ECE 385

Fall 2020

Experiment #7

Simple Computer SLC-3.2 in System Verilog

Name: Xiao Shuhong & Lu Yicheng
Lab Section: D225

1. Introduction

1.1. Summarize the basic functionality of the NIOS-II processor

NIOS-II processor is an IP based 32-bit CPU which is not real but constructed by FPGA(having lots of LE etc.). We can run c program on this processor to handle tasks that do not need high performance. For example, user interface, data input and output. In this lab, we will use this processor as a system controller to control the LED.

2. Written Description and Diagrams of NIOS-II System

2.1 Describe in words the hardware component of the lab

For nios2_gen2_0: it is an IP based 32-bit CPU which is used to run c program.

For,onchip_memory2_0: it is a Writable RAM with total memory size to be 16 bytes. We do not use on chip memory in this lab. However, if we store program in on chip memory, the execution will be faster.

For led, it is a parallel output IO with width 8. With this IO, we can assign base address to it by mapping, so that c program can control the LED.

For SDRAM, it is a controller that can send data and control signal to real SRAM. This controller is a bridge between CPU and SRAM. It also deals with the mapping job.

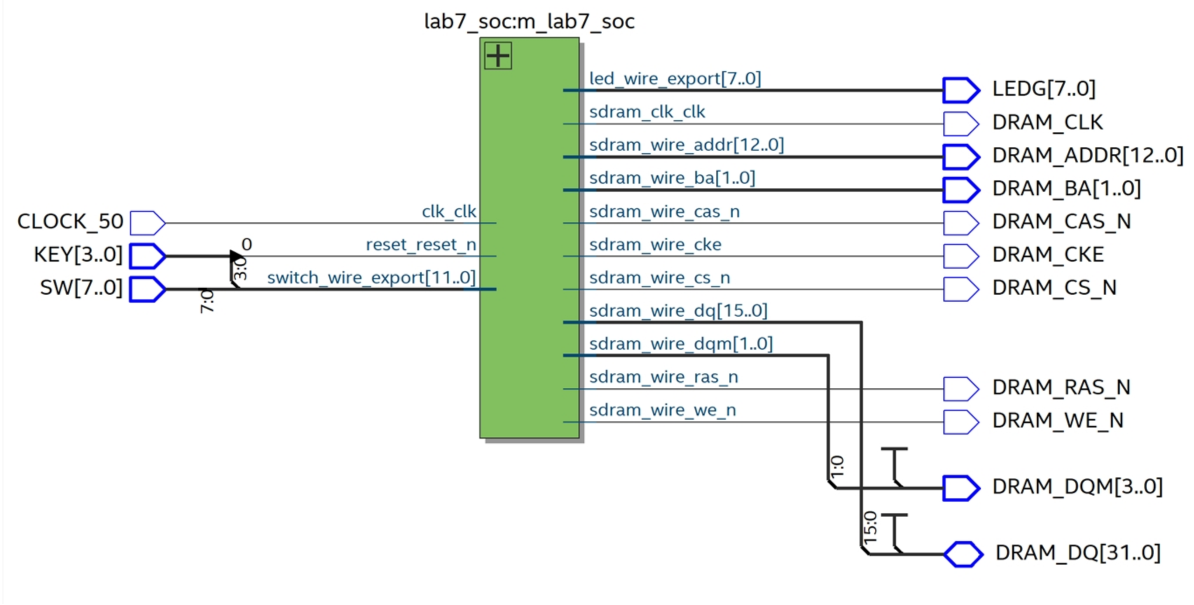
For sdram_pll, it provides the required clock signal for the SDRAM chip. Since the distance between CPU and SDRAM is quit long, we have to make sure they have the same clock. So, pll will send a modified clock signal that is 3ns ahead of the CPU clock. So that when this modified clock arrives SDRAM, it is same as CPU clock.

