|  |  |
| --- | --- |
| Bus Communication with External Memory.  3rd October 2019.  Prof Randall Brouwer | Daniel Ackuaku |

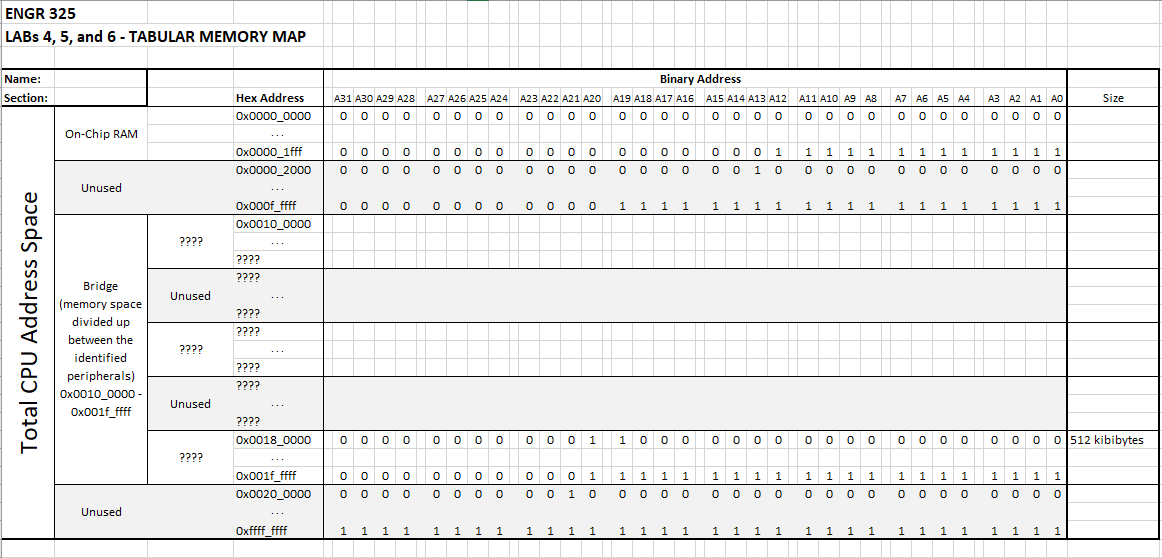


Figure An image of the Lab456AddressMap.xlsx file

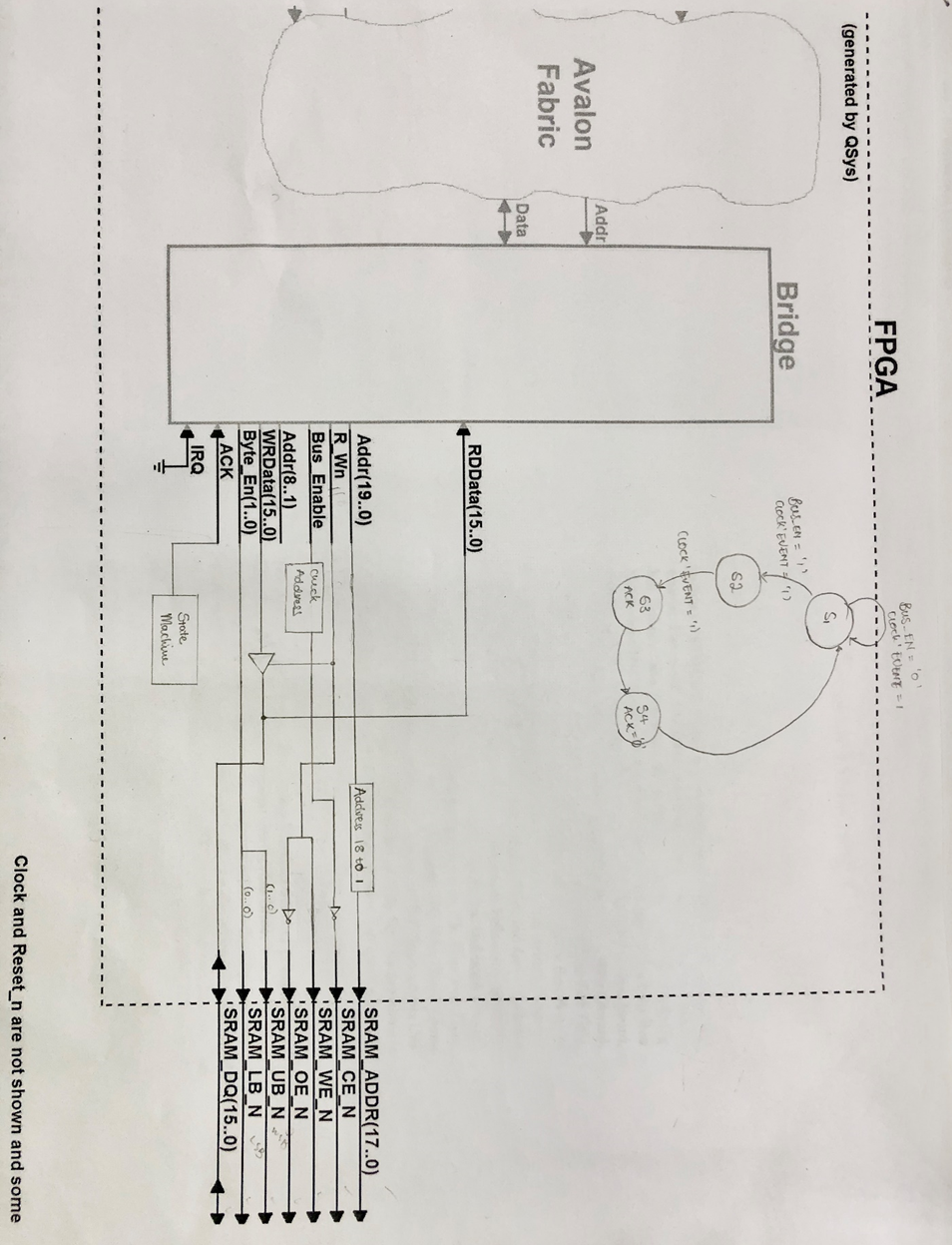


Figure A copy of the block diagram and state diagram.

VHDL CODE for Part A

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY Lab4 IS

PORT (

CLOCK\_50 : IN STD\_LOGIC;

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;

ACK : OUT STD\_LOGIC;

BUS\_EN : OUT STD\_LOGIC

);

END Lab4;

ARCHITECTURE Lab4\_rtl OF Lab4 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);

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') -- read\_data

);

end component nios\_system;

