|  |  |
| --- | --- |
| Bus Communication with Internal Memory.  9th October 2019.  Prof Randall Brouwer | Daniel Ackuaku |

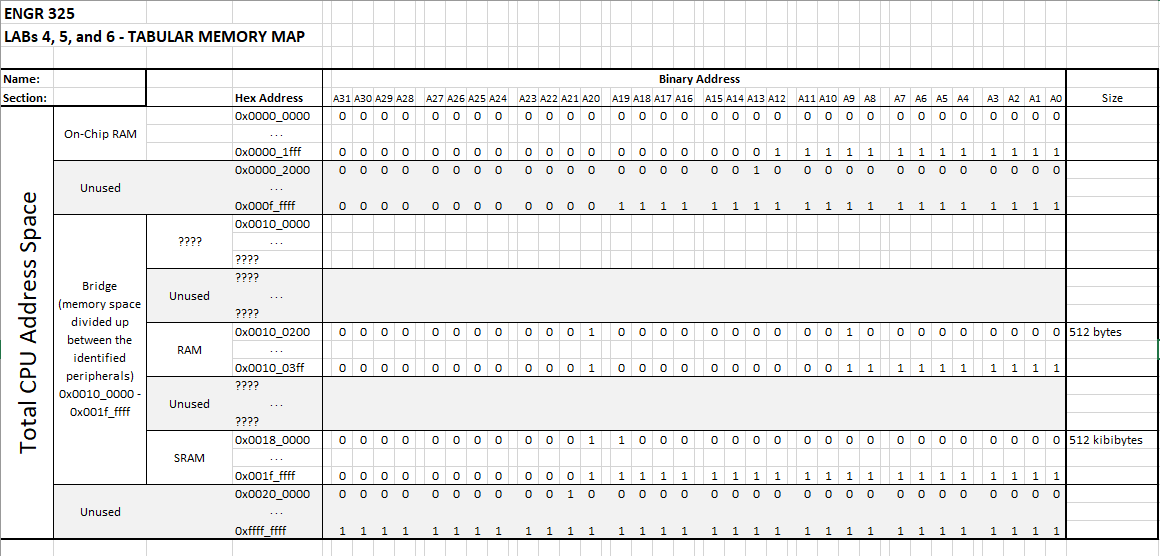


Figure 1 An image of the Lab456AddressMap.xlsx file

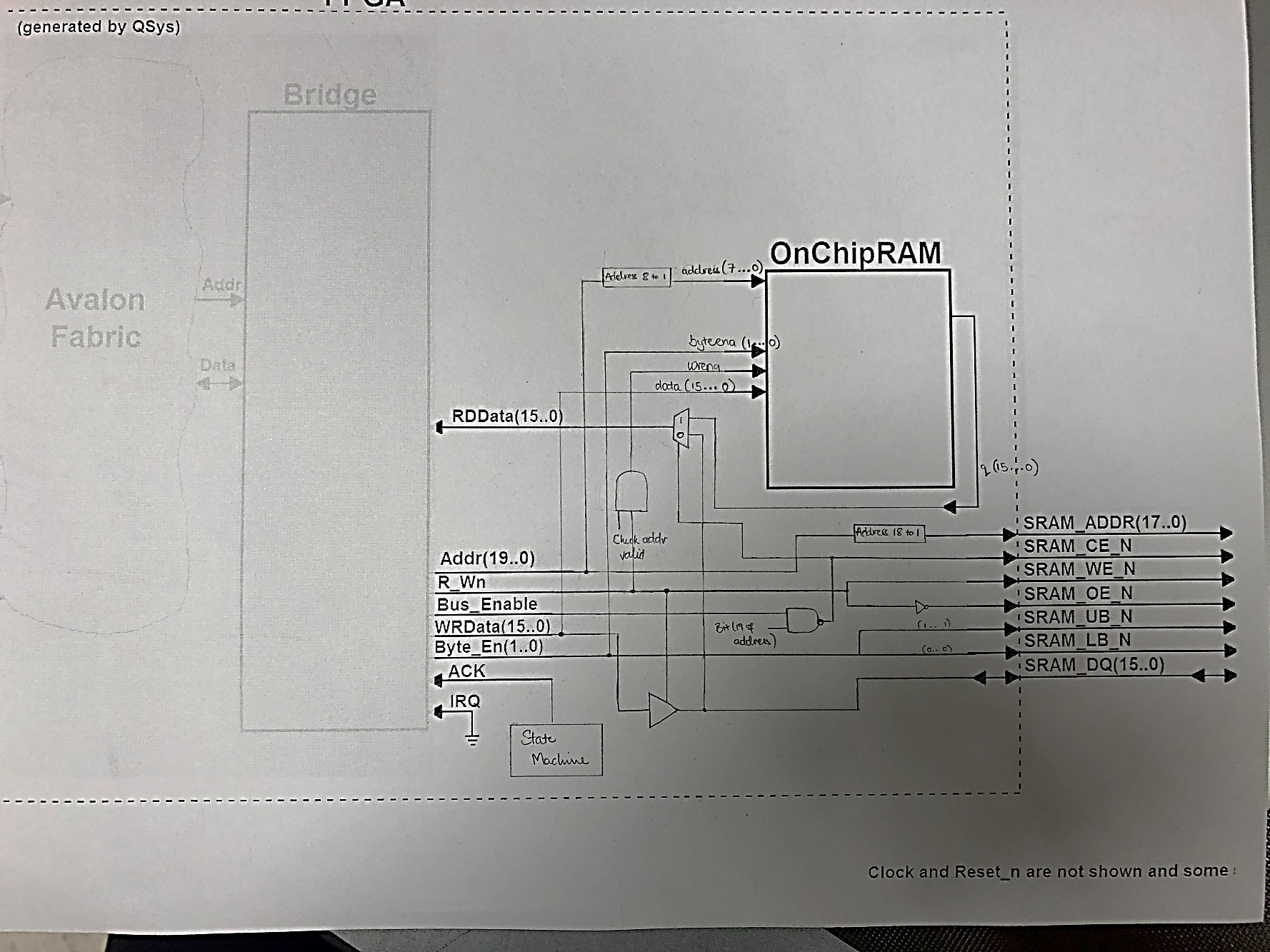


Figure 2 A copy of the block diagram and state diagram.

VHDL CODE for Part A

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY Lab5 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;

BUS\_EN : OUT STD\_LOGIC

);

END Lab5;

ARCHITECTURE Lab5\_rtl OF Lab5 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;

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;

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;

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") else '0';

-- tri-state buffer for write data

