|  |  |
| --- | --- |
| Bus Communication with Segment Displays.  17rd October 2019.  Prof. Randall Brouwer | Daniel Ackuaku |

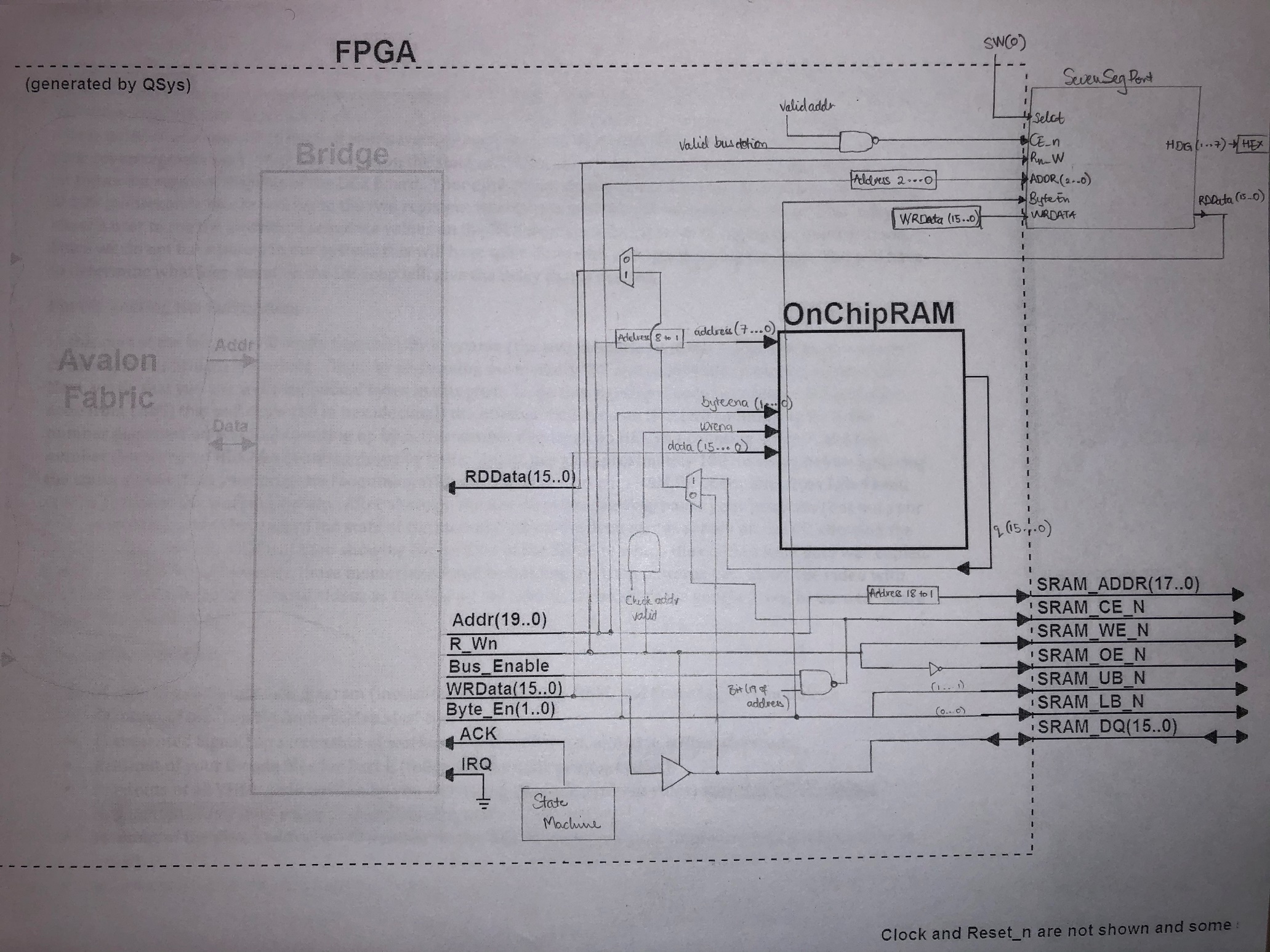


Figure . A block diagram of the SRAM, OnChipRAM and SevenSegPort

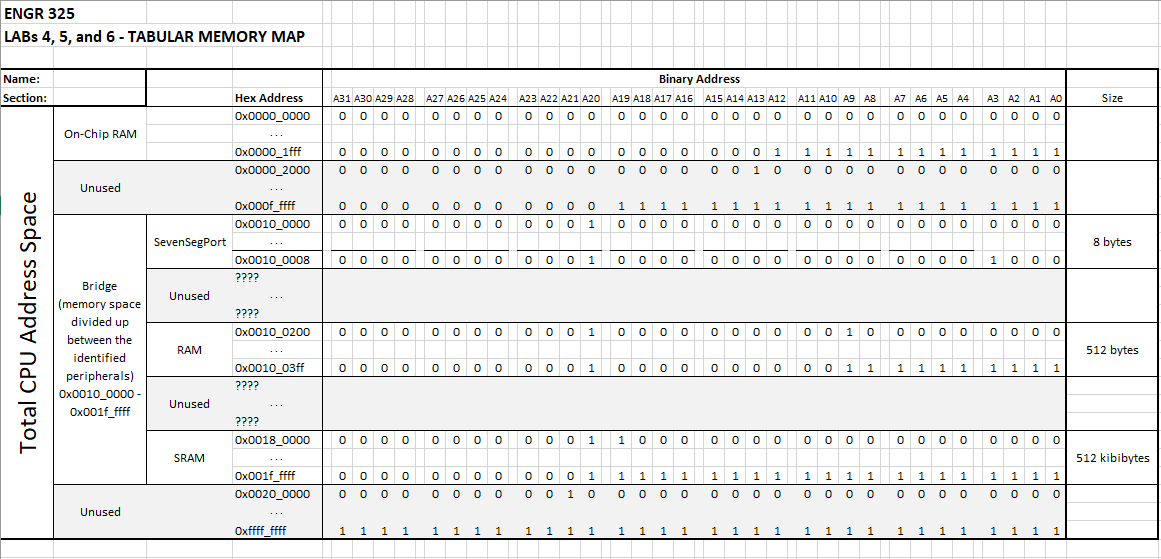


Figure Lab456AddressMap excel file

C CODE for Part E

#include <io.h>

#include "busbridgehal.h"

int main() {

unsigned int SRAMbase; // SRAM base address

unsigned int OnChipRAMbase;

unsigned int mem\_point; // memory fill pointer and fill variable

unsigned int mem\_fill;

unsigned int SevenSegBase;

OnChipRAMbase = 0x100200;

SevenSegBase = 0x100000;

int sub = 999;

mem\_fill=200000;

mem\_point=0x00;

while( mem\_point <= 512){

writeBusIO(4, OnChipRAMbase, mem\_point, mem\_fill);

mem\_fill = mem\_fill-sub;

mem\_point += 0x04;

}

SRAMbase = 0x1c0000;

int readPointer = 0x00;

int i = 0;

int read;

while (readPointer < 512){

read = readBusIO(4, OnChipRAMbase, readPointer);

writeBusIO(4, SRAMbase, readPointer, read);

readPointer += 0x04;

}

unsigned int HEXplus3 = 0x0;

unsigned int HEXplus6 = 0x0;

unsigned int HEXplus7 = 0x0;

unsigned int HEXsuba = 0x0;

unsigned int y = 0;

while(1){

writeBusIO(1, SevenSegBase, 0x00, HEXplus3);

writeBusIO(1, SevenSegBase, 0x01, HEXplus6);

writeBusIO(1, SevenSegBase, 0x02, HEXplus7);

writeBusIO(1, SevenSegBase, 0x03, HEXsuba);