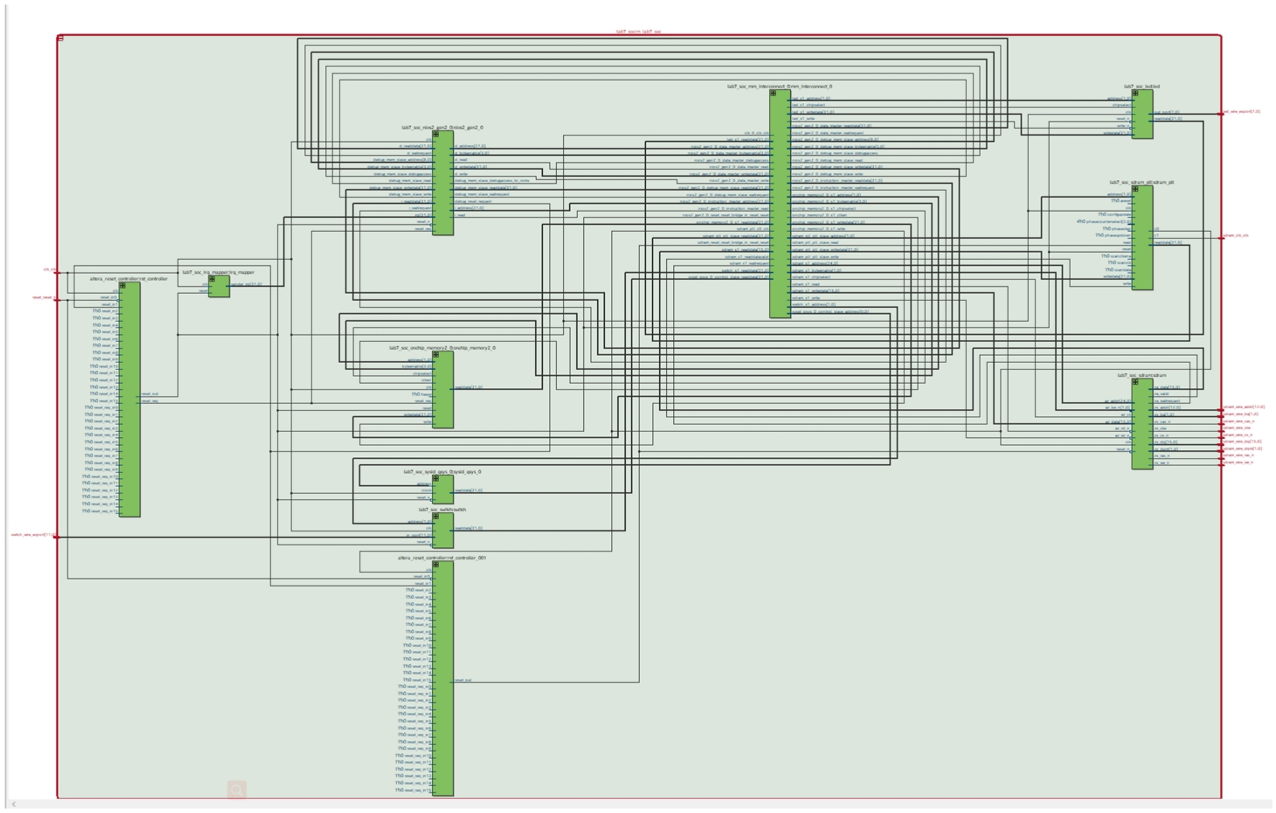
For sysid_qsys_0,is is an ID checker to ensure the compatibility between hardware and software. In other word, it prevents us from loading software onto an FPGA which has an incompatible NIOS II configuration.

For switch, it is a parallel input IO with width 12. It will receive both KEY and SW. it has form: {KEY,SW}. With this PIO, c program can receive the KEY and SW by visiting the corresponding address.

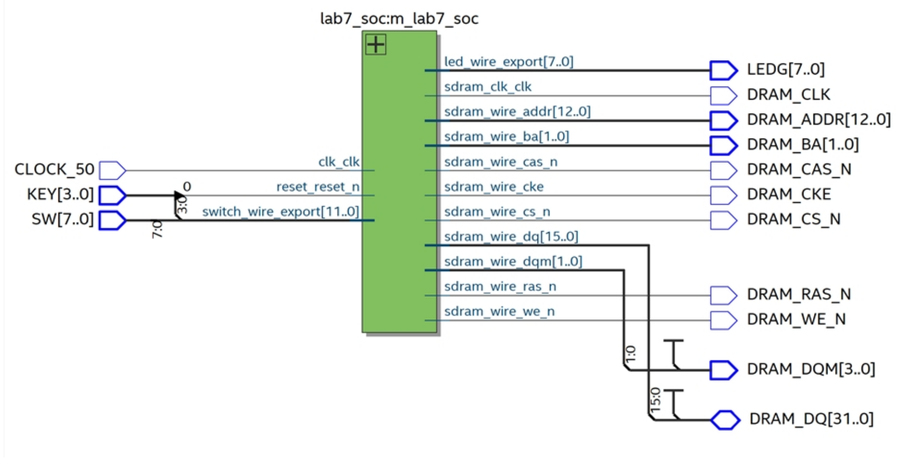
For conclusion, our “switch” pio will receive the control signal key, and the data SW. Then cpu will read these by visiting the corresponding mapped address. So that CPU can get key can SW Base on key and SW, CPU will finally change the value that is in “led” pio mapped address.

3,Top level Block Diagram





4, Written Description of all .sv Modules



Module:lab7.sv

Inputs:

```
        input           CLOCK_50,
        input   [3:0]   KEY,
        input  [7:0]    SW
```

Outputs:

```
        output [7:0]        LEDG,
        output [12:0]       DRAM_ADDR,
        output [1:0]        DRAM_BA,
        output              DRAM_CAS_N,
        output              DRAM_CKE,
        output              DRAM_CS_N,
        output  [3:0]       DRAM_DQM,
        output              DRAM_RAS_N,
        output              DRAM_WE_N,
        output              DRAM_CLK,
```

