Texas A&M University

ECEN-719 Lab3 Report

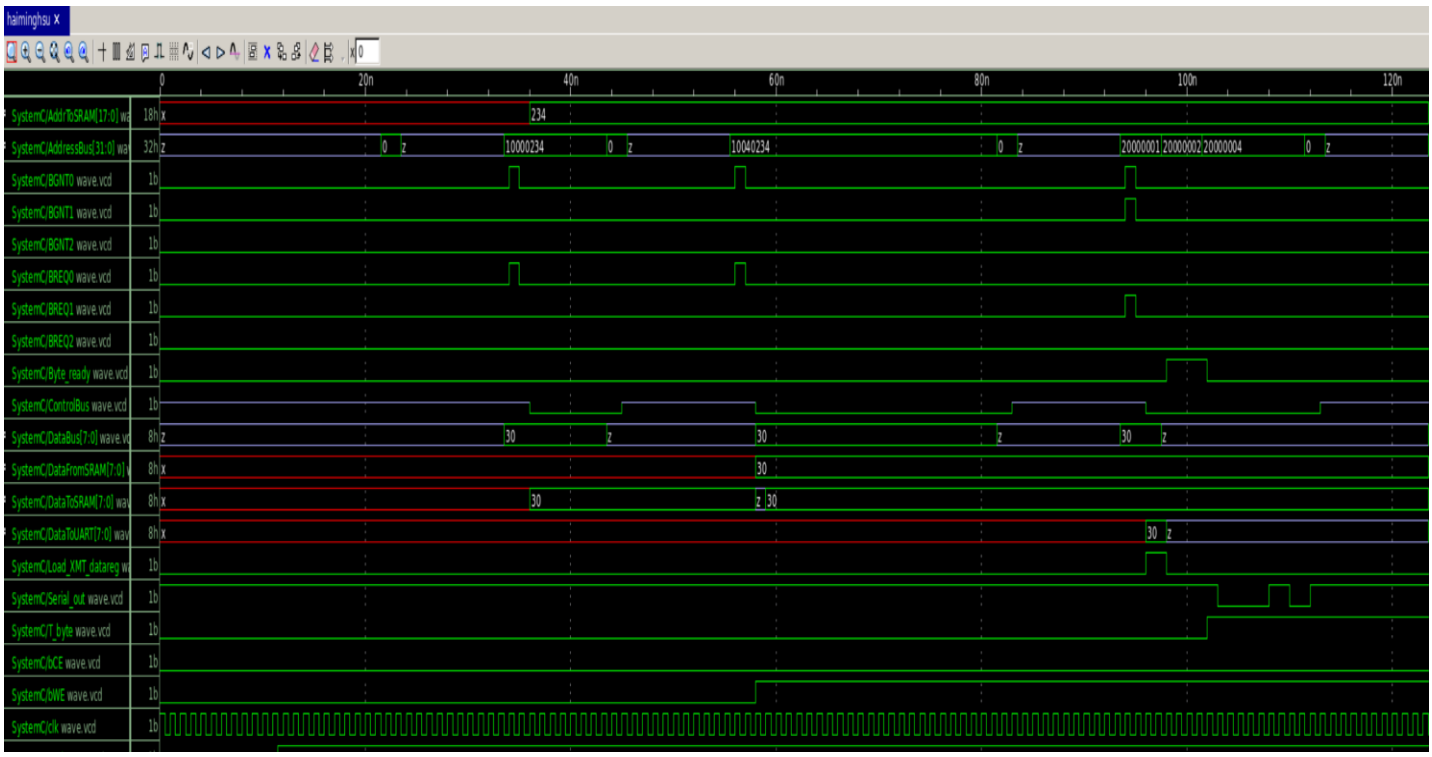
NAME: Hai-Ming, Hsu

UID: 433004106

Section: 602

Department of Electrical & Computer Engineering

1. *Screenshots of the waveform with analysis*



This time's lab we connect what we built before with some new stuffs all together. SRAM, UART Transmitter, Decoders for both, Bus and an Arbiter

From the waveform, it works functionally as what we expect from the Testbench. After the reset, Tb writes data 30 to SRAM 0x00234(the last 18 bits from Address bus)

ID[31:28]	reserved[27:20]	CE_b[19]	WE_b[18]	Addr[17:0]

Figure 5. Address map to the SRAM

We can also see there is a Bus request and grant between the sram and arbiter.

Then, Tb read data 30 from SRAM 0x00234.

The last operation is to transmit data which is read from SRAM through UART. The control signals are set in the last three bits from Address bus.

