observations of the board's behavior once the FPGA is programmed. Does it behave as you expected?

Ans: Once the hardware and software are completed. While it is programmed, we see a list of menu options on the console for the user as shown in the below pictures. We have added two other menu options 5 and t as said in the Manual. The board behaves as expected. 7 segment display shows the numbers from 1to 9 and then characters from A to F. While the LED shows a pattern starting from LED0 to LED9. While changing the code (*display_value = ~ display_value;*) it inverts the direction.

Extra Credit: When added with accelerometer IP, the board was programmed to read the x axis value. The raw data is then converted to readable values and accordingly the LEDs are configured to grow based on the readings. If x=0, LED0 glows and if x=6, LED0-6 glows,etc.



Programming hardware



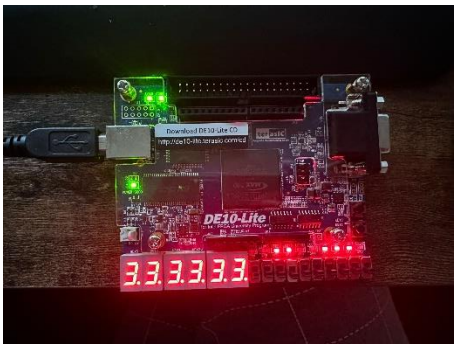
`

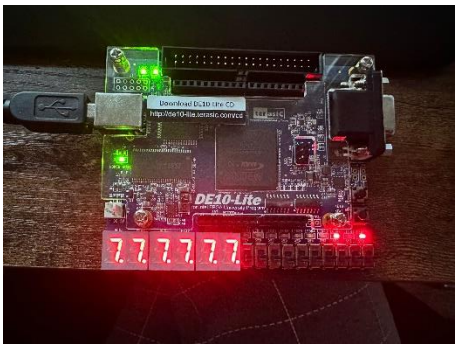
The Updated Console with New Menu Options is shown in the below picture.

Problems | Tasks | Console | Nios II Console | Properties

Embed_System Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart.jtag

```
led_control program starting...

CONGRATULATIONS!  You have successfully compiled a Nios II project!

Press 'u' to count up
Press 'd' to count down
Press '3' to count by threes
Press '5' to count by five
Press 't' to count down by ten

t
You selected: 't'
 -  counting down by 10
0xff 0xf5 0xeb 0xe1 0xd7 0xcd 0xc3 0xb9 0xaf 0xa5 0x9b 0x91 0x87 0x7d 0x73 0x69
0x5f 0x55 0x4b 0x41 0x37 0x2d 0x23 0x19 0xf 0x5 0xfb 0xf1 0xe7 0xdd 0xd3 0xc9
0xbf 0xb5 0xab 0xa1 0x97 0x8d 0x83 0x79
5
0x6f 0x65 0x5b 0x51 0x47 0x3d 0x33 0x29
0x1f 0x15 0xb 0x1 0xf7 0xed 0xe3 0xd9 0xcf 0xc5 0xbb 0xb1 0xa7 0x9d 0x93 0x89
0x7f 0x75 0x6b 0x61 0x57 0x4d 0x43 0x39 0x2f 0x25 0x1b 0x11 0x7 0xfd 0xf3 0xe9
0xdf 0xd5 0xcb 0xc1 0xb7 0xad 0xa3 0x99
```