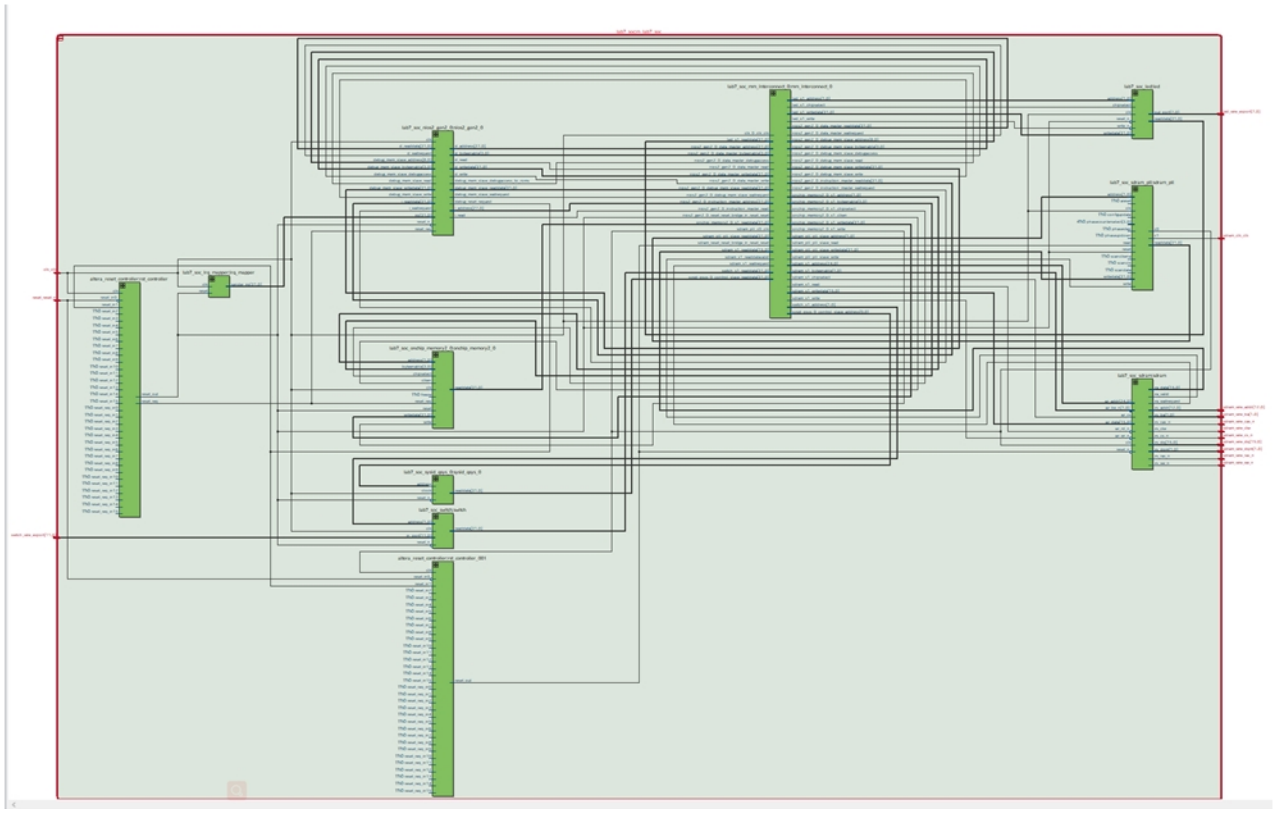
Inouts:

```
       inout  [31:0]        DRAM_DQ,
```

Description: This is the top-level. We Instantiate things that we construct in the platform designer.

Purpose: Instantiate the whole hardware system and give the pin name so that we can connect with key and switch.

Module: lab7_soc.v



Inputs:

```
input  wire        clk_clk,
input  wire        reset_reset_n,
input  wire [11:0] switch_wire_export
```

Outputs:

```
output wire [7:0]  led_wire_export,
output wire        sdram_clk_clk,
output wire [12:0] sdram_wire_addr,
output wire [1:0]  sdram_wire_ba,
output wire        sdram_wire_cas_n,
output wire        sdram_wire_cke,
output wire        sdram_wire_cs_n,
output wire [1:0]  sdram_wire_dqm,
output wire        sdram_wire_ras_n,
output wire        sdram_wire_we_n,
```

Inout:

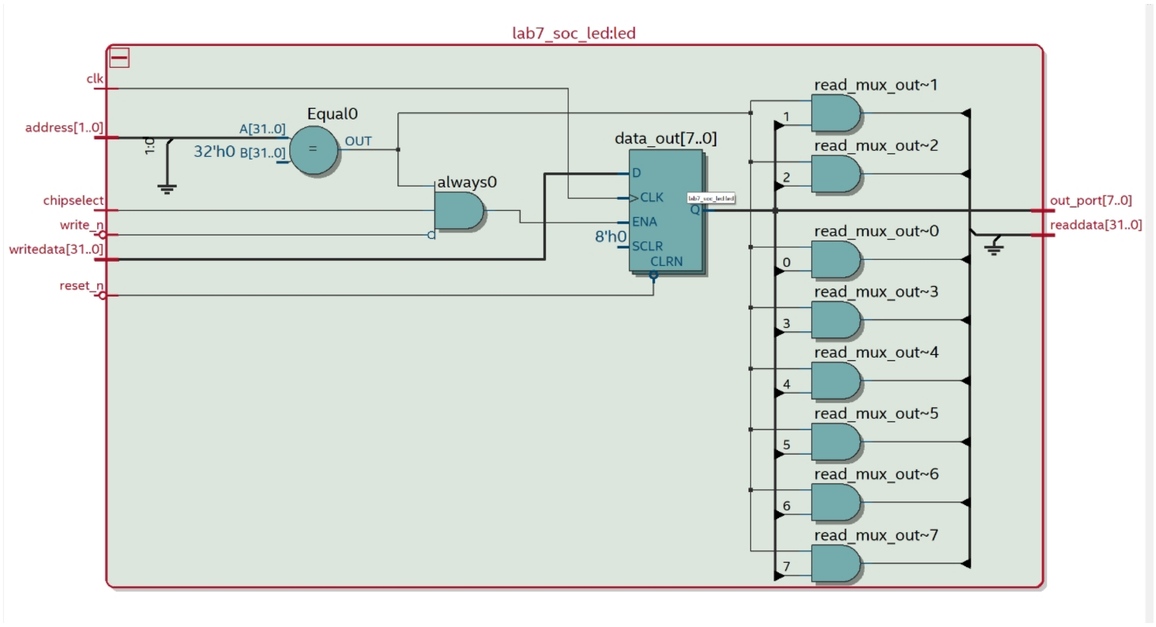
```
inout  wire [15:0] sdram_wire_dq,
```