ID[31:28]	reserved[27:4]	T_byte[2]	Byte_ready[1]	Load_XMT_datareg[0]

Figure 6. Address map to the UART

2. *Screenshots of the simulation output in Vista*

```
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libc.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
Vista SystemC 2.2 Runtime Kernel.
Built September 15, 2011. License version 2011.9.
Copyright (c) 2005-2011, Mentor Graphics Corporation.

        SystemC 2.2.0 --- Sep 15 2011 00:24:25
      Copyright (c) 1996-2006 by all Contributors
            ALL RIGHTS RESERVED

Warning: (W506) illegal characters: SRAM WRAP substituted by SRAM_WRAP
In file: /vista/Vista32x/64bit/v2/Vista312release/src/tlm2.0.1/../CPP/systemc22/sysc/impl/kernel/sc_object.cpp:267
Merging /home/grads/h/haiminghsu/ECEN719/lab3/bus/build/D_PRJDIR_/src/main.exe ...Done.
Saving types data file /home/grads/h/haiminghsu/ECEN719/lab3/bus/sim/main.db...Done.
WARNING: Default time step is used for VCD tracing.

@1500 ps:: >>>>>>>>>>>> Start Simulation

@33500 ps:: >>  Write data to SRAM: 0x30

@55500 ps:: >>  Set SRAM to read mode

@71500 ps:: >>  Read data from data bus: 0x030

@123500 ps:: >>>>>>>>>>>> End Simulation

@123500 ps:: >>>>>>>>>>>> Start Simulation
SystemC: simulation stopped by user.
Program is about to exit.
#0  0x0000000000502e60 in summit_sc::Runtime::atExitCallback ()
(vista)
```

3. *Screenshots of your code in this design with reasonable comments*

```
//=========================================
// Function : Asynchronous SSRAM
//=========================================
#include "systemc.h"

#define DATA_WIDTH        8
#define ADDR_WIDTH        18
#define SRAM_DEPTH         1 << ADDR_WIDTH

SC_MODULE (SRAM) {
  // ----- Declare Input/Output ports -----
    sc_in_rv < ADDR_WIDTH >   Addr ;
    sc_in < bool >      bWE;
    sc_in < bool >      bCE;
    sc_in_rv < DATA_WIDTH  >  InData;
    sc_out_rv < DATA_WIDTH  > OutData;

  // ----- Internal variables -----
  // ...
  sc_uint <DATA_WIDTH> RAM [SRAM_DEPTH];
  uint data, adr;

  // ----- Code Starts Here -----
  // Memory Write Block
  // Write Operation : When we_b = 0, ce_b = 0
  void function_write(){
      if(!(bWE.read()) && !(bCE.read())){
          data = InData.read().to_uint();
          adr = Addr.read().to_uint();
          RAM[adr] = data;
      }
  }

  // Memory Read Block
  // Read Operation : When we_b = 1, ce_b = 0
  void function_read(){
      if((bWE.read()) && !(bCE.read())){
          adr = Addr.read().to_uint();
          OutData.write(RAM[adr]);
      }
  }


  // ----- Constructor for the SC_MODULE -----
  // sensitivity list
  SC_CTOR(SRAM) {
    SC_METHOD(function_write);
  sensitive << bWE << bCE << Addr << InData;
    SC_METHOD(function_read);
  sensitive << bWE << bCE << Addr;
  }
};
```

SRAM

```
//=========================================
// Function : SRAM_WRAP.cpp
//=========================================
#include "SRAM_WRAP.h"

// ----- Code Starts Here -----
void SRAM_WRAP::Bus_Control() {
  sc_uint <4> Adr = AddressBus.read().to_uint() >> 28;
  sc_uint <1> bWRITE = (AddressBus.read().to_uint() >> 18) & 0x1;

  uint data = OutData.read().to_uint();
  uint adr = AddressBus.read().to_uint();

  if(IntEnable){
    ControlBus.write(0);
  }
  else {
    ControlBus.write("Z");
  }

  if(IntEnable){
    if(Adr == 0x1 && bWRITE ==1)
      DataBus.write(data);
    else
      DataBus.write("ZZZZZZZZ");
  }
  else {
    DataBus.write("ZZZZZZZZ");
  }
}

void SRAM_WRAP::Function_SRAM_WRAP() {
  AddrDecoded = AddressBus.read().to_uint() >> 28;

  if(!bReset.read()){
    IntEnable = 0;
    Breq.write(0);
  }
  else if(IntEnable){
    if(AddrDecoded == 0x1) {
      // Address Decoding Matching
      // Decode the message from system bus, and pass it to SRAM.
      // Insert your code here.

                  Address.write(AddressBus.read().to_uint() & 0x3ffff);
                  bWE.write(AddressBus.read().to_uint() >> 18 & 0x1);
                  bCE.write(AddressBus.read().to_uint() >> 19 & 0x1);
                  InData.write(DataBus.read());

    }
    else {
      IntEnable = 0;
      Breq.write(0);
    }
  }
  else {  // !IntEnable
    if(Bgnt) {
      IntEnable = 1;
      Breq.write(0);
    }
    else if(AddrDecoded == 0x1){
      IntEnable = 0;
      Breq.write(1);
    }
    else {
      IntEnable = 0;
      Breq.write(0);
    }
  }
 }
```