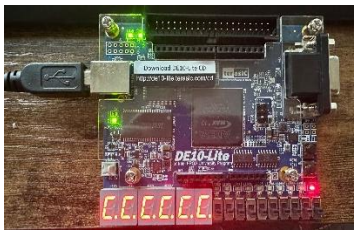


Example board outputs for count by 10 and count down

`

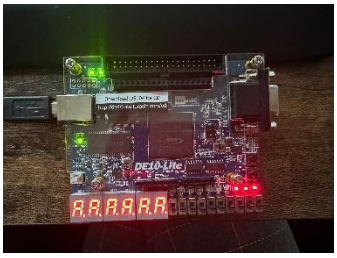
Extra credit:

```
Accelerometer active!
 x = 0

Press 'a' for accelerometer or 'b' for binary counter

a
You selected: 'a'
Switched to accelerometer mode
Accelerometer active!
 x = 2

Press 'a' for accelerometer or 'b' for binary counter

a
You selected: 'a'
Switched to accelerometer mode
Accelerometer active!
 x = 3

Press 'a' for accelerometer or 'b' for binary counter

a
You selected: 'a'
Switched to accelerometer mode
Accelerometer active!
 x = 4

Press 'a' for accelerometer or 'b' for binary counter

a
You selected: 'a'
Switched to accelerometer mode
Accelerometer active!
 x = 5

Press 'a' for accelerometer or 'b' for binary counter

a
You selected: 'a'
Switched to accelerometer mode
Accelerometer active!
 x = 6

Press 'a' for accelerometer or 'b' for binary counter
```

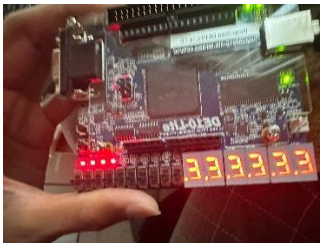


Accel x =0

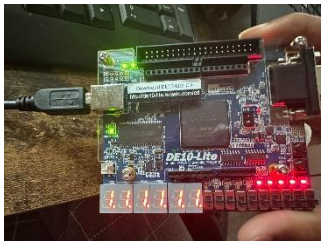
x =1



x =2



x =3



x =4

x =5

`

Answer the following

a. Why is the instruction master of NIOS II connected to multiple components? Explain the usage of each of them.

Ans: The instruction master in an NIOS II-based system connects to several components for various reasons, each fulfilling a specific purpose:
On-Chip Memory: Typically, the instruction master is linked to the on-chip memory. OCM is a fast, low-latency memory built within the FPGA that holds the program instructions required by the NIOS II processor.
Flash Memory: Larger programs or configurations can be stored in flash memory. When the instruction master is linked to external flash memory, the NIOS II processor can read program instructions from flash storage during boot-up or when on-chip memory is insufficient.
SDRAM: Instruction master connects to SDRAM for larger data memory capacity, crucial for data storage and manipulation.
SPI: Instruction master links to SPI for communication with external devices, facilitating data exchange via a serial bus. Commonly used for interacting with peripherals.
ADC: Instruction master connects to the ADC module, allowing NIOS II to convert analog signals into digital values for FPGA processing, useful for precise analog measurements.

b. What's the function for exporting external connection for different components?

Ans: To make signals available at the top level, we export them from Qsys. The Export column in Qsys lets us choose which interfaces are accessible outside the Qsys block. This allows us to connect these signals to other parts of the design or external components.

c. What is the requirement for multiple clocks in this design?

Ans: As we can see from the below picture, different peripherals have different input clock frequencies. For this reason, we need multiple clocks in the design.

Component	Input Clock Frequency	Source	Designation
1. SDRAM PLL	50 MHz	Oscillator on DE10-Lite	MAX10_CLK1_50
2. Nios II processor and peripherals	80 MHz	Output 'c0' of PLL	sdram_pll_c0
3. SDRAM phase shifted clock	80 MHz (-90° phase)	Output 'c1' of PLL	sdram_pll_c1
4. Slow Peripherals	40 MHz	Output 'c2' of PLL	pll_c2
5. ADC PLL	10 MHz	Oscillator on DE10-Lite	ADC_CLK_10
6. internal ADC	10 MHz	Output 'c0' of ADC PLL	adc_pll_c0

Questions:

1. Is the PWM of the LEDs implemented in hardware or in software?

Ans: PWM of the LEDs is done in Quartus. It is done in hardware.

2. Is the control of the 7-segment LEDs done by hardware or by software?

Ans: Control of the 7-segment LEDs done by software.

3. How much memory is required to run your Nios II program? Can you fit it into the onchip RAM if you redesign the onchip RAM block?