Description: This module contains processor, memory, IO. So, it acts as a tiny computer. It will fetch the instruction from memory, decode it and finally execute the instruction. To be specific, this module can just read and write data to the expected addresses and these addresses are actually the external devices. In this way, we can read and write to the external devices.

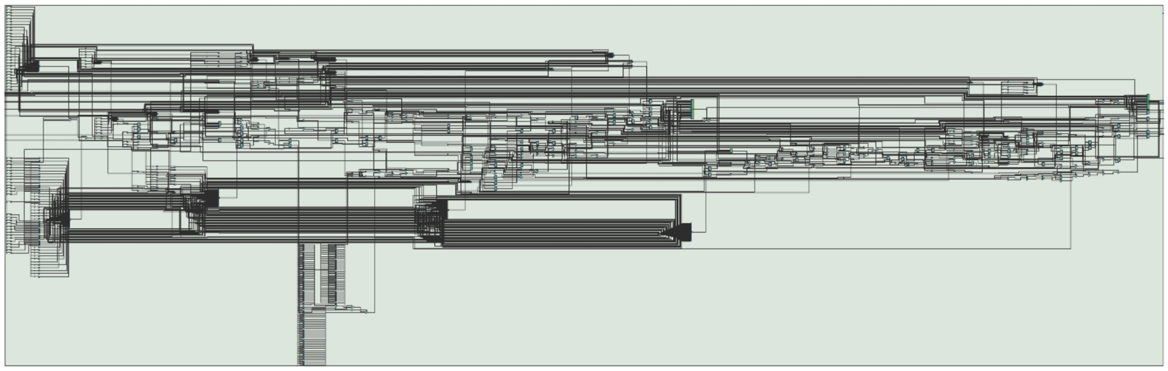
Purpose: With this system, our c program can run on it and can control LED, receive signal by visiting the corresponding PIO address.

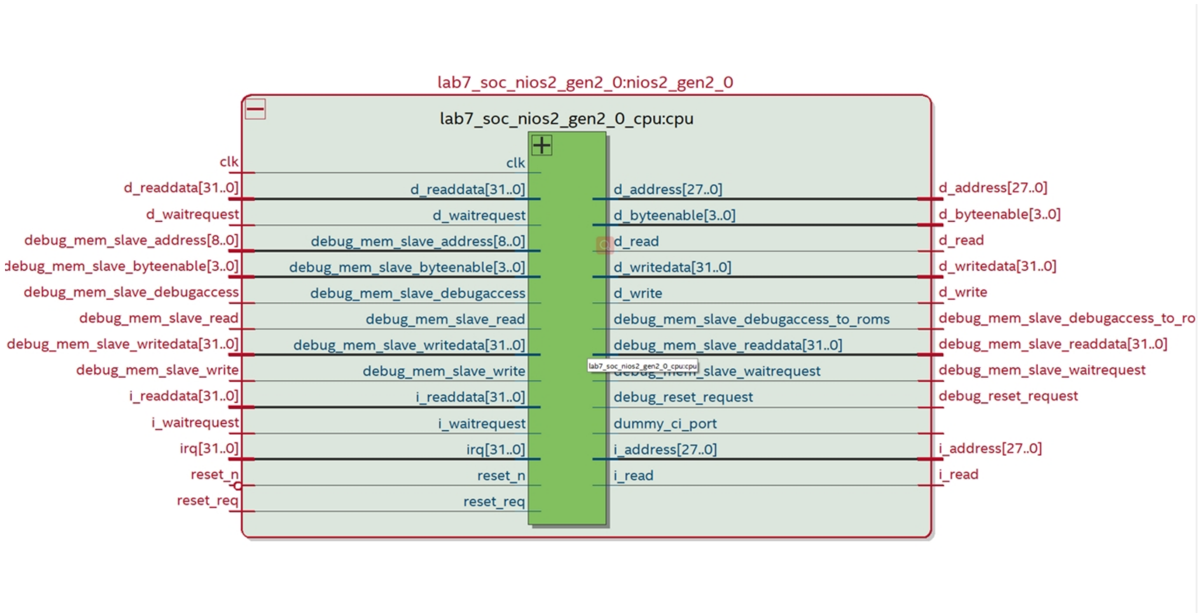
5,System Level Block Diagram

5.1 functionality of each block.