SRAM decoder

```
//=========================================
// Function : UART Transmitter
//=========================================
#include "UART_XMTR.h"

  // ----- Code Starts Here -----

  void UART_XMTR::Send_bit() {
                        switch(IntState)  //STATE_MACHINE
    {
      case STATE_IDLE:
                            if( Load_XMT_datareg.read() == 1){
                              XMT_datareg  = Data_Bus.read();
                              NextIntState = STATE_IDLE;
                            }
                            else if ( Byte_ready.read() == 1){
                              XMT_shftreg  = (XMT_datareg.range(WORD_SIZE-1,0),'1');
                              NextIntState = STATE_WAITING;
                            }
                            else{
                              NextIntState = STATE_IDLE;
                            }
                            break;

      case STATE_WAITING:
        if( T_byte.read() == 1){
                              XMT_shftreg  = XMT_shftreg-1;
                              NextIntState = STATE_SENDING;
                            }
                            else{
                              NextIntState = STATE_WAITING;
                            }
                            break;

      case STATE_SENDING:
                            if( bit_count <= WORD_SIZE+1){
                              XMT_shftreg = ('1',XMT_shftreg.range(8,1));
                              bit_count   = bit_count + 1;
                              NextIntState = STATE_SENDING;
                            }
                            else{
                              bit_count = 0;
                              NextIntState = STATE_IDLE;
                            }
                            break;

      default: {
        NextIntState = STATE_IDLE;
      }

    }

                        Serial_out.write(XMT_shftreg[0]);

  }

  void UART_XMTR::Initialize() {
    if(!rst_b.read()) {
      IntState = STATE_IDLE;

      XMT_shftreg = 0x1ff;
      bit_count = 0;
    }
    else {
      IntState = NextIntState;
    }
  }
```

UART

```
//=========================================
// Function : UART_XMTR_WRAP.cpp
//=========================================
#include "UART_XMTR_WRAP.h"

// ----- Code Starts Here -----
void UART_XMTR_WRAP::Bus_Control() {
  uint bControl = ControlBus.read().to_uint();
  if(IntEnable){
    ControlBus.write(0);
  }
  else {
    ControlBus.write("Z");
  }
}

void UART_XMTR_WRAP::Function_UART_XMTR_WRAP() {
  AddrDecoded = AddressBus.read().to_uint() >> 28;

  if(!bReset.read()){
    IntEnable = 0;
    Breq.write(0);
  }else if(IntEnable){
    if(AddrDecoded == 0x2) {
      // Address Decoding Matching
      // Decode the message from the system bus, and pass it to UART_XMTR.
      // Insert your code here.
                Load_XMT_datareg.write(AddressBus.read().to_uint() & 0x1);
                Byte_ready.write(AddressBus.read().to_uint() >> 1 & 0x1);
                T_byte.write(AddressBus.read().to_uint() >> 2 & 0x1);
                InData.write(DataBus.read());

    }else {
      IntEnable = 0;
      Breq.write(0);
    }
  }else { // !IntEnable
    if(Bgnt) {
      IntEnable = 1;
      Breq.write(0);
    }else if(AddrDecoded == 0x2){
      IntEnable = 0;
      Breq.write(1);
    }else {
      IntEnable = 0;
      Breq.write(0);
    }
  }
 }
```