Ans: The code size with accelerometer is 67kb. Since my code size (67,000 bytes) exceeds the size of the on-chip RAM (16,384 bytes), it won't fit into the on-chip RAM. But I can expand the memory size accordingly to fit my program.

`

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Processor is already paused
Reading System ID at address 0x08000040: verified
Initializing CPU cache (if present)
OK

Downloading 04000000 ( 0%)
Downloading 040109E8 (95%)
Downloading 09404000 (99%)
Downloaded 67KB in 0.9s (74.4KB/s)

Verifying 04000000 ( 0%)
Verifying 040109E8 (95%)
Verifying 09404000 (99%)
Verified OK
Starting processor at address 0x0400016C
```

4. Your Nios II processor is running at what clock speed? How much faster can it run in your MAX10 design?
Ans: The clock speed set to NIOS2 processor is from c0 of pll0 which is 80Mhz, and the Fmax recorded would be 108.42Mhz in MAX10 design, which is the maximum speed it could run.

5. In a new menu option, "i", the LED Display could be inverted. According to DE10_Embedded_System_LabR2.pdf this may not appear on the options. Can it be implemented from scratch?
Ans: No, implementing from scratch is not an efficient way. Instead we could control it in software through NIOS2. After modifying the software, the blinking of LEDs are inverted. When the counter is counting up, the LEDs blink in a pattern as if the counter is counting down, and vice versa.

```c
#include "system.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include <string.h>
#include "altera_avalon_pio_regs.h"

#include "../inc/main_includes.h"

// declare and initialize global variables for the led sta
volatile alt_u8  LED_STATE = 0;

volatile alt_u8  LED_MASK = 0;

// convenience routine for updating the bank of leds
void update_led(alt_u8 display_value)
{
//display_value = ~(display_value);
LED_STATE = display_value & 0xFF;
LED_STATE &= ~LED_MASK;
IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, LED_STATE);
}

```

6. Ponder the Nios II for a moment. Why do you think the architecture is as it is? What advantages come from building a processor out of programmable logic?
Ans: The Nios II architecture was designed to be adaptable or flexible. We can easily design hardware and software components to meet specific application needs by building a processor out of programmable logic.
7. Open the software source files, including all .c and .h files. What do you observe about the coding style and use of comments?
Ans: The coding style is clear and well-documented with comments, allowing anyone evaluating or maintaining the code to comprehend it. It should be noted, however, that the code could be part of a larger project, and certain sections are currently unused or commented off.
8. When you modified the software, how did the LED display change?
Ans: The LEDs will now display the inverted or complemented value of display_value.

Lessons Learned

Module 1- DEVELOP A MIXED-SIGNAL SYSTEM

- Developed a voltmeter that alters LED intensity dependent on voltage and shows ADC values on a 7-segment display
- Learnt about analog-to-digital conversion (ADC) and how to interface it with an FPGA to read voltage data.
-This module provides a thorough hands-on experience in FPGA hardware design, covering all stages of the design flow, actual applications, and opportunities for design improvement and development.

`

Module 2- DISPLAY HELLO WORLD AND TOGGLE LEDS USING NIOS II AND Q SYS

- We learned how to design a hardware system using QSys, incorporating essential components such as a processor, RAM, UART, switches, and LEDs.
- We gained knowledge about mapping signals in the HDL language, specifically Verilog, to connect hardware components and define their functionality within the FPGA design.
- We explored the use of software tools, such as the NIOS II Software Build Tools in Eclipse, to develop and run programs on the designed hardware system. This included printing messages on the console and utilizing push buttons to control program behavior.

Module 3- CREATE A SYSTEM ON A CHIP WITH PROGRAMMABLE HARDWARE AND A SOFT PROCESSOR

- Proper clock management is critical, especially when using several clocks.
- Using peripheral interfaces such as SPI improves system functionality.
- Memory management is critical for on-chip RAM, flash memory, and external SDRAM.
- Qsys signal exporting enables connecting with external components.
- Additionally learnt how to configure an extra accelerometer module and programmed in it software to acquire values from onchip accelerometer.

Conclusions

Module 1- DEVELOP A MIXED-SIGNAL SYSTEM

PWM Implementation: Using previous PWM (Pulse-Width Modulation) expertise, this project has enabled me to properly apply PWM techniques, understanding how they can be utilized to control LED intensity and interface with the ADC.

Control for the 7-Segment Display: Implementing control for the 7-segment display has increased my understanding of multiplexing techniques and how to operate external peripherals from an FPGA.

ADC Integration: This project taught me how to integrate analog-to-digital converters (ADCs) into FPGA designs. I learned about sampling, quantization, and integrating ADC data with digital logic.

Module 2 and 3- DISPLAY HELLO WORLD AND TOGGLE LEDS USING NIOS II AND Q SYS. CREATE A SYSTEM ON A CHIP WITH PROGRAMMABLE HARDWARE AND A SOFT PROCESSOR

These modules were a great start to utilizing Qsys as a tool and designing a softcore processor. These exercises highlights the significance of systematic FPGA design, which begins with clear system requirements. It focuses on the integration of various hardware components, good clock management, software development, and stringent testing. These lessons provide a solid foundation for future FPGA projects, highlighting the need for adaptability and careful documentation.

Appendix: References

[1] Intel Altera. (2016). *Cyclone V Device Handbook*. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/fpga/cyclone-series/cyclone-v/support.html

[2] Intel Altera. (2016). *Max 10 Device Handbook*. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/fpga/max-series/max-10/support.html

[3] Intel Altera. (2016). *Qsys System Design Tutorial.* [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_qsys_intro.pdf

`

Code for hello_world:

