ECE 385

Fall 2020

Experiment #7

Introduction to SoC with NIOS II

Alex Wen (acwen2), Manav Agrawal (manava3)

Section ABF

Xinbo Wu

Introduction

This lab functions to implement designs with NIOS II processor as an introduction to System-on-Chips (SoC). The NIOS II processor is a central processing unit that can operate through a high-level language (C) and implement software programming algorithms for dataflow, along with platform modules to designate the hardware components, which are then processed through the FPGA. The application and interaction between hardware and software implementation for SoC designs are performance based. That is, the NIOS II handles tasks that do not require high performance (user interface, data I/O) while the FPGA takes care of the high-performance tasks; the interactivity of these two components makes for an efficient design. In this lab, the objective is to begin the understanding of SoC Designs and NIOS II processors by interfacing a datapath algorithm in C with hardware peripherals on the FPGA (including buttons, on-board switches, and LEDs).

NIOS II Processor

The NIOS II architecture is an embedded processor designed such that it can be configured with a field-programmable gate array (FPGA). More specifically, it is a modified Harvard reduced instruction set computer (RISC) interacting with the Avalon Bus that processes the instruction and data that is configured within its processor core and allows for the transfer of information between the processor itself and the rest of the FPGA. The Avalon Bus is a 32-bit memory-mapped (MM) interface that allows for devices interfaced on the bus to be assigned a block of addresses such that it is compatible with the software algorithm; it is functioned through Qsys or the Platform Designer. Since the NIOS II maintains a separability in terms of the design functions relative to the rest of the FPGA - for the software compilation - the processor maintains its own reset, clock and its own I/O signals. The software implementation of the design also enables such that the processor is able to configure a debug module which, in this case, is interfaced with the JTAG module.

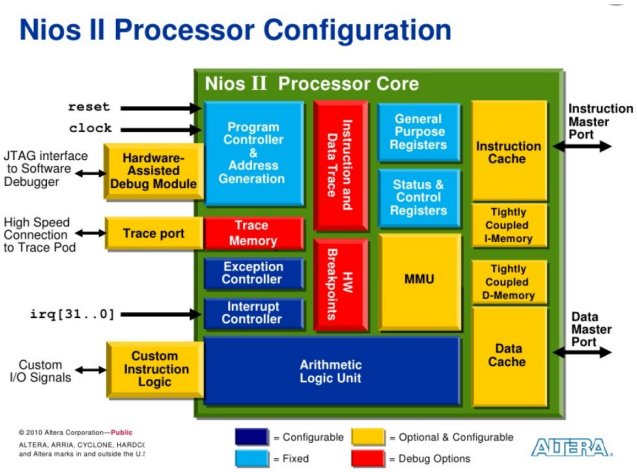


Figure 1: Full NIOS II Processor Configuration from Altera

The above figure shows the full configuration of the NIOS II but this lab will use a NIOS II/e which is a more economical version that does not have cache or Memory Protection unit and it is simpler as it trades speed with space complexity. The hardware consists of ALU, registers for status, control, data, controllers and many more which are in-built in the FPGA which will be used by the different array of software programs.

Top Level Block Diagram

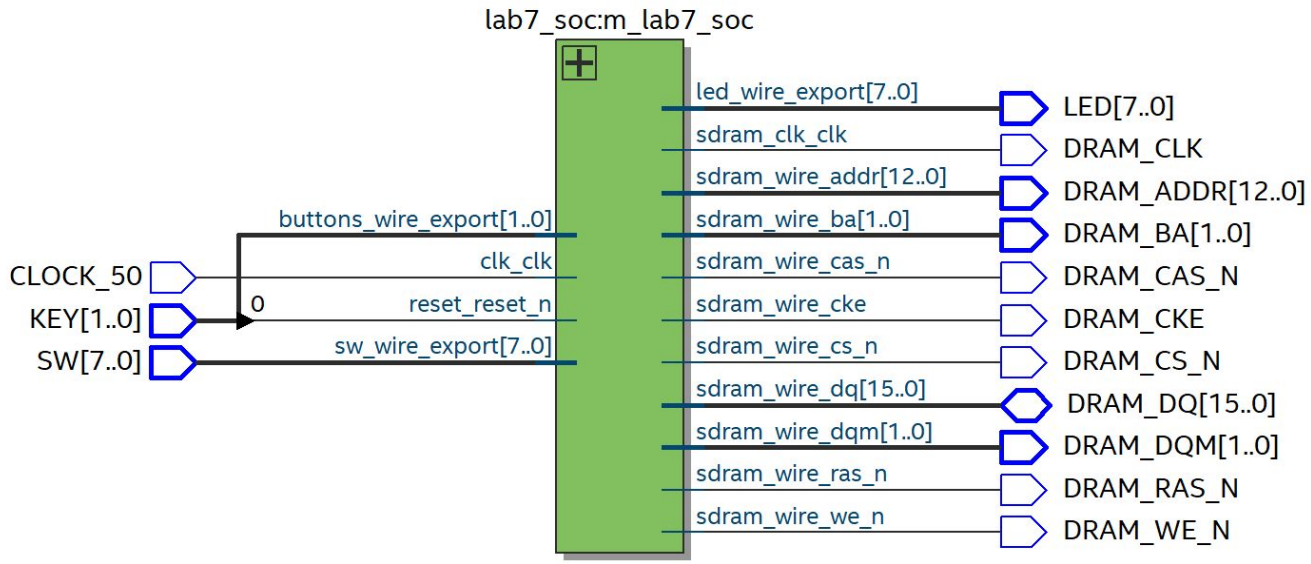


Figure 2: Top Level Diagram Lab 7 NIOSProcessor

This is the top-level while which combines the NIOS processor, on-chip memory, SDRAM, parallel I/O modules and a lot more which can be referred to the platform design also. Just to give an idea of how big the block diagram can be, it can be seen in the next image.

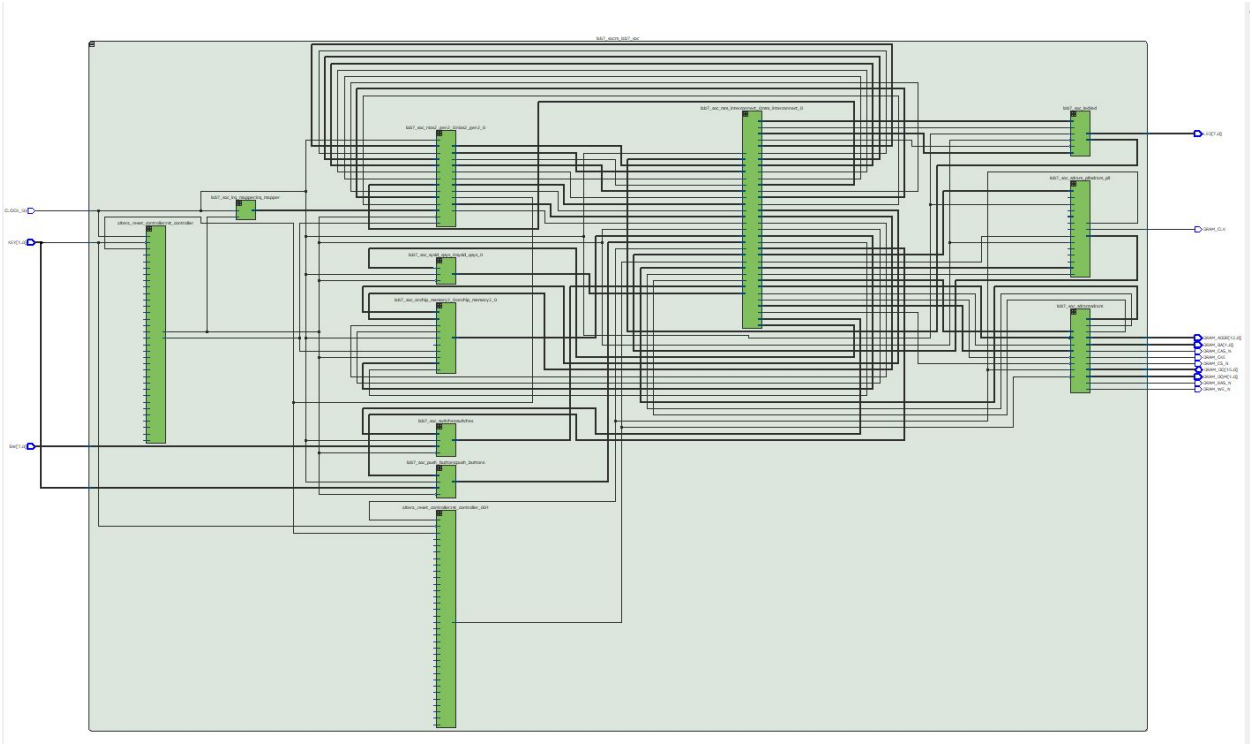


Figure 3: Inside the top level of Lab 7

This shows all the modules being bridged together which creates the entire functionality.