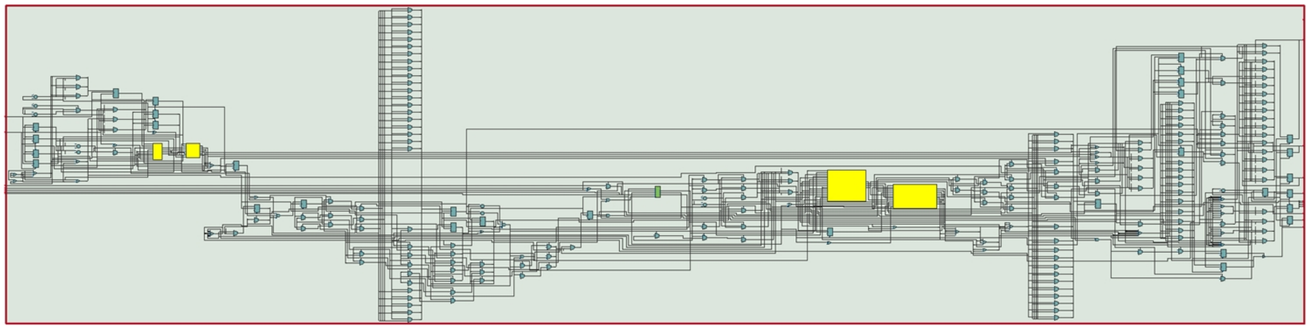


For led, it is a parallel output IO with width 8. With this IO, we can assign base address to it by mapping, so that c program can control the LED by read or write the address.

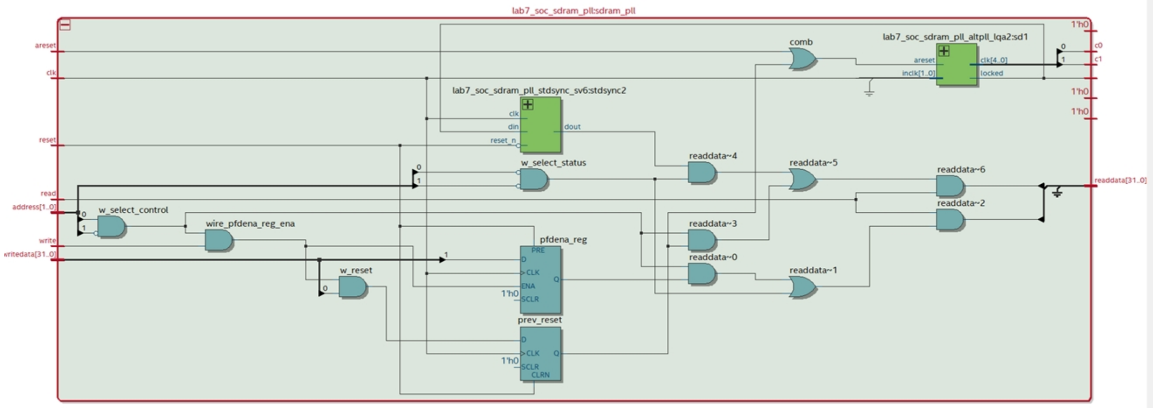



lab7_soc_nios2_gen2_0:nios2_gen2_0
lab7_soc_nios2_gen2_0_cpu:cpu
clk
d_readdata[31..0]
d_waitrequest
debug_mem_slave_address[8..0]
debug_mem_slave_byteenable[3..0]
debug_mem_slave_debugaccess
debug_mem_slave_read
debug_mem_slave_writedata[31..0]
debug_mem_slave_write
i_readdata[31..0]
i_waitrequest
irq[31..0]
reset_n
reset_req
d_address[27..0]
d_byteenable[3..0]
d_read
d_writedata[31..0]
d_write
debug_mem_slave_debugaccess_to_roms
debug_mem_slave_readdata[31..0]
debug_mem_slave_waitrequest
debug_reset_request
dummy_ci_port
i_address[27..0]
i_read

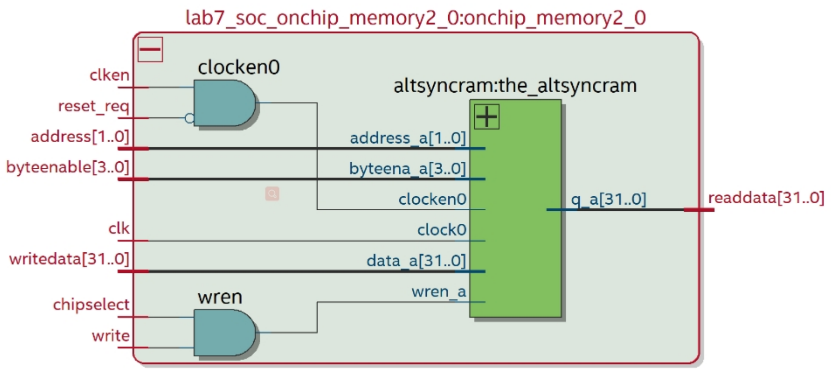
For nios2_gen2_0: it is an IP based 32-bit CPU which can handle tasks that do not need high performance. For example, user interface, data input and output. In this lab, it is used to run c program and receive the key, sw signal and control the led.



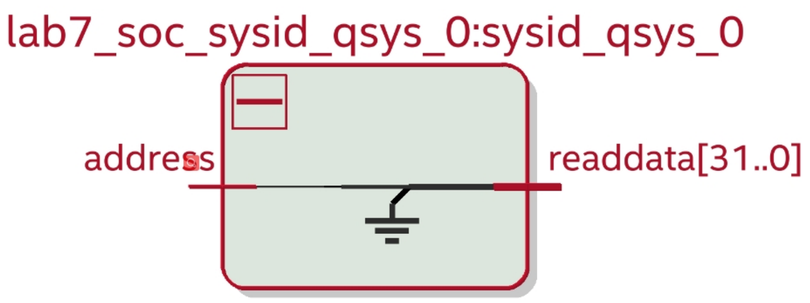
For SDRAM, it is a controller that can send data and control signal to real SRAM. This controller is a bridge between CPU and SRAM. It also deals with the mapping job.