UART decoider

```
#include "SRAM_WRAP.h"
#include "Arbiter.h"
#include "test.h"

int sc_main (int argc, char* argv[]) {

  // Input/Output Signal --------------------------------

  // Wrapper(Decoder) - Arbiter
  sc_signal < bool >    BREQ2, BREQ1, BREQ0;
  sc_signal < bool >        BGNT2, BGNT1, BGNT0;

  // Wrapper(Decoder) of SRAM - SRAM
  sc_signal_rv < WORD_SIZE >  DataToSRAM;
  sc_signal_rv < WORD_SIZE >  DataFromSRAM;
  sc_signal_rv < WORD_SIZE >  DataToUART;
  sc_signal_rv < ADDR_SIZE >  AddrToSRAM;
  sc_signal < bool >    bCE, bWE;

  // Wrapper(Decoder) of UART - UART
  sc_signal < bool >    Load_XMT_datareg;
  sc_signal < bool >    Byte_ready;
  sc_signal < bool >    T_byte;

  // UART - Test-Bench
  sc_signal < bool >    Serial_out;

  // Test-Bench - Wrapper
  sc_signal_rv < 1 >    ControlBus;
  sc_signal_rv < 8 >        DataBus;
  sc_signal_rv < 32 >       AddressBus;

  sc_signal < bool >    rst_b;
  sc_clock clk("clk", 1, SC_NS);

  // Report Handler
  sc_report_handler::set_actions(SC_ID_VECTOR_CONTAINS_LOGIC_VALUE_, SC_DO_NOTHING);
  sc_report_handler::set_actions(SC_ID_LOGIC_Z_TO_BOOL_, SC_DO_NOTHING);

  // Connect your UART_XMTR module
        UART_XMTR UART_01("UART");
        UART_01(Load_XMT_datareg, Byte_ready, T_byte, rst_b,
                DataToUART, Serial_out, clk);

  // Connect your UART_XMTR_WRAP module
        UART_XMTR_WRAP UARTW_1("UART_WRAP");
        UARTW_1 ( Load_XMT_datareg,Byte_ready,T_byte,DataToUART,
                        BREQ1, BGNT1, DataBus, AddressBus,ControlBus, rst_b, clk);

  // Connect your SRAM module
        SRAM SRAM_01("SRAM");
        SRAM_01(AddrToSRAM, bWE, bCE, DataToSRAM, DataFromSRAM);

  // Connect your SRAM_WRAP module
        SRAM_WRAP SRAMW_1 ("SRAM WRAP");
        SRAMW_1 (AddrToSRAM, DataToSRAM,bCE,bWE, DataFromSRAM, BREQ0, BGNT0, DataBus, AddressBus,ControlBus,
                rst_b, clk);

  // Connect your Arbiter module
        Arbiter ARBITER_01("ARBITER");
        ARBITER_01(BREQ2, BREQ1, BREQ0, BGNT2, BGNT1, BGNT0);

  // Connect your Test-Bench module
        test TEST_01("TESTBENCH");
        TEST_01(clk, rst_b, Serial_out, BREQ2, BGNT2, DataBus,
                AddressBus, ControlBus);

  // Open VCD file
  sc_trace_file *wf = sc_create_vcd_trace_file("wave");

  // Dump the desired signals
  sc_trace(wf, rst_b, "rst_b");
  sc_trace(wf, clk, "clk");

  sc_trace(wf, Load_XMT_datareg, "Load_XMT_datareg");
  sc_trace(wf, Byte_ready, "Byte_ready");
  sc_trace(wf, T_byte, "T_byte");
  sc_trace(wf, DataToUART, "DataToUART");
  sc_trace(wf, Serial_out, "Serial_out");

  sc_trace(wf, DataToSRAM, "DataToSRAM");
  sc_trace(wf, DataFromSRAM, "DataFromSRAM");
  sc_trace(wf, AddrToSRAM, "AddrToSRAM");
  sc_trace(wf, bCE, "bCE");
  sc_trace(wf, bWE, "bWE");

  sc_trace(wf, BREQ0, "BREQ0");
  sc_trace(wf, BREQ1, "BREQ1");
  sc_trace(wf, BREQ2, "BREQ2");
  sc_trace(wf, BGNT0, "BGNT0");
  sc_trace(wf, BGNT1, "BGNT1");
  sc_trace(wf, BGNT2, "BGNT2");

  sc_trace(wf, DataBus, "DataBus");
  sc_trace(wf, AddressBus, "AddressBus");
  sc_trace(wf, ControlBus, "ControlBus");
```

Mod connection

4. Question:

Suppose we have three devices (A, B, C) connected to the Arbiter on ports (2, 1, 0) in "Arbiter.h", please list the order of the priorities of the three devices from high to low.

Solution:

The order of the priorities of these three devices from high to low should be ACB . We may tell it from the sequence if-else sentences defined in the arbiter.

```
if(BREQ2.read())    IntGrant = 4;     // 100
  else if(BREQ0.read())    IntGrant = 1; // 001
  else if(BREQ1.read())    IntGrant = 2; // 010
```

We may jump to the branch of the first matching condition.
So the priorities are ranged as Port 2 (A) > Port 0 (C) > Port 1 (B)