Figure 4: In-depth look at NIOS, On-Chip Memory and the main interconnect component

These 2 images are interesting as the top left image shows the block for the NIOS II processor which shows all the necessary inputs and outputs. The Peripheral I/O is also visible below the processor. Later after the sys_id, we can also see the on-chip memory which will store all the instructions that the FPGA may need and it is beneficial to have on-chip memory which will be explained throughout the lab. The right component shows the main block which combines and connects all the necessary inputs and outputs together. The Platform designer will go in detail for each module

The HDL modules for FPGA implementation will be described thoroughly in the Appendix.

Systems Level Block Diagram

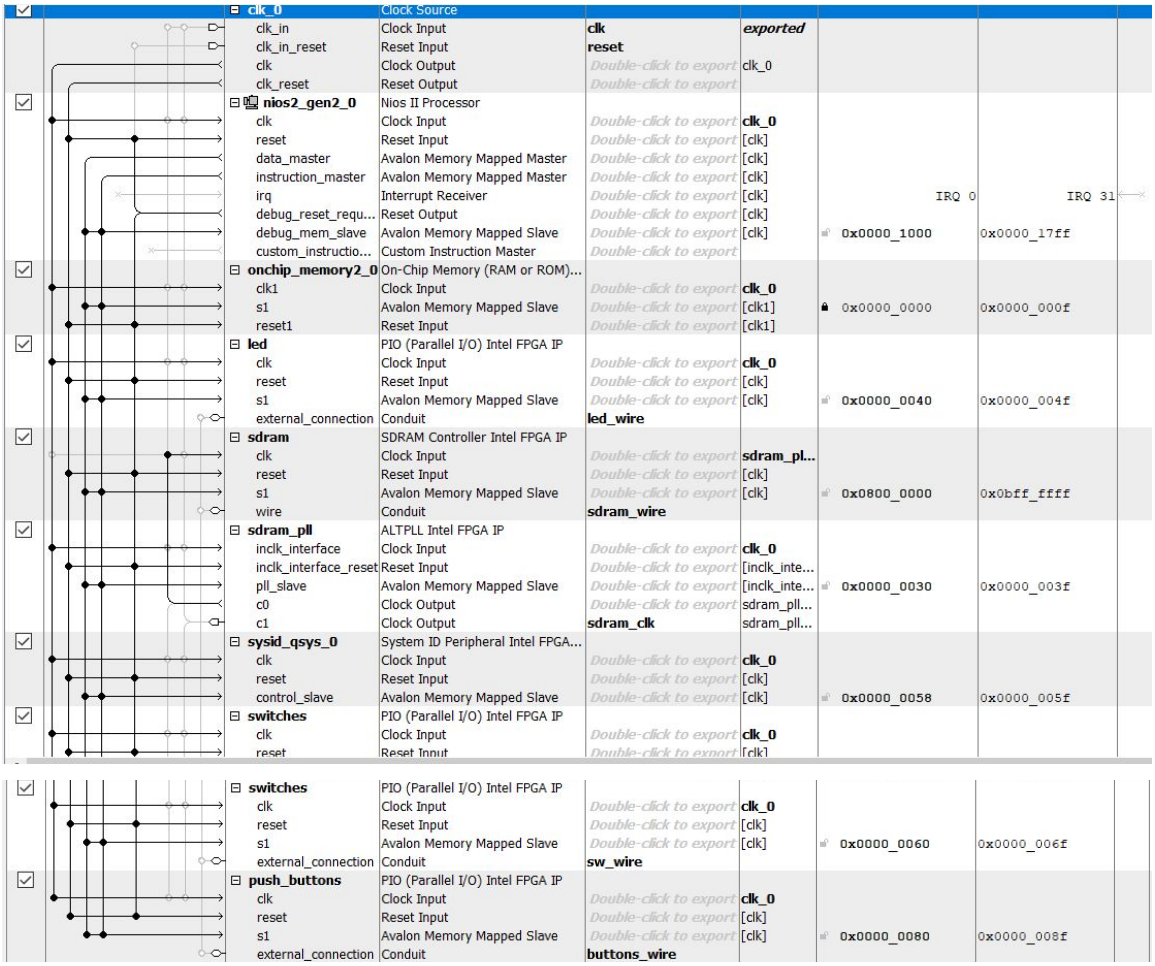
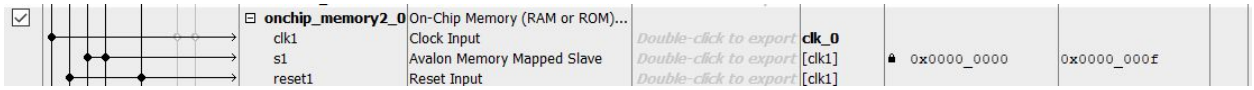


Figure 5: General Platform Designer View

This is the general snapshot of all the modules about each of the components and now we will go in detail what each and every component does.

This module is about providing the clock inputs which will be used by all the components except SDRAM. SDRAM has a separate clock but all the other components are synchronous to 50 MHz clock.

This is the NIOS/e processor which is the entire brain of this lab. As you can see there is a separate data and instruction bus but it can be accessed the same which shows that this is a modified harvard architecture. There is an IRQ which provides the interrupt, the Avalon Memory Mapped Slave is a 32 bit memory mapped interface which is shared by a lot of memory and other components.

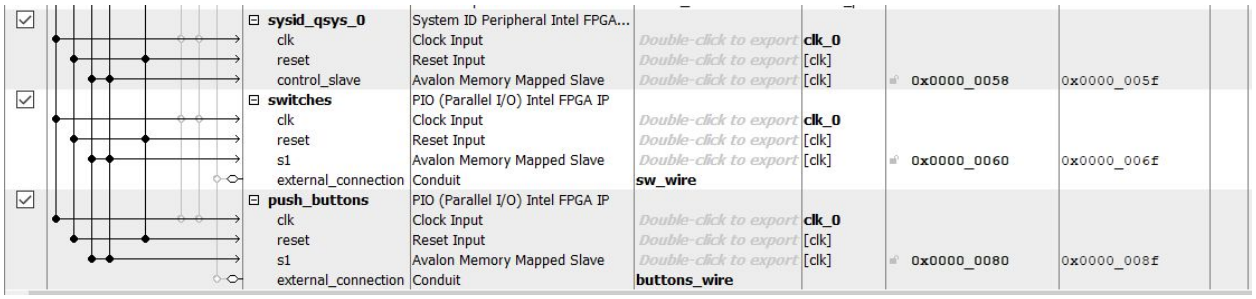


This is the on chip memory which is used as a good way to access data to decrease latency. This also has the memory mapped slave with a reset input and a synchronous clock with a total memory size of around 16 bytes.

This is a PIO module which will provide the data output. The Avalon Memory Mapped slave will take the input from the memory, or the NIOS processor. The clock is synchronous and reset functionality. This is done to provide the Input/Output functionality. The PIO modules that we chose had the data width of 8 bits since 8 LEDs.