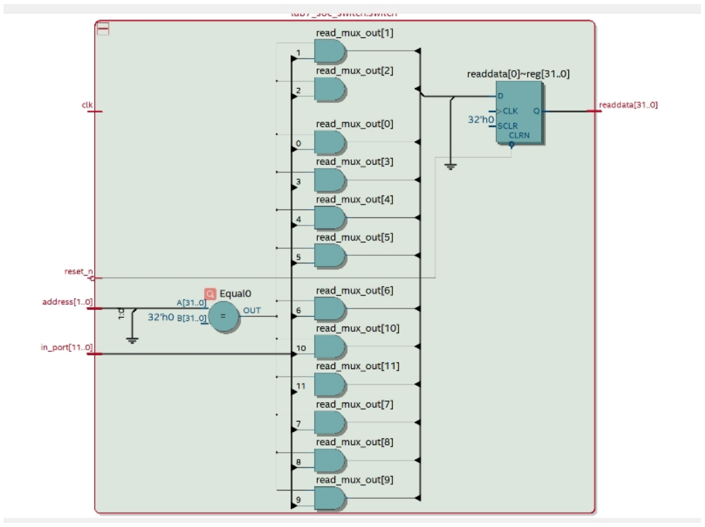
For sdram_pll, it provides the required clock signal for the SDRAM chip. Since the distance between CPU and SDRAM is quit long, we have to make sure they have the same clock. So, pll will send a modified clock signal that is 3ns ahead of the CPU clock. So that when this modified clock arrives SDRAM, it is same as CPU clock



For,onchip_memory2_0: it is a Writable RAM with total memory size to be 16 bytes. We do not use on chip memory in this lab. However, if we store program in on chip memory, the execution will be faster.

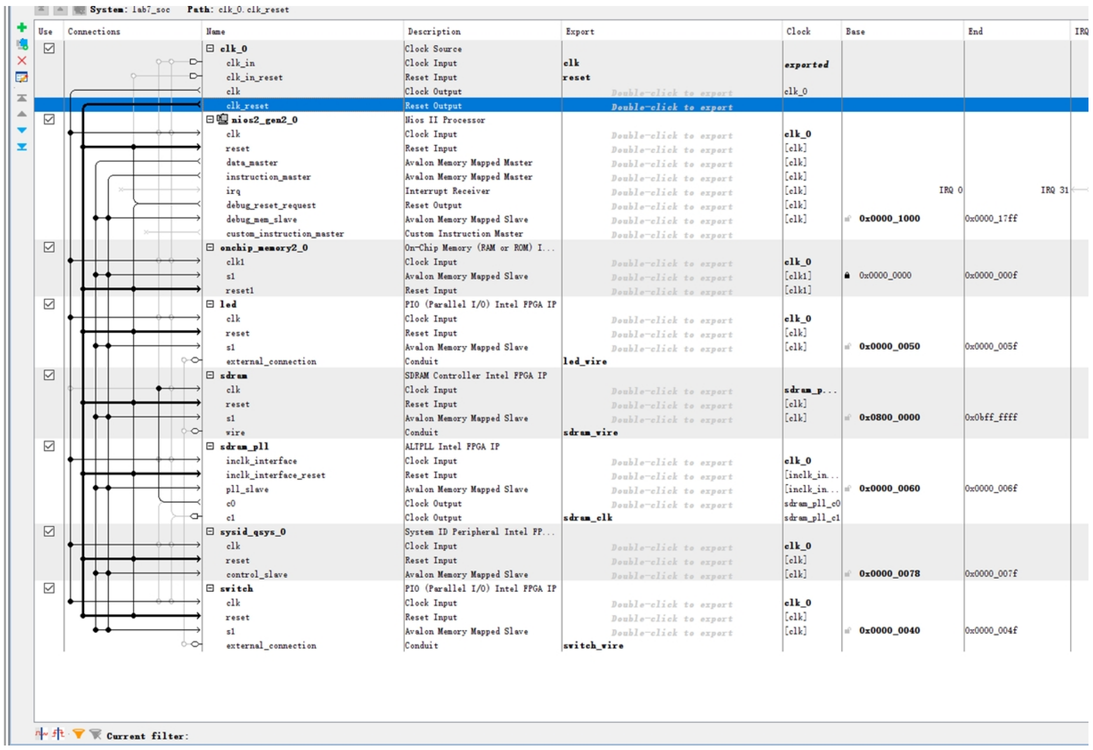


For sysid_qsys_0,is is an ID checker to ensure the compatibility between hardware and software. In other word, it prevents us from loading software onto an FPGA which has an incompatible NIOS II configuration.



For switch, it is a parallel input IO with width 12. It will receive both KEY and SW. it has form: {KEY,SW}. With this PIO, c program can receive the KEY and SW by visiting the corresponding address.