```
/*Submitted by Shrinithi Venkatesan and Ruthvik Chanda*/
#include "sys/alt_stdio.h"
#include <stdio.h>
#include "altera_avalon_pio_regs.h"
#include "system.h"
int main()
{
int switch_datain;
int key_0_datain;
int key_1_datain;
alt_putstr("Hello from Nios II!\n");
/* Event loop never exits. Read the PB, display on the LED */
while (1)
{
//LED toggles according to switch 1 state
switch_datain = IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE);
IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,switch_datain);
//Names are displayed according to key0 and key1 state
key_0_datain = IORD_ALTERA_AVALON_PIO_DATA(KEY_0_BASE);
key_1_datain = IORD_ALTERA_AVALON_PIO_DATA(KEY_1_BASE);

if(key_0_datain == 0)
{
      alt_putstr("Ruthvik Chanda\n");
      /*Two methods to perform Debouncing: 1.  Waiting in a loop until the button's state
remains stable for a period (Implemented).
                                          2. Setting delay for the print function */
    while(IORD_ALTERA_AVALON_PIO_DATA(KEY_0_BASE) == 0);
}
if(key_1_datain == 0)
{
      alt_putstr("Shrinithi Venkatesan\n");
    while(IORD_ALTERA_AVALON_PIO_DATA(KEY_1_BASE) == 0);
}
}
return 0;
}
```

Code integrating accelerometer and binary counter:

```
// include the c and hal files that we need in main()
#include <string.h>
#include "sys/alt_stdio.h"
#include "alt_types.h"
#include "altera_up_avalon_accelerometer_spi.h"

// these includes allow us to perform a quick sanity check that the system was properly
assembled in SOPC Builder
#include "system.h"
#include "inc/system_validation.h"

// all helper routine includes are condensed into this one file for neatness
#include "inc/main_includes.h"

#define MAX_COUNT 0xFF  //use a 8-bit maximum count value since we have a 8-bit LED PIO

//initialize accelerometer value variable
alt_up_accelerometer_spi_dev * accelerometer;
```

`