This is the SDRAM (Synchronous Dynamic Random Access Memory) which is very important. It is assigned the address 0x30 to 0x3f for the phase locked loop (PLL) and the actual SDRAM

controller was assigned the address 0x0800000 to 0x0bffffff. The clock for the SDRAM is different (explained in questions) as it has a phase shift. Both require the avalon 32 bit memory mapped slave data bus. The PLL is a useful component as it provides a clock with a phase offset to account for delays and SDRAM stability. There will be a table for the SDRAM size in the post-lab inquiries which is really helpful in figuring how this SDRAM was constructed.



The last part has a couple of more modules. The first part is the System ID peripheral which is a read-only that provides the SOPC. Creates an identification and identity for each of the modules. The second two are the parallel Input/Output Intel FPGA IP. The switches provide input to this and puts it in Avalon Memory Mapped Slave. The push_buttons are the same and it is implemented for Key 0 and Key 1. Every module will be assigned a different address otherwise it would conflict and overwrite the 32 bit avalon memory mapped slave. The sw_wire and the buttons_wire are exported as this will act as inputs from the FPGA. The PIO module for switches was 8 bits and the push_buttons was 2 switches.

Software Component of Lab

The exact functionality of the code is better described in the post-lab section. The statistics are in the next section and the post-lab inquiries are after that. This section will just generally describe how and what is the software component. The FPGA compiles the platform designer and creates the HDL which is then compiled to create and run a program on FPGA. Now this can then be compiled into a board supporting Package which we can then use low-level programming skills to utilize the modules and create different functionality. Here we had used C to code the accumulator and blinker.

Design Resources and Statistics
Table Statistics

	NIOS Processor
LUT	2278
DSP	0
Memory (BRam)	10,368 / 1,677,312 (< 1 %) - Total Block Memory Bits 36,864 / 1,677,312 (2 %) - Total Block Memory Bit Implementation
Flip-Flop	1740 Logic Registers, With I/O it is 1809 Registers
Frequency	84.52 MHz
Static Power	96.44 mW
Dynamic Power	48.40 mW
Total Power	162.23 mW

INQ Inquiries

What are the differences between the Nios II/e and Nios II/f CPUs?

Nios II/e is resource optimized whereas Nios II/f optimizes speed and performance (achieves maximum DMIPS/MHz)

Nios II/f has Data Cache, shadow register sets which are not included in NIOS II/e. The data caches enable for faster operations but take more resources hence it is not available in Nios II/e. The NIOS II/f has an optional memory management and protection unit which makes it protected and more efficient as compared to NIOS II/e which does not possess such functionality.

#We have instantiated 16 bytes (128 bits) of on-chip memory.
What advantage might on-chip memory have for program execution?
On chip memory requires no board space or wiring as it can directly be implemented by the FPGA. This saves some development time and potential future costs. In addition, since the memory is on chip, it can decrease some of the latency for frequently accessed data

Note the bus connections coming from the NIOS II; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?

Modified Harvard Architecture allows the contents of the instruction memory to be accessed as data. Most modern computers are modified Harvard Architecture. The NIOS II processor seems to be a modified Harvard architecture since the instruction memory can be accessed as data. The class notes says that NIOS II is a modified Harvard RISC architecture and that even though in the platform designer the buses are separate, the data and instruction memory share the same space and can be accessed together.

Note that while the on-chip memory needs access to both the data and program bus, the led peripheral only needs access to the data bus. Why might this be the case?
The program bus is used for memory related components whereas we want to display the data from memory and the program therefore, the LEDs will only use the data bus to display the content. The data bus will be used for the LED peripheral.

Why does SDRAM require constant refreshing?
SDRAM (Synchronous Dynamic Random Access Memory): DRAM is usually built with capacitors and transistors. The capacitor will discharge with time therefore, it needs constant refreshing to maintain its voltage level otherwise the information will be lost.

*Note that there is one 32M*16 chips, so the total amount of memory should be 512MBits (64 Mbytes), make sure this is consistent with your above numbers; you will need to justify how you came up with 512 Mbit to your TA.*

SDRAM parameter	Short name	Parameter value (fill in from the data sheet)
Data Width	[width]	16 bits
# of Rows	[nrows]	8k(A0 - A12) (13 rows)
# of columns	[ncols]	1k(A0 - A9) (10 cols)
# of Chip Selects	[ncs]	1 chip select (CS)’
# of Banks	[nbanks]	4 banks (BA0 and BA1 for addressing it)

512MBit = Data Width in each address * # of rows of Addr * # of Addr Cols * # of banks = 16 * 8k*1k*4

What is the maximum theoretical transfer rate to the SDRAM according to the timings given?
Fastest Rate
Frequency from the datasheet: 145 Mhz
# of bits: 16
Transfer rate: 145*16 = 2320 Mb/s or 2.320 Gbps

The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?
A slower clock would mean that the refresh rate for the SDRAM will decrease hence increasing the possibility of SDRAM losing its charge and therefore losing bits of information. This will not be good for program/memory storage. The capacitor has a specific time constant and if the clock's time period extends that, the refresh rate and the storage of information will be accessed. When SDRAM transactions are done, the address, data and control signal have a valid time interval or a window for which the pins can be accessed and will yield data in a systematic manner, at slower frequencies it runs with different risk of not good refreshing rate and with faster frequencies, it experiences latency errors.

Make another output by clicking clk c1, and verify it has the same settings, except that the phase shift should be -1ns. This puts the clock going out to the SDRAM chip (clk c1) 1ns behind of the controller clock (clk c0). Why do we need to do this? Hint, check Altera Embedded Peripheral IP datasheet under SDRAM controller.
For the SDRAM to be provided the right inputs and outputs synchronously, there needs to be a valid window and its latency to be accounted for so it can get it at the correct time. This creates the glitches not to arize and ensures data stability.

A possible type of latency that affects the timing:
There is a Common Address Strobe latency which is the time delay between the command and actually the data output. In addition, there is a time delay between accessing and outputting the data so you would want to activate the SDRAM a little before the processor so that latency can be taken care of so when the clock rises up, correct inputs are going in properly and it is a synchronous, smooth processor.

What address does the NIOS II start execution from? Why do we do this (reset and exception vectors, choose sdram.s1) step after assigning the addresses?
NIOS II starts execution from address 0x0800 0000 since that is where the starting address of the SDRAM chip is located.
0x0bff ffff - Ending Address

The reset and exception vectors are edited after assigning the addresses since the reset vector asks for sdram.s1 and its address and if the addresses are not assigned, the offset and addressing can be wrong.

You must be able to explain what each line of this (very short) program does to your TA. Specifically, you must be able to explain what the volatile keyword does (line 8), and how the set and clear functions work by working out an example on paper (lines 13 and 16).

```
int i = 0;
volatile unsigned int *LED_PIO = (unsigned int*)0x40;

*LED_PIO = 0; //clear all LEDs
while ( (1+1) != 3) //infinite loop
{
    for (i = 0; i < 100000; i++); //software delay
    *LED_PIO |= 0x1; //set LSB
    for (i = 0; i < 100000; i++); //software delay
    *LED_PIO &= ~0x1; //clear LSB
}
return 1; //never gets here*/
```

Blinker Code:

One of the first things we need to do is to set a variable which has the address of the LEDs. This will help us access it and store any data to it or from it. Volatile keyword is used to indicate that the value of the variables might change outside the scope of the program. It is always updated in the memory. So for example we define the switches as volatile as it tells the compiler that the switches may change any time without any action being taken by the code. After setting the LED to 0 to clear it, we start with the main body of the blinker code. Lines 13-16 are situated inside an infinite while loop. Line 13 sets up a for loop for i from 0 to 100,000. Here we will make the least significant bit 1 and the i iterates through 100,000 such that it provides a time frame for which the light remains on. Line 14 is a simple OR operation where we set the least significant bit to 1 as we do LED or Hex value 1. Line 15 is another for loop which will be used to set it to 0. It iterates i from 0 to 100,000 such that it provides the same time frame or duration for which the light is off. Line 16 is where we actually clear the least significant bit by doing an AND operation between the LED and not x1 which is 111111….0 where we just clear the last bit. This was the general idea of the Blinker code.

```
volatile unsigned int *LED_PIO = (unsigned int*)0x40;
volatile unsigned int *SW = (unsigned int*)0x60;
volatile unsigned int *BUTTON = (unsigned int*)0x80;
int flag = 0;

while( (1+1)!=3){
    if(*BUTTON == 1 && flag == 0){
        *LED_PIO += *SW;
        flag = 1;
    }
    if(*BUTTON == 3) flag = 0;
}
return 1;
```

Accumulator Code:
For an accumulator, since we also use buttons and switches, we need to also set variables with its address like we did with LED for blinkers. Herre, the buttons and switches will provide us the required inputs. So to achieve this, an infinite while loop was created in which if we want to accumulate and we are pressing it, then it will add the value from switches to whatever LED it already has. The difference is there was a flag functionality otherwise if we will press it, it will just continue adding. Additional if statement was added that if reset, clear the LEDs.