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));

SRAM\_DQ <= Bridge\_write\_data\_sig when Bridge\_rw\_sig = '0'

else (others =>'Z'); --"ZZZZZZZZZZZZZZZZ";

Bridge\_read\_data\_sig <= SRAM\_DQ;

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

);

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

-- 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;

IRQ\_sig <= '0';

ACK <= ACK\_sig;

BUS\_EN <= Bridge\_bus\_enable\_sig;

END Lab4\_rtl;

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

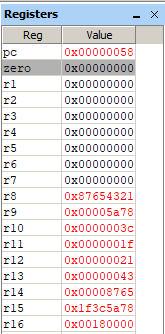


Figure A diagram of the registers for the assembly code SRAM instructions.

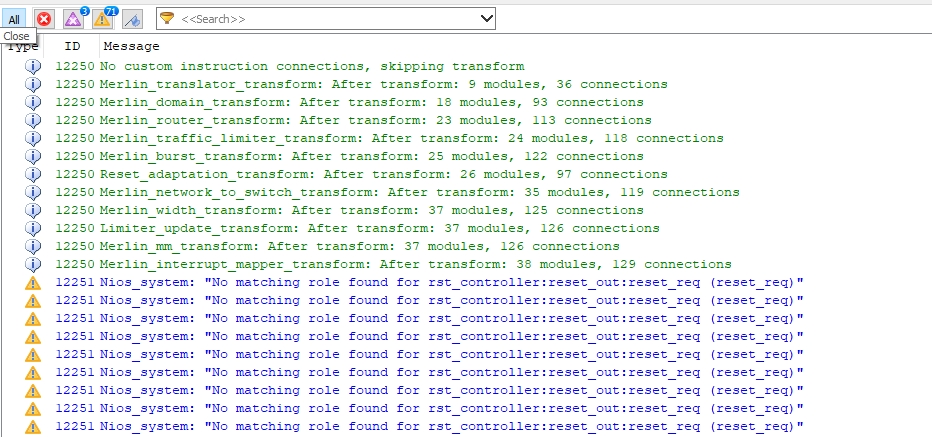


Figure . Quartus warnings 1.

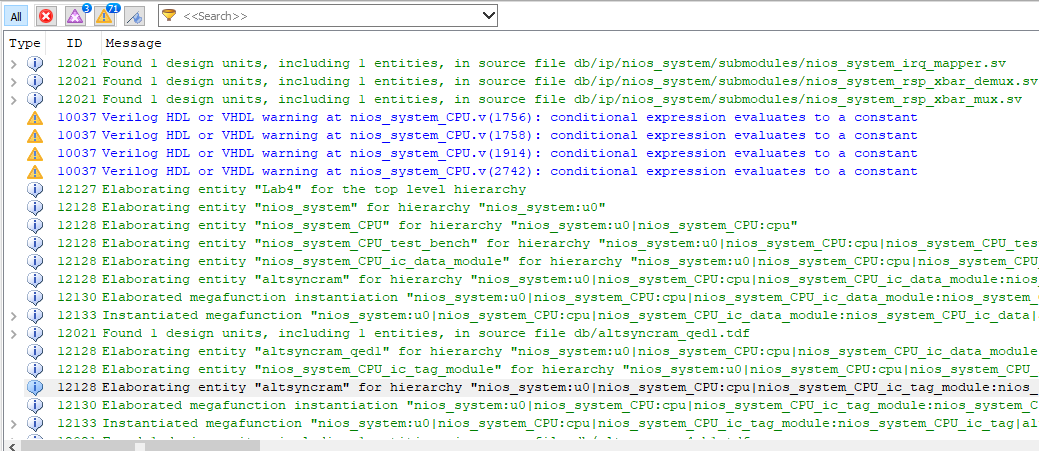


Figure Quartus warnings 2.