HEXplus3 += 0x03;

HEXplus6 += 0x06;

HEXplus7 += 0x07;

HEXsuba -=0x0a;

for (y = 0; y < 5000000 ; y++) {

asm("nop;nop;nop;");

}

}

return 0;

}

VHDL CODE for Lab 6

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE work.SevenSeg\_pkg.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY Lab6 IS

PORT (

CLOCK\_50 : IN STD\_LOGIC;

SW : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0);

KEY : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0);

SRAM\_ADDR : OUT STD\_LOGIC\_VECTOR(17 DOWNTO 0);

SRAM\_DQ : INOUT STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SRAM\_WE\_N : OUT STD\_LOGIC;

SRAM\_OE\_N : OUT STD\_LOGIC;

SRAM\_UB\_N : OUT STD\_LOGIC;

SRAM\_LB\_N : OUT STD\_LOGIC;

SRAM\_CE\_N : OUT STD\_LOGIC;

HEX0 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX1 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX2 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX3 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX4 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX5 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX6 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0);

HEX7 : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0)

);

END Lab6;

ARCHITECTURE Lab6\_rtl OF Lab6 IS

-------------------------------------------------------------------------

-- Signal definitions

-------------------------------------------------------------------------

type State\_type IS (S1,S2,S3,S4);

signal y\_present, y\_next : State\_type;

Signal ACK\_sig : STD\_LOGIC;

Signal IRQ\_sig : STD\_LOGIC;

Signal Bridge\_address\_sig : STD\_LOGIC\_VECTOR(19 DOWNTO 0);

Signal Bridge\_bus\_enable\_sig : STD\_LOGIC;

Signal Bridge\_byte\_enable\_sig : STD\_LOGIC\_VECTOR(1 DOWNTO 0);

Signal Bridge\_rw\_sig : STD\_LOGIC;

Signal Bridge\_write\_data\_sig : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Signal Bridge\_read\_data\_sig : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Signal q\_signal : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Signal wren\_signal : STD\_LOGIC;

Signal wren\_check : STD\_LOGIC;

Signal ChipEn\_signal : STD\_LOGIC;

Signal ChipEn\_check : STD\_LOGIC;

Signal RDData\_signal : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Signal Intermediate\_signal : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Signal HDig7\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig6\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig5\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig4\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig3\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig2\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig1\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Signal HDig0\_signal : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

component nios\_system is

port (

clk\_clk : in std\_logic := 'X'; -- clk

reset\_reset\_n : in std\_logic := 'X'; -- reset\_n

bridge\_acknowledge : in std\_logic := 'X'; -- acknowledge

bridge\_irq : in std\_logic := 'X'; -- irq

bridge\_address : out std\_logic\_vector(19 downto 0); -- address

bridge\_bus\_enable : out std\_logic; -- bus\_enable

bridge\_byte\_enable : out std\_logic\_vector(1 downto 0); -- byte\_enable

bridge\_rw : out std\_logic; -- rw

bridge\_write\_data : out std\_logic\_vector(15 downto 0); -- write\_data

bridge\_read\_data : in std\_logic\_vector(15 downto 0) := (others => 'X')

);

end component nios\_system;

component OnChipRAM

PORT(

address : IN STD\_LOGIC\_VECTOR (7 DOWNTO 0);

byteena : IN STD\_LOGIC\_VECTOR (1 DOWNTO 0) := (OTHERS => '1');

clock : IN STD\_LOGIC := '1';

data : IN STD\_LOGIC\_VECTOR (15 DOWNTO 0);

wren : IN STD\_LOGIC ;

q : OUT STD\_LOGIC\_VECTOR (15 DOWNTO 0)

);

end component;

component SevenSegPort is

Port (

Clock : IN std\_logic;

Reset\_n : IN std\_logic;

Selct : IN std\_logic;