Look at the various segments (.bss, .heap, .rodata, .rwdata, .stack, .text), what does each section mean? Give an example of C code which places data into each segment, e.g. the code: const int my_constant[4] = {1, 2, 3, 4} will place 1, 2, 3, 4 into the .rodata segment.

.text segment: Sections of a program in an object file or in memory which contains instructions. Machine code, main().

.bss segment: int i; This is the segment where uninitialized data is usually stored. It also stores global variables initialized to 0. Block Starting Signal is the full form of this segment.

```
Float a;
Int main(){
        Int x;
}
```

.heap: Dynamic memory allocation takes place in the heap by using keywords in C such as ‘malloc, to allocate memory.

```
Int size;
Int * x = (int *) malloc(size);
free(x);
```

.rodata: .rodata stores const variables and its values

```
Const float x = 3.14;
```

.rwdata: This is stored in the RAM area and the value of variables can be altered during the runtime (any point of time). This differs from .rodata as in rodata you can only read-only.

.stack: Stack automatically stores variables each time a function is called.

```
Int calcLargest (int a, int b, int c){
        if(a>b&&a>c) return a;
```

}

Source: https://www.geeksforgeeks.org/memory-layout-of-c-program/#:~:text=A%20text%20segment%20%2C%20also%20known,stack%20overflows%20from%20overwriting%20it.

Conclusion

The functionality of the design went as it should to full effect. In other words, the FPGA properly increments according to the set on-board switches while the HEX board displays the values accordingly. Since this lab is implemented on 8 bits, the accumulator is bounded to 256, and it was demonstrated through proper output overflow. In other words, adding 1 to a stored value of 255 would reset the accumulation to 0 and adding 2 to 255 would reset to 1, and so on. The push buttons also work correctly according to the functions they are designated to operate - by updating the accumulator with set value and resetting the accumulator.

In the design process, there were 3 main points of troubleshooting. The first one was that we had failed to realize that when we press the button, it does not mean it is 1; it was inverted so we needed to correctly identify which value meant what. It also took some time to debug the constrained paths as the path/source names had to be specific and false paths needed to be carefully laid out. Finally, there were just minor coding errors here and there but this lab was a great learning experience.

The instruction for this lab is fairly straightforward - in particular towards the project creation of System-on-Chips design and implementation. It could however be more beneficial in terms of understanding SoC design if we have a more detailed understanding of the Qsys interface. Also, in order to fully ascertain the functionality of the design, the timing constraints needed to be properly implemented (to the proper sources and false paths implementation) - there could be more details in regards to adding the constraints into the design.

APPENDIX

I. Description of SV Modules

```
module lab7(     input         CLOCK_50,
                 input  [1:0]  KEY,
                 input  [7:0]  SW,
                 output [7:0]  LED,
                 output [12:0] DRAM_ADDR,
                 output [1:0]  DRAM_BA,
                 output        DRAM_CAS_N,
                 output        DRAM_CKE,
                 output        DRAM_CS_N,
                 inout  [15:0] DRAM_DQ,
                 output  [1:0] DRAM_DQM,
                 output        DRAM_RAS_N,
                 output        DRAM_WE_N,
                 output        DRAM_CLK

              );

              // You need to make sure that the port names here are identical to the port names at
              // the interface in lab7_soc.v
              lab7_soc m_lab7_soc (.clk_clk(CLOCK_50),
                                   .reset_reset_n(KEY[0]),
                                   .buttons_wire_export(KEY),
                                   .led_wire_export(LED),
                                   .sw_wire_export(SW),
                                   .sdram_wire_addr(DRAM_ADDR),    //  sdram_wire.addr
                                   .sdram_wire_ba(DRAM_BA),        //  .ba
                                   .sdram_wire_cas_n(DRAM_CAS_N),    //  .cas_n
                                   .sdram_wire_cke(DRAM_CKE),      //  .cke
                                   .sdram_wire_cs_n(DRAM_CS_N),      //  .cs_n
                                   .sdram_wire_dq(DRAM_DQ),        //  .dq
                                   .sdram_wire_dqm(DRAM_DQM),      //  .dqm
                                   .sdram_wire_ras_n(DRAM_RAS_N),    //  .ras_n
                                   .sdram_wire_we_n(DRAM_WE_N),      //  .we_n
                                   .sdram_clk_clk(DRAM_CLK)        //  clock out to SDRAM from other PLL port
                                   );

            //Instantiate additional FPGA fabric modules as needed
endmodule
```

Module: lab7.sv

Inputs: CLOCK_50, KEY[1:0], SW[7:0]

Outputs: LED [7:0], DRAM_ADDR[12:0], DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_DQM[1:0], DRAM_RAS_N, DRAM_WE_N, DRAM_CLK

Inouts: DRAM_DQ [15:0]

Description: This shows all the inputs are mainly the switches and the FPGA clock with outputs being mostly related to SDRAM which will be connected to FPGA pins. This is the upper-level entity.