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

);

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

-- MUX for the Read data conditions

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

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

IRQ\_sig <= '0';

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

END Lab5\_rtl;

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



Figure . Data being written into the RAM.

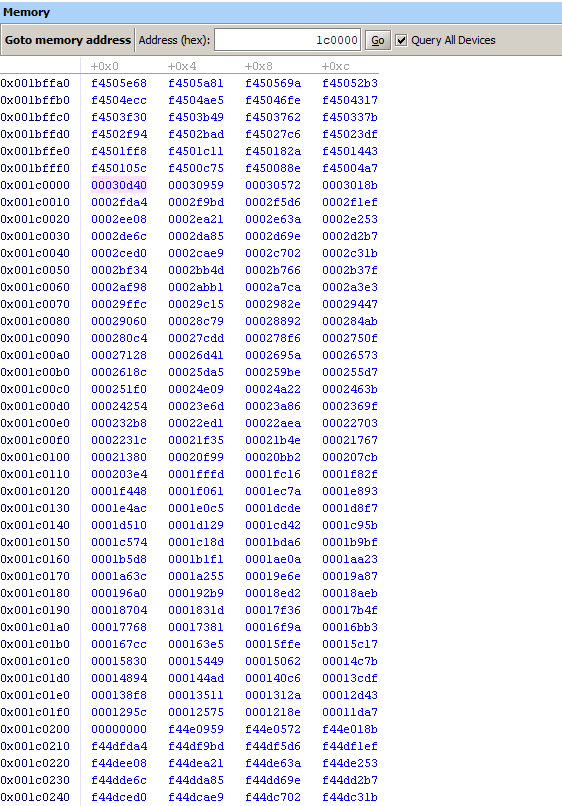


Figure . Data read from RAM being written into the SRAM

C code foreading and writing to the RAM

#include <io.h>

#include "busbridgehal.h"

int main() {

unsigned int SRAMbase; // SRAM base address

unsigned int OnChipRAMbase;

unsigned int mem\_point; unsigned int mem\_fill;