5.2 Platform designer view of the SOC module



System: lab7_soc **Path:** clk_0.clk_reset

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		⊟ **clk_0**	Clock Source					
		clk_in	Clock Input	**clk**	***exported***			
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	*Double-click to export*	clk_0			
		clk_reset	Reset Output	*Double-click to export*				
✓		⊟ **nios2_gen2_0**	Nios II Processor					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31	
		debug_reset_request	Reset Output	*Double-click to export*	[clk]			
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_1000**	0x0000_17ff	
		custom_instruction_master	Custom Instruction Master	*Double-click to export*				
✓		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM) I...					
		clk1	Clock Input	*Double-click to export*	**clk_0**			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0000_0000	0x0000_000f	
		reset1	Reset Input	*Double-click to export*	[clk1]			
✓		⊟ **led**	PIO (Parallel I/O) Intel FPGA IP					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_0050**	0x0000_005f	
		external_connection	Conduit	**led_wire**				
✓		⊟ **sdram**	SDRAM Controller Intel FPGA IP					
		clk	Clock Input	*Double-click to export*	**sdram_p...**			
		reset	Reset Input	*Double-click to export*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0800_0000**	0x0bff_ffff	
		wire	Conduit	**sdram_wire**				
✓		⊟ **sdram_pll**	ALTPLL Intel FPGA IP					
		inclk_interface	Clock Input	*Double-click to export*	**clk_0**			
		inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_in...			
		pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_in...	**0x0000_0060**	0x0000_006f	
		c0	Clock Output	*Double-click to export*	sdram_pll_c0			
		c1	Clock Output	**sdram_clk**	sdram_pll_c1			
✓		⊟ **sysid_qsys_0**	System ID Peripheral Intel FP...					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_0078**	0x0000_007f	
✓		⊟ **switch**	PIO (Parallel I/O) Intel FPGA IP					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0000_0040**	0x0000_004f	
		external_connection	Conduit	**switch_wire**				

Current filter:

6. Describe in word the software component.

6.1. Answers to all INQ questions

6.1.1. What are the differences between the Nios ii/e and Nios ii/f CPUs?

The letter e means economic and f for fast, to be simplify, the economic version has simpler features than the fast version, which only contains features JTAG Debugs and ECC RAM Protection. Also, we use economic version for free while need to pay for fast version. In addition, for a very simple project as what we do in lab 7, these two versions nearly have no performance difference.

6.1.2. What advantage might on-chip memory have for program execution?

Using on-chip memory makes program run much faster, it will have only one clock cycle memory latency, and as it will be much close to CPU, the clock it receive will synchronize better (have less delay than outside memory which is far away).

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

We use the modified Harvard machine. First, we get separate bus to receive and transfer data and instruction, it makes machine to be Harvard. Also, it is modified as when coming into memory, data bus and instruction bus become one so we can access instruction just like data in memory. So we say this is a modified Harvard machine.

6.1.4. Why led peripheral only needs too access to the data bus and no program bus?

As for on-chip memory, it will both send out data and instruction as well as accept. While for led, we only use it as an output which get data and show the value through led on the board and can’t take instruction.

6.1.5. Why dose SDRAM require constant refreshing?

For SDRAM which is a kind of dynamic memory, it uses capacitor as base unit to store data. As we know, capacitors will discharge if we leave it alone and we lose data. That is, in order to keep data, we always keep refreshing to prevent capacitor from discharging.

6.1.6. Justify how to co come up with 1Gbit

SDRAM Parameter	Short Name	Parameter Value
Data Width	[width]	32
# of Rows	[nrows]	13
# of Columns	[ncols]	10
# of Chip Selects	[ncs]	1
# of Banks	[nbanks]	4

$$width\ X\ 2^{nrows}\ X\ 2^{ncols}\ X\ ncs\ X\ nbanks$$
$$= 32bit\ X\ 2^{13}\ X\ 2^{10}\ X\ 1\ X\ 4$$
$$= 32M\ X\ 32bits = 1Gbit$$

6.1.7. What is the theoretical transfer rate to the SDRAM according to the time given?

For access time 5.5ns and 32 bits bus, we get transfer rate:

$$\frac{1s}{5.5ns}x\ 32bit/s = 5.8182x10^9 bit = 5818Mbit/s$$

7. 6.1.8. The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?

For the SDRAM controlled by its controller, time should synchronize, if it is too slow, it may affect the whole system time. Also, as SDRAM need to refresh all the time, if its clock time too slow, the data may crash when being visited.

6.1.9. Why do we need to do this? Hint, check Altera Embedded Peripheral IP datasheet under SDRAM controller.

Because the SDRAM is outside of our chip and as it is far away so the long data path cannot be ignored. That is, if we do not make the delay, the clock enter SDRAM will a little bit later, so we make -3 ns delay to make it arrive early to achieve synchronization. The parameter -3 may decide because of the physical data path to SDRAM on our FPGA.

6.1.10. What address does the NIOS II start execution from? Why do we do this step after assigning the addresses?

It will start from 0x02000000. Every time our program reset, we want it start with a fix rather than random address, here 0x02000000, to avoid conflict with other thing in memory. And we do that by setting sdram.s1 to 0x02000000.

6.1.11. 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). This question is referring to the blinker code.