Purpose: The purpose of the SV file is that it is a top level that brings the NIOS processor (lab7_soc) and the entire memory (on-chip and SDRAM) components. It provides a pathway for LED, Switches, Push buttons and the lab7_soc processor. It combines the whole NIOS Processor together.

```
`timescale 1 ps / 1 ps
module lab7_soc (
      input  wire [1:0]  buttons_wire_export, // buttons_wire.export
      input  wire        clk_clk,             //         clk.clk
      output wire [7:0]  led_wire_export,     //     led_wire.export
      input  wire        reset_reset_n,       //       reset.reset_n
      output wire        sdram_clk_clk,       //   sdram_clk.clk
      output wire [12:0] sdram_wire_addr,     //  sdram_wire.addr
      output wire [1:0]  sdram_wire_ba,       //            .ba
      output wire        sdram_wire_cas_n,    //            .cas_n
      output wire        sdram_wire_cke,      //            .cke
      output wire        sdram_wire_cs_n,     //            .cs_n
      inout  wire [15:0] sdram_wire_dq,       //            .dq
      output wire [1:0]  sdram_wire_dqm,      //            .dqm
      output wire        sdram_wire_ras_n,    //            .ras_n
      output wire        sdram_wire_we_n,     //            .we_n
      input  wire [7:0]  sw_wire_export       //     sw_wire.export
  );

  wire         sdram_pll_c0_clk;
  wire  [31:0] nios2_gen2_0_data_master_readdata;
  wire         nios2_gen2_0_data_master_waitrequest;
  wire         nios2_gen2_0_data_master_debugaccess;
  wire  [27:0] nios2_gen2_0_data_master_address;
  wire   [3:0] nios2_gen2_0_data_master_byteenable;
  wire         nios2_gen2_0_data_master_read;
  wire         nios2_gen2_0_data_master_write;
  wire  [31:0] nios2_gen2_0_data_master_writedata;
  wire  [31:0] nios2_gen2_0_instruction_master_readdata;
  wire         nios2_gen2_0_instruction_master_waitrequest;
  wire  [27:0] nios2_gen2_0_instruction_master_address;
  wire         nios2_gen2_0_instruction_master_read;
  wire  [31:0] mm_interconnect_0_sysid_qsys_0_control_slave_readdata;
  wire   [0:0] mm_interconnect_0_sysid_qsys_0_control_slave_address;
  wire  [31:0] mm_interconnect_0_nios2_gen2_0_debug_mem_slave_readdata;
  wire         mm_interconnect_0_nios2_gen2_0_debug_mem_slave_waitrequest;
  wire         mm_interconnect_0_nios2_gen2_0_debug_mem_slave_debugaccess;
  wire   [8:0] mm_interconnect_0_nios2_gen2_0_debug_mem_slave_address;
  wire         mm_interconnect_0_nios2_gen2_0_debug_mem_slave_read;
  wire   [3:0] mm_interconnect_0_nios2_gen2_0_debug_mem_slave_byteenable;
  wire         mm_interconnect_0_nios2_gen2_0_debug_mem_slave_write;
  wire  [31:0] mm_interconnect_0_nios2_gen2_0_debug_mem_slave_writedata;
  wire  [31:0] mm_interconnect_0_sdram_pll_pll_slave_readdata;
  wire   [1:0] mm_interconnect_0_sdram_pll_pll_slave_address;
  wire         mm_interconnect_0_sdram_pll_pll_slave_read;
  wire         mm_interconnect_0_sdram_pll_pll_slave_write;
  wire  [31:0] mm_interconnect_0_sdram_pll_pll_slave_writedata;
  wire         mm_interconnect_0_onchip_memory2_0_s1_chipselect;
  wire  [31:0] mm_interconnect_0_onchip_memory2_0_s1_readdata;
  wire   [1:0] mm_interconnect_0_onchip_memory2_0_s1_address;
  wire   [3:0] mm_interconnect_0_onchip_memory2_0_s1_byteenable;
  wire         mm_interconnect_0_onchip_memory2_0_s1_write;
  wire  [31:0] mm_interconnect_0_onchip_memory2_0_s1_writedata;
  wire         mm_interconnect_0_onchip_memory2_0_s1_clken;
  wire         mm_interconnect_0_led_s1_chipselect;
  wire  [31:0] mm_interconnect_0_led_s1_readdata;
  wire   [1:0] mm_interconnect_0_led_s1_address;
  wire         mm_interconnect_0_led_s1_write;
  wire  [31:0] mm_interconnect_0_led_s1_writedata;
  wire         mm_interconnect_0_sdram_s1_chipselect;
  wire  [15:0] mm_interconnect_0_sdram_s1_readdata;
  wire         mm_interconnect_0_sdram_s1_waitrequest;
  wire  [24:0] mm_interconnect_0_sdram_s1_address;
  wire         mm_interconnect_0_sdram_s1_read;
  wire   [1:0] mm_interconnect_0_sdram_s1_byteenable;
  wire         mm_interconnect_0_sdram_s1_readdatavalid;
  wire         mm_interconnect_0_sdram_s1_write;
  wire  [15:0] mm_interconnect_0_sdram_s1_writedata;
  wire  [31:0] mm_interconnect_0_switches_s1_readdata;
  wire   [1:0] mm_interconnect_0_switches_s1_address;
  wire  [31:0] mm_interconnect_0_push_buttons_s1_readdata;
  wire   [1:0] mm_interconnect_0_push_buttons_s1_address;
  wire  [31:0] nios2_gen2_0_irq_irq;
  wire         rst_controller_reset_out_reset;
  wire         rst_controller_reset_out_reset_req;
  wire         nios2_gen2_0_debug_reset_request_reset;
  wire         rst_controller_001_reset_out_reset;
```

```
lab7_soc_led led (
   .clk        (clk_clk),                            //               clk.clk
   .reset_n    (~rst_controller_reset_out_reset),    //             reset.reset_n
   .address    (mm_interconnect_0_led_s1_address),   //                s1.address
   .write_n    (~mm_interconnect_0_led_s1_write),    //                  .write_n
   .writedata  (mm_interconnect_0_led_s1_writedata), //                  .writedata
   .chipselect (mm_interconnect_0_led_s1_chipselect), //                  .chipselect
   .readdata   (mm_interconnect_0_led_s1_readdata),  //                  .readdata
   .out_port   (led_wire_export)                     // external_connection.export
);

lab7_soc_nios2_gen2_0 nios2_gen2_0 (
   .clk                                (clk_clk),                                                        //                     clk.clk
   .reset_n                            (~rst_controller_reset_out_reset),                                //                   reset.reset_n
   .reset_req                          (rst_controller_reset_out_reset_req),                             //                         .reset_req
   .d_address                          (nios2_gen2_0_data_master_address),                               //               data_master.address
   .d_byteenable                       (nios2_gen2_0_data_master_byteenable),                            //                         .byteenable
   .d_read                             (nios2_gen2_0_data_master_read),                                  //                         .read
   .d_readdata                         (nios2_gen2_0_data_master_readdata),                              //                         .readdata
   .d_waitrequest                      (nios2_gen2_0_data_master_waitrequest),                           //                         .waitrequest
   .d_write                            (nios2_gen2_0_data_master_write),                                 //                         .write
   .d_writedata                        (nios2_gen2_0_data_master_writedata),                             //                         .writedata
   .debug_mem_slave_debugaccess_to_roms (nios2_gen2_0_data_master_debugaccess),                          //                         .debugaccess
   .i_address                          (nios2_gen2_0_instruction_master_address),                        //        instruction_master.address
   .i_read                             (nios2_gen2_0_instruction_master_read),                           //                         .read
   .i_readdata                         (nios2_gen2_0_instruction_master_readdata),                       //                         .readdata
   .i_waitrequest                      (nios2_gen2_0_instruction_master_waitrequest),                    //                         .waitrequest
   .irq                                (nios2_gen2_0_irq_irq),                                           //                     irq.irq
   .debug_reset_request                (nios2_gen2_0_debug_reset_request_reset),                         //       debug_reset_request.reset
   .debug_mem_slave_address            (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_address),         //           debug_mem_slave.address
   .debug_mem_slave_byteenable         (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_byteenable),      //                         .byteenable
   .debug_mem_slave_debugaccess        (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_debugaccess),     //                         .debugaccess
   .debug_mem_slave_read               (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_read),            //                         .read
   .debug_mem_slave_readdata           (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_readdata),        //                         .readdata
   .debug_mem_slave_waitrequest        (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_waitrequest),     //                         .waitrequest
   .debug_mem_slave_write              (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_write),           //                         .write
   .debug_mem_slave_writedata          (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_writedata),       //                         .writedata
   .dummy_ci_port                      ()                                                                // custom_instruction_master.readra
);


lab7_soc_onchip_memory2_0 onchip_memory2_0 (
   .clk        (clk_clk),                            //   clk1.clk
   .address    (mm_interconnect_0_onchip_memory2_0_s1_address),    //     s1.address
   .clken      (mm_interconnect_0_onchip_memory2_0_s1_clken),      //       .clken
   .chipselect (mm_interconnect_0_onchip_memory2_0_s1_chipselect), //       .chipselect
   .write      (mm_interconnect_0_onchip_memory2_0_s1_write),      //       .write
   .readdata   (mm_interconnect_0_onchip_memory2_0_s1_readdata),   //       .readdata
   .writedata  (mm_interconnect_0_onchip_memory2_0_s1_writedata),  //       .writedata
   .byteenable (mm_interconnect_0_onchip_memory2_0_s1_byteenable), //       .byteenable
   .reset      (rst_controller_reset_out_reset),     // reset1.reset
   .reset_req  (rst_controller_reset_out_reset_req), //       .reset_req
   .freeze     (1'b0)                                // (terminated)
);

lab7_soc_push_buttons push_buttons (
   .clk      (clk_clk),                            //               clk.clk
   .reset_n  (~rst_controller_reset_out_reset),    //             reset.reset_n
   .address  (mm_interconnect_0_push_buttons_s1_address),  //                s1.address
   .readdata (mm_interconnect_0_push_buttons_s1_readdata), //                  .readdata
   .in_port  (buttons_wire_export)                 // external_connection.export
);

lab7_soc_sdram sdram (
   .clk            (sdram_pll_c0_clk),                       //   clk.clk
   .reset_n        (~rst_controller_001_reset_out_reset),    // reset.reset_n