CE\_n : IN std\_logic;

R\_Wn : IN std\_logic;

ADDR : IN std\_logic\_vector(2 downto 0);

ByteEn : IN std\_logic\_vector(1 downto 0);

WRData : IN std\_logic\_vector(15 downto 0);

RDData : OUT std\_logic\_vector(15 downto 0);

HDig7 : OUT std\_logic\_vector(3 downto 0);

HDig6 : OUT std\_logic\_vector(3 downto 0);

HDig5 : OUT std\_logic\_vector(3 downto 0);

HDig4 : OUT std\_logic\_vector(3 downto 0);

HDig3 : OUT std\_logic\_vector(3 downto 0);

HDig2 : OUT std\_logic\_vector(3 downto 0);

HDig1 : OUT std\_logic\_vector(3 downto 0);

HDig0 : OUT std\_logic\_vector(3 downto 0)

);

end component SevenSegPort;

BEGIN

-------------------------------------------------------------------------

-- Signal assigingments

-------------------------------------------------------------------------

SRAM\_ADDR <= Bridge\_address\_sig(18 downto 1);

SRAM\_WE\_N <= Bridge\_rw\_sig;

SRAM\_OE\_N <= not Bridge\_rw\_sig;

SRAM\_UB\_N <= not Bridge\_byte\_enable\_sig(1);

SRAM\_LB\_N <= not Bridge\_byte\_enable\_sig(0);

SRAM\_CE\_N <= not (Bridge\_bus\_enable\_sig AND Bridge\_address\_sig(19)) ;

wren\_signal <= ( (NOT Bridge\_rw\_sig) AND wren\_check);

wren\_check <= '1' when (Bridge\_address\_sig(19 downto 9) = "0000000001" AND Bridge\_bus\_enable\_sig = '1') else '0';

-- tri-state buffer for write data -- 19 downto-- 0000000001

SRAM\_DQ <= Bridge\_write\_data\_sig when Bridge\_rw\_sig = '0' else (others =>'Z'); --"ZZZZZZZZZZZZZZZZ";

u0 : component nios\_system

port map (

clk\_clk => CLOCK\_50, -- clk.clk

reset\_reset\_n => KEY(0), -- reset.reset\_n

bridge\_acknowledge => ACK\_sig, -- bridge.acknowledge

bridge\_irq => IRQ\_sig, -- .irq

bridge\_address => Bridge\_address\_sig, -- .address

bridge\_bus\_enable => Bridge\_bus\_enable\_sig, -- .bus\_enable

bridge\_byte\_enable => Bridge\_byte\_enable\_sig, -- .byte\_enable

bridge\_rw => Bridge\_rw\_sig, -- .rw

bridge\_write\_data => Bridge\_write\_data\_sig, -- .write\_data

bridge\_read\_data => Bridge\_read\_data\_sig -- .read\_data

);

OnChipRAM\_inst : OnChipRAM

PORT MAP (

address => Bridge\_address\_sig(8 downto 1),

byteena => Bridge\_byte\_enable\_sig,

clock => CLOCK\_50,

data => Bridge\_write\_data\_sig,

wren => wren\_signal,

q => q\_signal

);

SevenSegPort\_inst : component SevenSegPort

port map (

Clock => CLOCK\_50,

Reset\_n => KEY(0),

Selct => SW(0),

CE\_n => ChipEn\_signal,

R\_Wn => Bridge\_rw\_sig,

ADDR => Bridge\_address\_sig(2 downto 0),

ByteEn => Bridge\_byte\_enable\_sig,

WRData => Bridge\_write\_data\_sig,

RDData => RDData\_signal,

HDig7 => HDig7\_signal,

HDig6 => HDig6\_signal,

HDig5 => HDig5\_signal,

HDig4 => HDig4\_signal,

HDig3 => HDig3\_signal,

HDig2 => HDig2\_signal,

HDig1 => HDig1\_signal,

HDig0 => HDig0\_signal

);