Part C

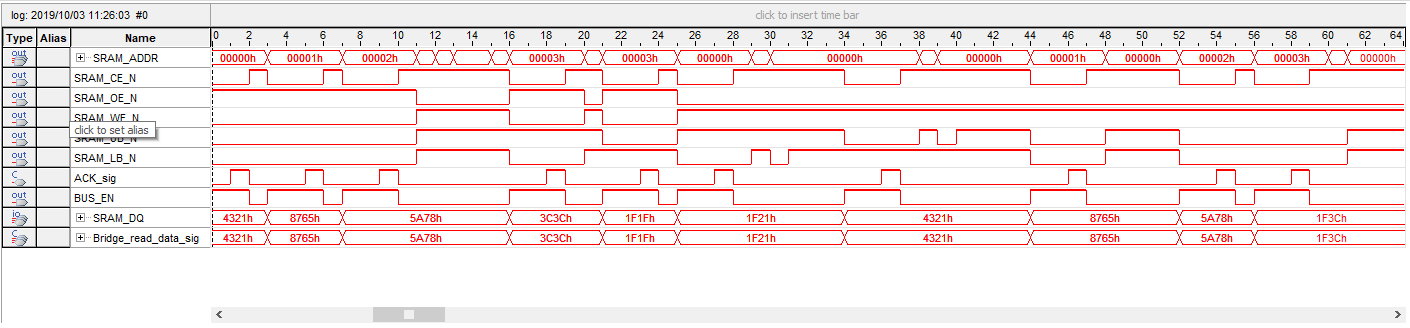


Figure Signal tap of the assembly version.

Part E

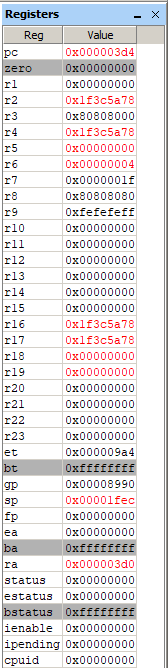
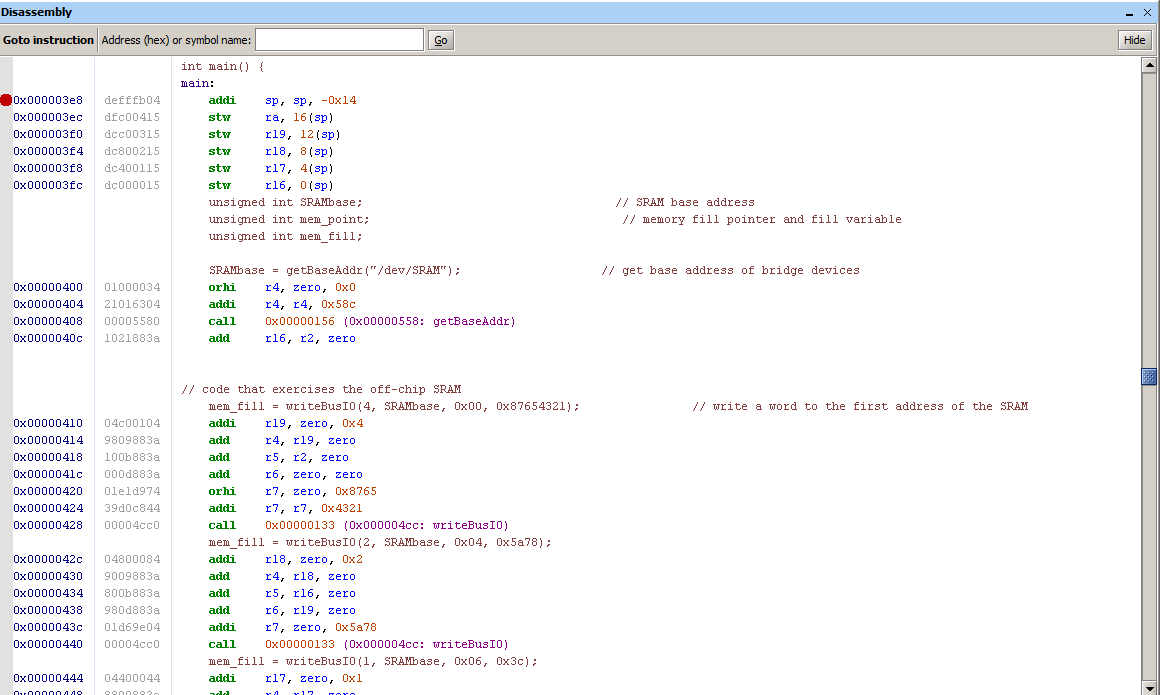
 

Figure 7. Register values of the c code. Figure 8. Disassembly of the c code.