   .az_addr        (mm_interconnect_0_sdram_s1_address),     //    s1.address
   .az_be_n        (~mm_interconnect_0_sdram_s1_byteenable), //      .byteenable_n
   .az_cs          (mm_interconnect_0_sdram_s1_chipselect),  //      .chipselect
   .az_data        (mm_interconnect_0_sdram_s1_writedata),   //      .writedata
   .az_rd_n        (~mm_interconnect_0_sdram_s1_read),       //      .read_n
   .az_wr_n        (~mm_interconnect_0_sdram_s1_write),      //      .write_n
   .za_data        (mm_interconnect_0_sdram_s1_readdata),    //      .readdata
   .za_valid       (mm_interconnect_0_sdram_s1_readdatavalid), //      .readdatavalid
   .za_waitrequest (mm_interconnect_0_sdram_s1_waitrequest), //      .waitrequest
   .zs_addr        (sdram_wire_addr),                        //  wire.export
   .zs_ba          (sdram_wire_ba),                          //      .export
   .zs_cas_n       (sdram_wire_cas_n),                       //      .export
   .zs_cke         (sdram_wire_cke),                         //      .export
   .zs_cs_n        (sdram_wire_cs_n),                        //      .export
   .zs_dq          (sdram_wire_dq),                          //      .export
   .zs_dqm         (sdram_wire_dqm),                         //      .export
   .zs_ras_n       (sdram_wire_ras_n),                       //      .export
```

```
    .zs_we_n        (sdram_wire_we_n)                       //      .export
);

lab7_soc_sdram_pll sdram_pll (
    .clk                (clk_clk),                                      //       inclk_interface.clk
    .reset              (rst_controller_reset_out_reset),               // inclk_interface_reset.reset
    .read               (mm_interconnect_0_sdram_pll_pll_slave_read),   //              pll_slave.read
    .write              (mm_interconnect_0_sdram_pll_pll_slave_write),  //                       .write
    .address            (mm_interconnect_0_sdram_pll_pll_slave_address),//                       .address
    .readdata           (mm_interconnect_0_sdram_pll_pll_slave_readdata),//                      .readdata
    .writedata          (mm_interconnect_0_sdram_pll_pll_slave_writedata),//                     .writedata
    .c0                 (sdram_pll_c0_clk),                             //                     c0.clk
    .c1                 (sdram_clk_clk),                                //                     c1.clk
    .scandone           (),                                             //           (terminated)
    .scandataout        (),                                             //           (terminated)
    .c2                 (),                                             //           (terminated)
    .c3                 (),                                             //           (terminated)
    .c4                 (),                                             //           (terminated)
    .areset             (1'b0),                                         //           (terminated)
    .locked             (),                                             //           (terminated)
    .phasedone          (),                                             //           (terminated)
    .phasecounterselect (3'b000),                                       //           (terminated)
    .phaseupdown        (1'b0),                                         //           (terminated)
    .phasestep          (1'b0),                                         //           (terminated)
    .scanclk            (1'b0),                                         //           (terminated)
    .scanclkena         (1'b0),                                         //           (terminated)
    .scandata           (1'b0),                                         //           (terminated)
    .configupdate       (1'b0)                                          //           (terminated)
);

lab7_soc_switches switches (
    .clk      (clk_clk),                                //               clk.clk
    .reset_n  (~rst_controller_reset_out_reset),        //             reset.reset_n
    .address  (mm_interconnect_0_switches_s1_address),  //                s1.address
    .readdata (mm_interconnect_0_switches_s1_readdata), //                  .readdata
    .in_port  (sw_wire_export)                          // external_connection.export
);

lab7_soc_sysid_qsys_0 sysid_qsys_0 (
    .clock    (clk_clk),                                //           clk.clk
    .reset_n  (~rst_controller_reset_out_reset),        //         reset.reset_n
    .readdata (mm_interconnect_0_sysid_qsys_0_control_slave_readdata), // control_slave.readdata

    .address  (mm_interconnect_0_sysid_qsys_0_control_slave_address)   //              .address
);

lab7_soc_mm_interconnect_0 mm_interconnect_0 (
    .clk_0_clk_clk                                  (clk_clk),                                                  //                                clk_0_clk.clk
    .sdram_pll_c0_clk                               (sdram_pll_c0_clk),                                         //                              sdram_pll_c0.clk
    .nios2_gen2_0_reset_reset_bridge_in_reset_reset (rst_controller_reset_out_reset),                           // nios2_gen2_0_reset_reset_bridge_in_reset.reset
    .sdram_reset_reset_bridge_in_reset_reset        (rst_controller_001_reset_out_reset),                       //       sdram_reset_reset_bridge_in_reset.reset
    .nios2_gen2_0_data_master_address               (nios2_gen2_0_data_master_address),                         //                 nios2_gen2_0_data_master.address
    .nios2_gen2_0_data_master_waitrequest           (nios2_gen2_0_data_master_waitrequest),                     //                                         .waitrequest
    .nios2_gen2_0_data_master_byteenable            (nios2_gen2_0_data_master_byteenable),                      //                                         .byteenable
    .nios2_gen2_0_data_master_read                  (nios2_gen2_0_data_master_read),                            //                                         .read
    .nios2_gen2_0_data_master_readdata              (nios2_gen2_0_data_master_readdata),                        //                                         .readdata
    .nios2_gen2_0_data_master_write                 (nios2_gen2_0_data_master_write),                           //                                         .write
    .nios2_gen2_0_data_master_writedata             (nios2_gen2_0_data_master_writedata),                       //                                         .writedata
    .nios2_gen2_0_data_master_debugaccess           (nios2_gen2_0_data_master_debugaccess),                     //                                         .debugaccess
    .nios2_gen2_0_instruction_master_address        (nios2_gen2_0_instruction_master_address),                  //          nios2_gen2_0_instruction_master.address
    .nios2_gen2_0_instruction_master_waitrequest    (nios2_gen2_0_instruction_master_waitrequest),              //                                         .waitrequest
    .nios2_gen2_0_instruction_master_read           (nios2_gen2_0_instruction_master_read),                     //                                         .read
    .nios2_gen2_0_instruction_master_readdata       (nios2_gen2_0_instruction_master_readdata),                 //                                         .readdata
    .led_s1_address                                 (mm_interconnect_0_led_s1_address),                         //                                led_s1.address
    .led_s1_write                                   (mm_interconnect_0_led_s1_write),                           //                                      .write
    .led_s1_readdata                                (mm_interconnect_0_led_s1_readdata),                        //                                      .readdata
    .led_s1_writedata                               (mm_interconnect_0_led_s1_writedata),                       //                                      .writedata
    .led_s1_chipselect                              (mm_interconnect_0_led_s1_chipselect),                      //                                      .chipselect
    .nios2_gen2_0_debug_mem_slave_address           (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_address),   //            nios2_gen2_0_debug_mem_slave.address
    .nios2_gen2_0_debug_mem_slave_write             (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_write),     //                                         .write
    .nios2_gen2_0_debug_mem_slave_read              (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_read),      //                                         .read
    .nios2_gen2_0_debug_mem_slave_readdata          (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_readdata),  //                                         .readdata
    .nios2_gen2_0_debug_mem_slave_writedata         (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_writedata), //                                         .writedata
    .nios2_gen2_0_debug_mem_slave_byteenable        (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_byteenable),//                                         .byteenable
    .nios2_gen2_0_debug_mem_slave_waitrequest       (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_waitrequest),//                                        .waitrequest
    .nios2_gen2_0_debug_mem_slave_debugaccess       (mm_interconnect_0_nios2_gen2_0_debug_mem_slave_debugaccess),//                                        .debugaccess
    .onchip_memory2_0_s1_address                    (mm_interconnect_0_onchip_memory2_0_s1_address),            //                  onchip_memory2_0_s1.address
    .onchip_memory2_0_s1_write                      (mm_interconnect_0_onchip_memory2_0_s1_write),              //                                      .write
    .onchip_memory2_0_s1_readdata                   (mm_interconnect_0_onchip_memory2_0_s1_readdata),           //                                      .readdata
    .onchip_memory2_0_s1_writedata                  (mm_interconnect_0_onchip_memory2_0_s1_writedata),          //                                      .writedata
    .onchip_memory2_0_s1_byteenable                 (mm_interconnect_0_onchip_memory2_0_s1_byteenable),         //                                      .byteenable
    .onchip_memory2_0_s1_chipselect                 (mm_interconnect_0_onchip_memory2_0_s1_chipselect),         //                                      .chipselect
    .onchip_memory2_0_s1_clken                      (mm_interconnect_0_onchip_memory2_0_s1_clken),              //                                      .clken