----------------------------------------------------------------------------

-- MUX for the Read data conditions

-- between the SRAM\_DQ and the onChipRam q\_signal

----------------------------------------------------------------------------

Intermediate\_signal <= SRAM\_DQ when (Bridge\_bus\_enable\_sig NAND Bridge\_address\_sig(19)) = '0' else q\_signal;

----------------------------------------------------------------------------

-- MUX2 for the Read data conditions

-- the result of MUX1 and the SevenSegPort RDData\_signal

----------------------------------------------------------------------------

Bridge\_read\_data\_sig <= Intermediate\_signal when ChipEn\_signal = '1' else RDData\_signal;

-----------------------------------------------------------------------------

-- State machine process for the acknowledge

-----------------------------------------------------------------------------

process (Bridge\_bus\_enable\_sig, y\_present) is

BEGIN

CASE y\_present IS

WHEN S1 => --State1

if (Bridge\_bus\_enable\_sig = '1') then

ACK\_sig <= '0';

y\_next <= S2;

else

ACK\_sig <= '0';

y\_next <= S1;

end if;

WHEN S2 => --State2

ACK\_sig <= '0';

y\_next <= S3;

WHEN S3 => --State3

ACK\_sig <= '1';

y\_next <= S4;

WHEN S4 => --State4

ACK\_sig <= '0';

y\_next <= S1;

END CASE;

end process;

process (CLOCK\_50, KEY(0))

BEGIN

if (KEY(0) = '0') then

y\_present <= S1;

elsif (CLOCK\_50' EVENT AND CLOCK\_50 = '1') then

y\_present <= y\_next;

end if;

end process;

-----------------------------------------------------------

-- SevenSegPort CE\_N\_signal definition.

-----------------------------------------------------------

ChipEn\_signal <= Bridge\_bus\_enable\_sig NAND ChipEn\_check;

ChipEn\_check <= '1' when (Bridge\_address\_sig(19 downto 5) = "0000000000000000" AND Bridge\_address\_sig(0) = '0') else '0';

------------------------------------------------------------

IRQ\_sig <= '0';

------------------------------------------------------------

-- Converting the register vlaues to SevenSeg

-- Hex display digits

------------------------------------------------------------

HEX7 <= convert\_to\_7seg(HDig7\_signal);

HEX6 <= convert\_to\_7seg(HDig6\_signal);

HEX5 <= convert\_to\_7seg(HDig5\_signal);

HEX4 <= convert\_to\_7seg(HDig4\_signal);

HEX3 <= convert\_to\_7seg(HDig3\_signal);

HEX2 <= convert\_to\_7seg(HDig2\_signal);

HEX1 <= convert\_to\_7seg(HDig1\_signal);

HEX0 <= convert\_to\_7seg(HDig0\_signal);

END Lab6\_rtl;



Figure . A photo of my ID and ID num on HEX display.

VHDL Code for SevenSeg

-- Daniel Ackuaku

-- Date: 10/12/2019

Library ieee;

USE ieee.std\_logic\_1164.all;

-- Explanation of the entity and what it does...

-- (don't make these lines too long to avoid word wrapping)

Entity SevenSegPort is

Port (

Clock : IN std\_logic; -- fill in with descriptions of each port

Reset\_n : IN std\_logic;

Selct : IN std\_logic;

CE\_n : IN std\_logic;

R\_Wn : IN std\_logic;

ADDR : IN std\_logic\_vector(2 downto 0);

ByteEn : IN std\_logic\_vector(1 downto 0);

WRData : IN std\_logic\_vector(15 downto 0);

RDData : OUT std\_logic\_vector(15 downto 0);

HDig7 : OUT std\_logic\_vector(3 downto 0);

HDig6 : OUT std\_logic\_vector(3 downto 0);

HDig5 : OUT std\_logic\_vector(3 downto 0);