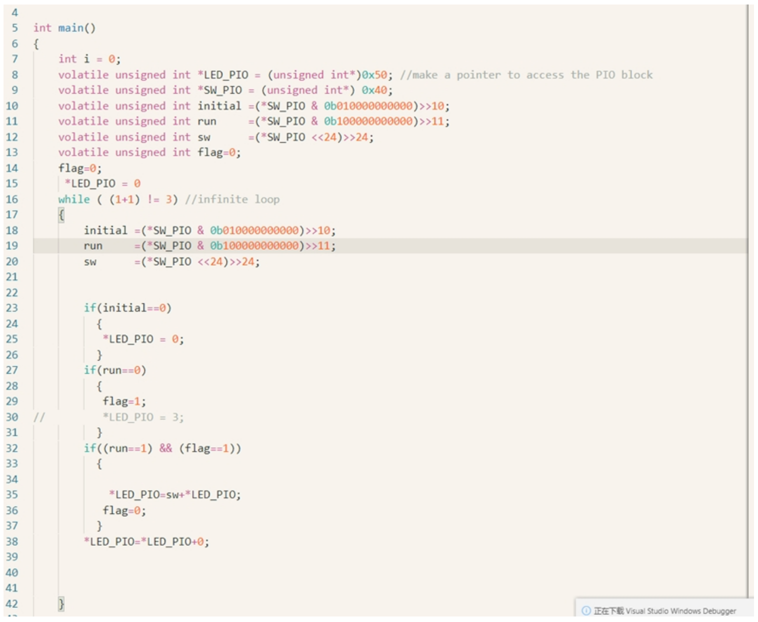
For led blink:

line 8: set pointer to access 0x20 in memory, this address contains the led data. The volatile keyword here is used to tell the complier that every time when it see *LED_PIO, it should read the data from the pointed memory, otherwise, it will only read once to save program running time.

for the inner loop (from line 13 to line 16): line 13 is just a time delay to keep led off for some clock time; line 14 light on the right most led on board by set last bit to 1 through a OR operation; line 15 keep a time delay to have the led on for some clock cycle; line 16 turn off the led by set the last bit back to 0 through an AND operation.

For accumulator:

There is nothing special for data accessing except we should split the data in *SW as actually we store both switch data and key data in it, we achieve it by left shift, right shift and AND operation. Then, another thing to notice that for every press of run button we only want to add one time. That is, we do the add when program detect run signal to be 1 and then 0, which means add happen after the key is released. So, we have a new value flag to show at some time run button has already been set to 1. For the change from 255 to 1, actually we don’t need to worry about it as we only display the 8 least significant bits and don’t care the forehand.



```

int main()
{
    int i = 0;
    volatile unsigned int *LED_PIO = (unsigned int*)0x50; //make a pointer to access the PIO block
    volatile unsigned int *SW_PIO = (unsigned int*) 0x40;
    volatile unsigned int initial =(*SW_PIO & 0b010000000000)>>10;
    volatile unsigned int run     =(*SW_PIO & 0b100000000000)>>11;
    volatile unsigned int sw      =(*SW_PIO <<24)>>24;
    volatile unsigned int flag=0;
    flag=0;
     *LED_PIO = 0
    while ( (1+1) != 3) //infinite loop
    {
        initial =(*SW_PIO & 0b010000000000)>>10;
        run     =(*SW_PIO & 0b100000000000)>>11;
        sw      =(*SW_PIO <<24)>>24;


        if(initial==0)
          {
           *LED_PIO = 0;
          }
        if(run==0)
          {
           flag=1;
//         *LED_PIO = 3;
          }
        if((run==1) && (flag==1))
          {

            *LED_PIO=sw+*LED_PIO;
           flag=0;
          }
        *LED_PIO=*LED_PIO+0;



    }
```

6.1.12. Look at the various segment (. bss, .heap,. rodata, .rwdata, .stack, .text),
what does each section mean? Give an example of C code which places data into each segment.

. bss: the global data which is not initialized
int a (should be outside any function)

. heap: memory allocated by malloc and some other function like it.
int *my_memory=malloc(sizeof(int))

. rodata: constant global data that can only read
constant int a=0 (should be outside any function)

. rwdata: global data that can both read and write
int a =0 (should be outside any function)

. stack: local variable and also stack fram
int a (in function)

. text: code and other descriptions in the programs
all files with .text

6.2. Post-lab Question

6.2.1. design statics table

LUT	2215
DSP	0
Memory (BRAM)	36864
Flip-Flop	1840bits

Frequency	90.48MHZ
Static Power	102.05 mW
Dynamic Power	41.08 mW
Total Power	204.24mW

6.2.2 Problem encountered and solution

In this lab, when dealing with running program in eclipse, we always have error like mismatching system ID or system timestamp and seems clean the makefile and generate again have no use. However, actually we found such error will not affect the performance of program, so the solution is we just click ignore mismatch system ID and system timestamp on Target Connection step.

7. Conclusion

7.1. Discuss functionality of your design

For this lab, I think the difficulty comes from the tutorial itself that how could we understand so much basic knowledge and idea that never appear before, such as how the total Avalon bus work, how could we using c code to access memory on board. For the lab itself, we successfully have led blink and design the accumulator, one more thing to pay attention is when design accumulator, for single run, actually we expect it only run one time, that is, the add operation happen when the run button is released.

7.2. Comment on the lab materials and anything to improve

As this material contains too much new ideas we never see before, it should be better if some more explanation or just website give simple explanation can be attached, not only the datasheet, but it is also hard to understand as well. Also, it is better to give some potential mistake for some difficult step so that we could know we are wrong and fix them very quickly.