    .push_buttons_s1_address                        (mm_interconnect_0_push_buttons_s1_address),                //                    push_buttons_s1.address

    .push_buttons_s1_readdata                       (mm_interconnect_0_push_buttons_s1_readdata),               //                                      .readdata
    .sdram_s1_address                               (mm_interconnect_0_sdram_s1_address),                       //                           sdram_s1.address
    .sdram_s1_write                                 (mm_interconnect_0_sdram_s1_write),                         //                                   .write
    .sdram_s1_read                                  (mm_interconnect_0_sdram_s1_read),                          //                                   .read
    .sdram_s1_readdata                              (mm_interconnect_0_sdram_s1_readdata),                      //                                   .readdata
    .sdram_s1_writedata                             (mm_interconnect_0_sdram_s1_writedata),                     //                                   .writedata
    .sdram_s1_byteenable                            (mm_interconnect_0_sdram_s1_byteenable),                    //                                   .byteenable
    .sdram_s1_readdatavalid                         (mm_interconnect_0_sdram_s1_readdatavalid),                 //                                   .readdatavalid
    .sdram_s1_waitrequest                           (mm_interconnect_0_sdram_s1_waitrequest),                   //                                   .waitrequest
    .sdram_s1_chipselect                            (mm_interconnect_0_sdram_s1_chipselect),                    //                                   .chipselect
    .sdram_pll_pll_slave_address                    (mm_interconnect_0_sdram_pll_pll_slave_address),            //                 sdram_pll_pll_slave.address
    .sdram_pll_pll_slave_write                      (mm_interconnect_0_sdram_pll_pll_slave_write),              //                                   .write
    .sdram_pll_pll_slave_read                       (mm_interconnect_0_sdram_pll_pll_slave_read),               //                                   .read
    .sdram_pll_pll_slave_readdata                   (mm_interconnect_0_sdram_pll_pll_slave_readdata),           //                                   .readdata
    .sdram_pll_pll_slave_writedata                  (mm_interconnect_0_sdram_pll_pll_slave_writedata),          //                                   .writedata
    .switches_s1_address                            (mm_interconnect_0_switches_s1_address),                    //                       switches_s1.address
    .switches_s1_readdata                           (mm_interconnect_0_switches_s1_readdata),                   //                                   .readdata
    .sysid_qsys_0_control_slave_address             (mm_interconnect_0_sysid_qsys_0_control_slave_address),     //           sysid_qsys_0_control_slave.address
    .sysid_qsys_0_control_slave_readdata            (mm_interconnect_0_sysid_qsys_0_control_slave_readdata)     //                                   .readdata
);

lab7_soc_irq_mapper irq_mapper (
    .clk        (clk_clk),                          //       clk.clk
    .reset      (rst_controller_reset_out_reset), // clk_reset.reset
    .sender_irq (nios2_gen2_0_irq_irq)              //    sender.irq
);

altera_reset_controller #(
    .NUM_RESET_INPUTS          (2),
    .OUTPUT_RESET_SYNC_EDGES   ("deassert"),
    .SYNC_DEPTH                (2),
    .RESET_REQUEST_PRESENT     (1),
    .RESET_REQ_WAIT_TIME       (1),
    .MIN_RST_ASSERTION_TIME    (3),
    .RESET_REQ_EARLY_DSRT_TIME (1),
    .USE_RESET_REQUEST_IN0     (0),
    .USE_RESET_REQUEST_IN1     (0),
    .USE_RESET_REQUEST_IN2     (0),
    .USE_RESET_REQUEST_IN3     (0),
    .USE_RESET_REQUEST_IN4     (0),
    .USE_RESET_REQUEST_IN5     (0),
    .USE_RESET_REQUEST_IN6     (0),
    .USE_RESET_REQUEST_IN7     (0),
```

```
    .USE_RESET_REQUEST_IN8     (0),
    .USE_RESET_REQUEST_IN9     (0),
    .USE_RESET_REQUEST_IN10    (0),
    .USE_RESET_REQUEST_IN11    (0),
    .USE_RESET_REQUEST_IN12    (0),
    .USE_RESET_REQUEST_IN13    (0),
    .USE_RESET_REQUEST_IN14    (0),
    .USE_RESET_REQUEST_IN15    (0),
    .ADAPT_RESET_REQUEST       (0)
) rst_controller (
    .reset_in0      (~reset_reset_n),                       // reset_in0.reset
    .reset_in1      (nios2_gen2_0_debug_reset_request_reset), // reset_in1.reset
    .clk            (clk_clk),                              //       clk.clk
    .reset_out      (rst_controller_reset_out_reset),       // reset_out.reset
    .reset_req      (rst_controller_reset_out_reset_req),   //         .reset_req
    .reset_req_in0  (1'b0),                                 // (terminated)
    .reset_req_in1  (1'b0),                                 // (terminated)
    .reset_in2      (1'b0),                                 // (terminated)
    .reset_req_in2  (1'b0),                                 // (terminated)
    .reset_in3      (1'b0),                                 // (terminated)
    .reset_req_in3  (1'b0),                                 // (terminated)
    .reset_in4      (1'b0),                                 // (terminated)
    .reset_req_in4  (1'b0),                                 // (terminated)
    .reset_in5      (1'b0),                                 // (terminated)
    .reset_req_in5  (1'b0),                                 // (terminated)
    .reset_in6      (1'b0),                                 // (terminated)
    .reset_req_in6  (1'b0),                                 // (terminated)
    .reset_in7      (1'b0),                                 // (terminated)
    .reset_req_in7  (1'b0),                                 // (terminated)
    .reset_in8      (1'b0),                                 // (terminated)
    .reset_req_in8  (1'b0),                                 // (terminated)
    .reset_in9      (1'b0),                                 // (terminated)
    .reset_req_in9  (1'b0),                                 // (terminated)
    .reset_in10     (1'b0),                                 // (terminated)
    .reset_req_in10 (1'b0),                                 // (terminated)
    .reset_in11     (1'b0),                                 // (terminated)
    .reset_req_in11 (1'b0),                                 // (terminated)
    .reset_in12     (1'b0),                                 // (terminated)
    .reset_req_in12 (1'b0),                                 // (terminated)
    .reset_in13     (1'b0),                                 // (terminated)
    .reset_req_in13 (1'b0),                                 // (terminated)
    .reset_in14     (1'b0),                                 // (terminated)

    .reset_req_in14 (1'b0),                                 // (terminated)
    .reset_in15     (1'b0),                                 // (terminated)
    .reset_req_in15 (1'b0)                                  // (terminated)
);

altera_reset_controller #(
    .NUM_RESET_INPUTS          (2),
    .OUTPUT_RESET_SYNC_EDGES   ("deassert"),
    .SYNC_DEPTH                (2),
    .RESET_REQUEST_PRESENT     (0),
    .RESET_REQ_WAIT_TIME       (1),
    .MIN_RST_ASSERTION_TIME    (3),
    .RESET_REQ_EARLY_DSRT_TIME (1),
    .USE_RESET_REQUEST_IN0     (0),
    .USE_RESET_REQUEST_IN1     (0),
    .USE_RESET_REQUEST_IN2     (0),
    .USE_RESET_REQUEST_IN3     (0),
    .USE_RESET_REQUEST_IN4     (0),
    .USE_RESET_REQUEST_IN5     (0),
    .USE_RESET_REQUEST_IN6     (0),
    .USE_RESET_REQUEST_IN7     (0),
    .USE_RESET_REQUEST_IN8     (0),
    .USE_RESET_REQUEST_IN9     (0),
    .USE_RESET_REQUEST_IN10    (0),
    .USE_RESET_REQUEST_IN11    (0),
    .USE_RESET_REQUEST_IN12    (0),
    .USE_RESET_REQUEST_IN13    (0),
    .USE_RESET_REQUEST_IN14    (0),
    .USE_RESET_REQUEST_IN15    (0),
    .ADAPT_RESET_REQUEST       (0)
) rst_controller_001 (
    .reset_in0      (~reset_reset_n),                       // reset_in0.reset
    .reset_in1      (nios2_gen2_0_debug_reset_request_reset), // reset_in1.reset
    .clk            (sdram_pll_c0_clk),                     //       clk.clk
    .reset_out      (rst_controller_001_reset_out_reset),   // reset_out.reset
    .reset_req      (),                                     // (terminated)
    .reset_req_in0  (1'b0),                                 // (terminated)
    .reset_req_in1  (1'b0),                                 // (terminated)
    .reset_in2      (1'b0),                                 // (terminated)
    .reset_req_in2  (1'b0),                                 // (terminated)
    .reset_in3      (1'b0),                                 // (terminated)
    .reset_req_in3  (1'b0),                                 // (terminated)

    .reset_in4      (1'b0),                                 // (terminated)
    .reset_req_in4  (1'b0),                                 // (terminated)
    .reset_in5      (1'b0),                                 // (terminated)
    .reset_req_in5  (1'b0),                                 // (terminated)
    .reset_in6      (1'b0),                                 // (terminated)
    .reset_req_in6  (1'b0),                                 // (terminated)
    .reset_in7      (1'b0),                                 // (terminated)
    .reset_req_in7  (1'b0),                                 // (terminated)
    .reset_in8      (1'b0),                                 // (terminated)
    .reset_req_in8  (1'b0),                                 // (terminated)
    .reset_in9      (1'b0),                                 // (terminated)
    .reset_req_in9  (1'b0),                                 // (terminated)
    .reset_in10     (1'b0),                                 // (terminated)
    .reset_req_in10 (1'b0),                                 // (terminated)
    .reset_in11     (1'b0),                                 // (terminated)
    .reset_req_in11 (1'b0),                                 // (terminated)
    .reset_in12     (1'b0),                                 // (terminated)
    .reset_req_in12 (1'b0),                                 // (terminated)
    .reset_in13     (1'b0),                                 // (terminated)
    .reset_req_in13 (1'b0),                                 // (terminated)
    .reset_in14     (1'b0),                                 // (terminated)
    .reset_req_in14 (1'b0),                                 // (terminated)
    .reset_in15     (1'b0),                                 // (terminated)
    .reset_req_in15 (1'b0)                                  // (terminated)
);

endmodule
```