OnChipRAMbase = 0x100200;

// code that exercises the off-chip SRAM

writeBusIO(4, OnChipRAMbase, 0x00, 0x87654321);

writeBusIO(2, OnChipRAMbase, 0x04, 0x5a78);

writeBusIO(1, OnChipRAMbase, 0x06, 0x3c);

writeBusIO(1, OnChipRAMbase, 0x07, 0x1f);

// add commands to read back the memory using readBusIO() function

readBusIO(1, OnChipRAMbase, 0x00);

readBusIO(1, OnChipRAMbase, 0x01);

readBusIO(2, OnChipRAMbase, 0x02);

readBusIO(4, OnChipRAMbase, 0x04);

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 read;

while (readPointer < 512){

read = readBusIO(4, OnChipRAMbase, readPointer);

writeBusIO(4, SRAMbase, readPointer, read);

readPointer += 0x04;

}

return 0;

}

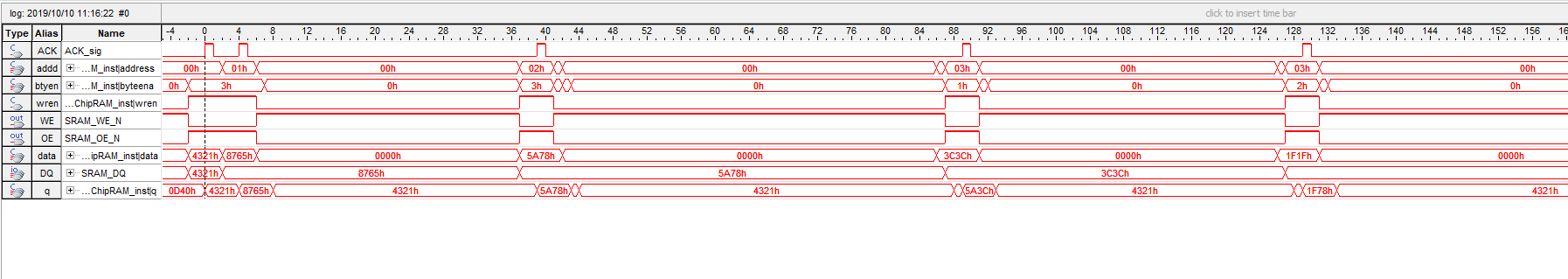


Figure . Signal tap for writes.

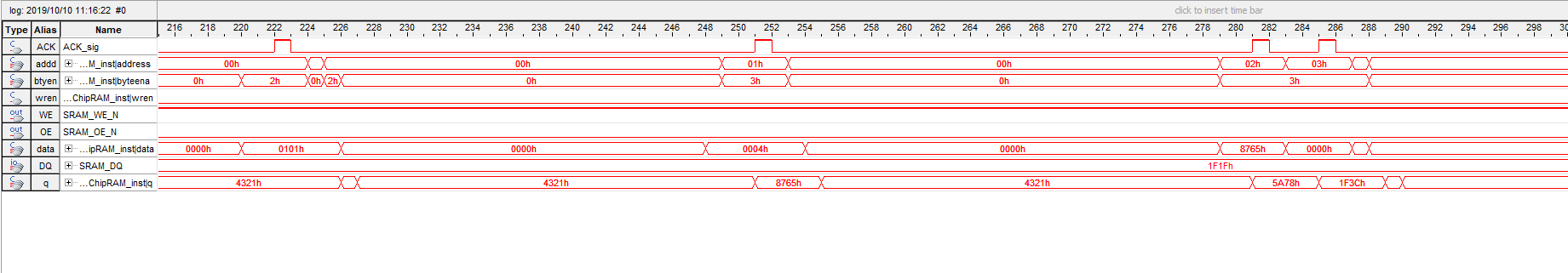


Figure . Signal tap for reads