```
//initialize accelerometer
void init_accel() {
    accelerometer = alt_up_accelerometer_spi_open_dev("/dev/accelerometer_spi_0");
    if (accelerometer != NULL) {
        alt_printf("Accelerometer active!\n ");
    }
}

int main(void) {
    alt_u8 binary_count;
    char input;
    int dir;
    alt_u32 xAccel = 0;
    alt_u8 mode = 'b'; // Default mode is binary counter

    alt_printf("\nled_control program starting...\n\n");
    alt_printf("CONGRATULATIONS!  You have successfully compiled a Nios II project!\n");

    alarm_init();


    while (1) {
        alt_printf("\nPress 'a' for accelerometer or 'b' for binary counter\n");
        input = alt_getchar();
        alt_getchar();
        alt_printf("You selected: '%c'\n", input);

        if (input == 'a') {
            mode = 'a'; // Switch to accelerometer mode
            alt_printf("Switched to accelerometer mode\n");
            init_accel();
        } else if (input == 'b') {
            mode = 'b'; // Switch to binary counter mode
            alt_printf("Switched to binary counter mode\n");
        } else {
            alt_printf("INVALID ENTRY\n");
        }

        if (mode == 'a') {
            // Accelerometer mode
            alt_up_accelerometer_spi_read_x_axis(accelerometer, (alt_32*)&xAccel);

            alt_u8 raw_xval = (alt_u8)(xAccel % 255);
            alt_u8 xval = ((7 * raw_xval) / 255);

            alt_u8 Led_Val = 0x00;

            for (alt_u8 i = 0; i <= xval; i++) {
                // Set the (LED) in LED_Vals
                  Led_Val |= (1 << i);
            }

            printf("x = %u\n", xval);

            update_led(Led_Val);
            delay_wait();

        } else if (mode == 'b') {
            // Binary counter mode
            // declare the local variables need in main()

            alt_printf("\nled_control program starting...\n\n");
`
```

```
            alt_printf("\nPress 'u' to count up by one\n");
            alt_printf("Press 'd' to count down by one\n");
            alt_printf("Press '3' to count up by threes\n");
            alt_printf("Press 'q' to count up by fives\n");
            alt_printf("Press 'w' to count down by tens\n");
            input = alt_getchar();
            alt_getchar();
            alt_printf("You selected: '%c'\n",input);
            if (input == 'u') {
            dir = 1;
            binary_count = 0;
            alt_printf(" - counting up by 1\n");
            }
            else if (input == 'd') {
            dir = -1;
            binary_count = MAX_COUNT;
            alt_printf(" - counting down by 1\n");
            }
            else if (input == 'w') {
            dir = -10;
            binary_count = MAX_COUNT;
            alt_printf(" - counting down by 10\n");
            }
            else if (input == '3') {
            dir = 3;
            binary_count = 0;
            alt_printf(" - counting up by 3\n");
            }
            else if (input == 'q') {
            dir = 5;
            binary_count = 0;
            alt_printf(" - counting up by 5\n");
            }
            else {
            dir = 1;
            binary_count = MAX_COUNT;
            alt_printf("INVALID ENTRY");
            }
            // initialize the line wrap count variable
            PRINT_STDIO_WRAP_COUNT = 0;

            // print the binary count out the STDOUT
            print_binary_count_stdio(binary_count); // see src/jtag_uart_util.c
            // update the green led display with the binary count
            update_led(binary_count); // see src/led_util.c
            // wait for the delay period
            delay_wait(); // see src/delay_wait.c
            // count until we reach all the maximum count
            while( (binary_count < MAX_COUNT && dir > 0) || (binary_count > 0 && dir < 0)
)
            {
            if ((dir == -10) && (binary_count) < 0xA)
            {
            break;
            }
            // increment the binary counter
            binary_count= binary_count + dir;
            // print the binary count out the STDOUT
            print_binary_count_stdio(binary_count); // see src/jtag_uart_util.c
            // update the green led display with the binary count
            update_led(binary_count); // see src/led_util.c
            // wait for the delay period
`
```

```
            delay_wait(); // see src/delay_wait.c
            }
            // announce loop completion on STDOUT and the UART
            alt_printf("\n\n LED control program completed its loop ...\n\n");
            // wait for the delay period
            delay_wait(); // see src/delay_wait.c
            }
    }

            // we should never get to this point
            return(0);
            }
```

`