Module: lab7_soc.v

Inputs: clk_clk, sw_wire_export, buttons_wire_export

Outputs: LED [7:0], SDRAM_ADDR[12:0], SDRAM_CAS_N, SDRAM_CKE, SDRAM_CS_N, SDRAM_DQM[1:0], SDRAM_RAS_N, SDRAM_WE_N, SDRAM_CLK

Description: This is the lab7_soc file which is generated by the platform designer that consists of all the modules discussed in the platform designer setup

Purpose: The purpose is that it provides a datapath in verilog to all the different components and provides the appropriate outputs to have a correct functioning SDRAM with data transferred to the LED etc. The inputs are the clock, switches and the buttons.

```
#**************************************************************
# Create Clock
#**************************************************************
create_clock -period "10.0 MHz" [get_ports ADC_CLK_10]
create_clock -period "50.0 MHz" [get_ports CLOCK_50]
create_clock -period "50.0 MHz" [get_ports MAX10_CLK2_50]



# SDRAM CLK
#create_generated_clock -source [get_nets { lab_7_soc_sdram_pll_altpll_vg92:sd1:wire_pll7_clk[1] }] \
                       -name sdram_clk
                       #[get_ports {lab_7_soc_sdram_pll_altpll_vg92:sd1:wire_pll7_clk[1]}]

create_generated_clock -name sdram_clk [get_ports {DRAM_CLK}] -source [get_nets {m_lab7_soc|sdram_pll|sd1|wire_pll7_clk[1]}]

#lab_7_soc_sdram_pll_altpll_vg92:sd1:wire_pll7_clk[1]
#**************************************************************
# Create Generated Clock
#**************************************************************
derive_pll_clocks


#**************************************************************
# Set Clock Latency
#**************************************************************


#**************************************************************
# Set Clock Uncertainty
#**************************************************************
derive_clock_uncertainty


#**************************************************************
# Set Input Delay
#**************************************************************
# suppose +- 100 ps skew
# Board Delay (Data) + Propagation Delay - Board Delay (Clock)
# max 5.4(max) +0.4(trace delay) +0.1 = 5.9
# min 2.7(min) +0.4(trace delay) -0.1 = 3.0
set_input_delay -max -clock sdram_clk 5.9 [get_ports DRAM_DQ*]
set_input_delay -min -clock sdram_clk 3.0 [get_ports DRAM_DQ*]
#set_input_delay -max -clock sdram_clk 5.9 [get_ports DRAM_DQM*]
#set_input_delay -min -clock sdram_clk 3.0 [get_ports DRAM_DQM*]



#shift-window
set_multicycle_path -from [get_clocks {sdram_clk}] -to [get_clocks { m_lab7_soc|sdram_pll|sd1|pll7|clk[0] }] -setup 2

#**************************************************************
# Set Output Delay
#**************************************************************
# suppose +- 100 ps skew
# max : Board Delay (Data) - Board Delay (Clock) + tsu (External Device)
# min : Board Delay (Data) - Board Delay (Clock) - th (External Device)
# max 1.5+0.1 =1.6
# min -0.8-0.1 = 0.9
set_output_delay -max -clock sdram_clk 1.6  [get_ports {DRAM_DQ* DRAM_DQM*}]
set_output_delay -min -clock sdram_clk -0.9 [get_ports {DRAM_DQ* DRAM_DQM*}]
set_output_delay -max -clock sdram_clk 1.6  [get_ports {DRAM_ADDR* DRAM_BA* DRAM_RAS_N DRAM_CAS_N DRAM_WE_N DRAM_CKE DRAM_CS_N}]
set_output_delay -min -clock sdram_clk -0.9 [get_ports {DRAM_ADDR* DRAM_BA* DRAM_RAS_N DRAM_CAS_N DRAM_WE_N DRAM_CKE DRAM_CS_N}]


#**************************************************************
# Set Clock Groups
#**************************************************************
```

```
#**************************************************************
# Set False Path
#**************************************************************
set_false_path -from * -to [get_ports LED*]
set_false_path -from [get_ports SW*] -to *
set_false_path -from [get_ports KEY*] -to *
set_false_path -from [get_ports altera_reserved_tdi] -to *
set_false_path -from [get_ports altera_reserved_tms] -to *
set_false_path -from * -to [get_ports altera_reserved_tdo]


#**************************************************************
# Set Multicycle Path
#**************************************************************



#**************************************************************
# Set Maximum Delay
#**************************************************************



#**************************************************************
# Set Minimum Delay
#**************************************************************



#**************************************************************
# Set Input Transition
#**************************************************************



#**************************************************************
# Set Load
#**************************************************************
```

Module: lab7.sdc

Purpose: The SDC file is to constrain the clocks so we can make sure that the clock is running at a constrained frequency, we can make sure to set specific delays to certain inputs and outputs and also exclude the LEDs, SWs and buttons from our timing analysis. It helps to create a multicycle path which is important.