HDig4 : OUT std\_logic\_vector(3 downto 0);

HDig3 : OUT std\_logic\_vector(3 downto 0);

HDig2 : OUT std\_logic\_vector(3 downto 0);

HDig1 : OUT std\_logic\_vector(3 downto 0);

HDig0 : OUT std\_logic\_vector(3 downto 0)

);

End Entity SevenSegPort;

Architecture Display OF SevenSegPort IS

-- This area is used to define types and any internal signals

-- Fill\_In as necessary to define signals like "Result", etc

-- Defining the signals for input, output and timing control of the progamme

Signal Register1 : std\_logic\_vector(31 downto 0); -- input register

Signal Register2 : std\_logic\_vector(31 downto 0); -- input reigster

Begin

-------------------------------------------------

-- Selct statement for the registers

-------------------------------------------------

Process (Selct, Register1, Register2) is

Begin

Case Selct IS

When '0' =>

HDig7 <= Register1(31 downto 28) ;

HDig6 <= Register1(27 downto 24) ;

HDig5 <= Register1(23 downto 20) ;

HDig4 <= Register1(19 downto 16) ;

HDig3 <= Register1(15 downto 12) ;

HDig2 <= Register1(11 downto 8) ;

HDig1 <= Register1(7 downto 4) ;

HDig0 <= Register1(3 downto 0) ;

When '1' =>

HDig7 <= Register2(31 downto 28) ;

HDig6 <= Register2(27 downto 24) ;

HDig5 <= Register2(23 downto 20) ;

HDig4 <= Register2(19 downto 16) ;

HDig3 <= Register2(15 downto 12) ;

HDig2 <= Register2(11 downto 8) ;

HDig1 <= Register2(7 downto 4) ;

HDig0 <= Register2(3 downto 0) ;

End Case;

End Process;

--CE-N

-----------------------------------------------------------------------------

-- process

-- reading the input into register1

-----------------------------------------------------------------------------

process (Clock, Reset\_n) is

begin

if ( Reset\_n = '0') then -- if reset is pushed

Register1 <= "00000000000000000000000000000000"; -- set register1 to 0

Register2 <= "00000000000000000000000000000000"; -- set register1 to 0

elsif (Clock'EVENT AND Clock = '1' ) then

if (CE\_N = '0' AND R\_Wn = '0') then -- when CE\_N is asserted low

-- wirte to the lower bits of Register1&2

if (ByteEn(0) = '1')then

if ( ADDR = "000" )then

Register1(7 downto 0) <= WRData(7 downto 0);

elsif ( ADDR = "010" )then

Register1(23 downto 16) <= WRData(7 downto 0);

elsif ( ADDR = "100" )then

Register2(7 downto 0) <= WRData(7 downto 0);

elsif ( ADDR = "110" )then

Register2(23 downto 16) <= WRData(7 downto 0);

end if;

end if;

-- wirte to the upper bits of Register1&2

if (ByteEn(1) = '1')then

if ( ADDR = "000" )then

Register1(15 downto 8) <= WRData(15 downto 8);

elsif ( ADDR = "010" )then

Register1(31 downto 24) <= WRData(15 downto 8);

elsif ( ADDR = "100" )then

Register2(15 downto 8) <= WRData(15 downto 8);

elsif ( ADDR = "110" )then

Register2(31 downto 24) <= WRData(15 downto 8);

end if;

end if;

end if;

end if;

end process;

-- assert read for Register 1 independently of the clock.

RDData(15 downto 0) <= Register1(15 downto 0) when ( R\_Wn = '1' AND (ADDR = "000") )else

Register1(31 downto 16) when ( R\_Wn = '1' AND (ADDR = "010") ) else

Register2(15 downto 0) when ( R\_Wn = '1' AND (ADDR = "100" )) else

Register2(31 downto 16) when ( R\_Wn = '1' AND (ADDR = "110")) else (others => '0');

End Display;

