```
//=========================================
// 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
                        sc_bv <32> full_address = AddressBus.read();
                        sc_logic logic = full_address[19] & "0x1";
                        // Bit 19 is CE_b
                        cout << "CE_b: " << logic.to_bool() << endl;
                        CE_b.write(logic.to_bool());
                        // Bit 18 is WE_b
                        logic = full_address[18] & "0x1";
                        cout << "WE_b: " << logic.to_bool() << endl;
                        WE_b.write(logic.to_bool());
                        // Bits [17:0] is Addr
                        sc_uint<18> data_to_write = AddressBus.read().range(18,0).to_uint
();
                        cout << "Addr: " << data_to_write << endl;
                        Addr.write(data_to_write);
                        cout << "Data: " << DataBus.read().to_uint() << endl;
                        // TODO write data to SRAM
                        InData.write(DataBus.read().to_uint());

                }
                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);
                }
        }
}
```

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

#define ADDR_SIZE               18      // 2^18 = 256k
#define WORD_SIZE               8       // 8 bits

//#define _DEBUG_

SC_MODULE (SRAM_WRAP){
        // Signals to SRAM
        sc_out_rv < ADDR_SIZE > Addr;
        sc_out <bool> WE_b;
        sc_out <bool> CE_b;
        sc_inout_rv <WORD_SIZE> InData;
        sc_inout_rv < WORD_SIZE > OutData;

        // Signals to System Bus
        sc_inout_rv <1>  ControlBus;
        sc_inout_rv <8> DataBus;
        sc_inout_rv <32> AddressBus;

        // Signals to Arbiter
        sc_out < bool > Breq;
        sc_in < bool > Bgnt;

        // to TEST-BENCH
        sc_in < bool >                  bReset;
        sc_in < bool >                  clk;

        // Internal signal
        sc_uint < 1 >                   IntEnable;
        sc_uint < 4 >                   AddrDecoded;

        // ----- Code Starts Here -----
        void Function_SRAM_WRAP();
        void Bus_Control();

        // ----- Constructor for the SC_MODULE -----
        // sensitivity list
        SC_CTOR(SRAM_WRAP) {
                SC_METHOD(Function_SRAM_WRAP);
                sensitive << clk.pos() << bReset.neg();

                SC_METHOD(Bus_Control);
                sensitive << clk.pos() << AddressBus << OutData << ControlBus ;
        }
};
```

```
//=========================================
// 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
                        sc_bv <32> full_address = AddressBus.read();
                        DataToUART.write(DataBus.read());
                        sc_logic logic = full_address[0] & "0x1";
                        // Bit 0 is Load_XMT_datareg
                        cout << "Load_XMT_datareg: " << logic.to_bool() << endl;
                        Load_XMT_datareg.write(logic.to_bool());
                        // Bit 1 is Byte_ready
                        logic = full_address[1] & "0x1";
                        cout << "Byte_ready: " << logic.to_bool() << endl;
                        Byte_ready.write(logic.to_bool());
                        // Bit 2 is T_byte
                        logic = full_address[2] & "0x1";
                        cout << "T_Byte: " << logic.to_bool() << endl;
                        T_byte.write(logic.to_bool());
                        cout << endl;
                }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);
                }
        }
 }
```

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

#define WORD_SIZE               8       // 8 bits

SC_MODULE (UART_XMTR_WRAP){
        // Signals to UART_XMTR
        sc_out <bool> T_byte;
        sc_out <bool> Byte_ready;
        sc_out <bool> Load_XMT_datareg;
        sc_inout_rv < WORD_SIZE > DataToUART;

        // Signals to System Bus
        sc_inout_rv <1> ControlBus;
        sc_inout_rv <8> DataBus;
        sc_inout_rv <32> AddressBus;

        // Signals to Arbiter
        sc_out < bool > Breq;
        sc_in < bool > Bgnt;

        // to TEST-BENCH
        sc_in < bool >          bReset;
        sc_in < bool >          clk;

        // Internal Variable
        sc_uint < 1 >           IntEnable;
        sc_uint < 4 >           AddrDecoded;

        // ----- Code Starts Here -----
        void Function_UART_XMTR_WRAP();
        void Bus_Control();

        // ----- Constructor for the SC_MODULE -----
        // sensitivity list
        SC_CTOR(UART_XMTR_WRAP) {
                SC_METHOD(Function_UART_XMTR_WRAP);
                sensitive << clk.pos() << bReset.neg();

                SC_METHOD(Bus_Control);
                sensitive << clk.pos() ;
        }
};
```

```
#include "UART_XMTR_WRAP.h"
#include "UART_XMTR.h"
#include "SRAM.cpp"
#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("TEST 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 uart_wrap_01("UART_WRAP");
        uart_wrap_01(T_byte, Byte_ready, Load_XMT_datareg, DataToUART,
                ControlBus, DataBus, AddressBus, BREQ1, BGNT1, 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 sram_wrap_01("SRAM WRAP");
        sram_wrap_01(AddrToSRAM, bWE, bCE, DataToSRAM, DataFromSRAM, ControlBus, DataBus,
 AddressBus,
                BREQ0, BGNT0, 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,
```

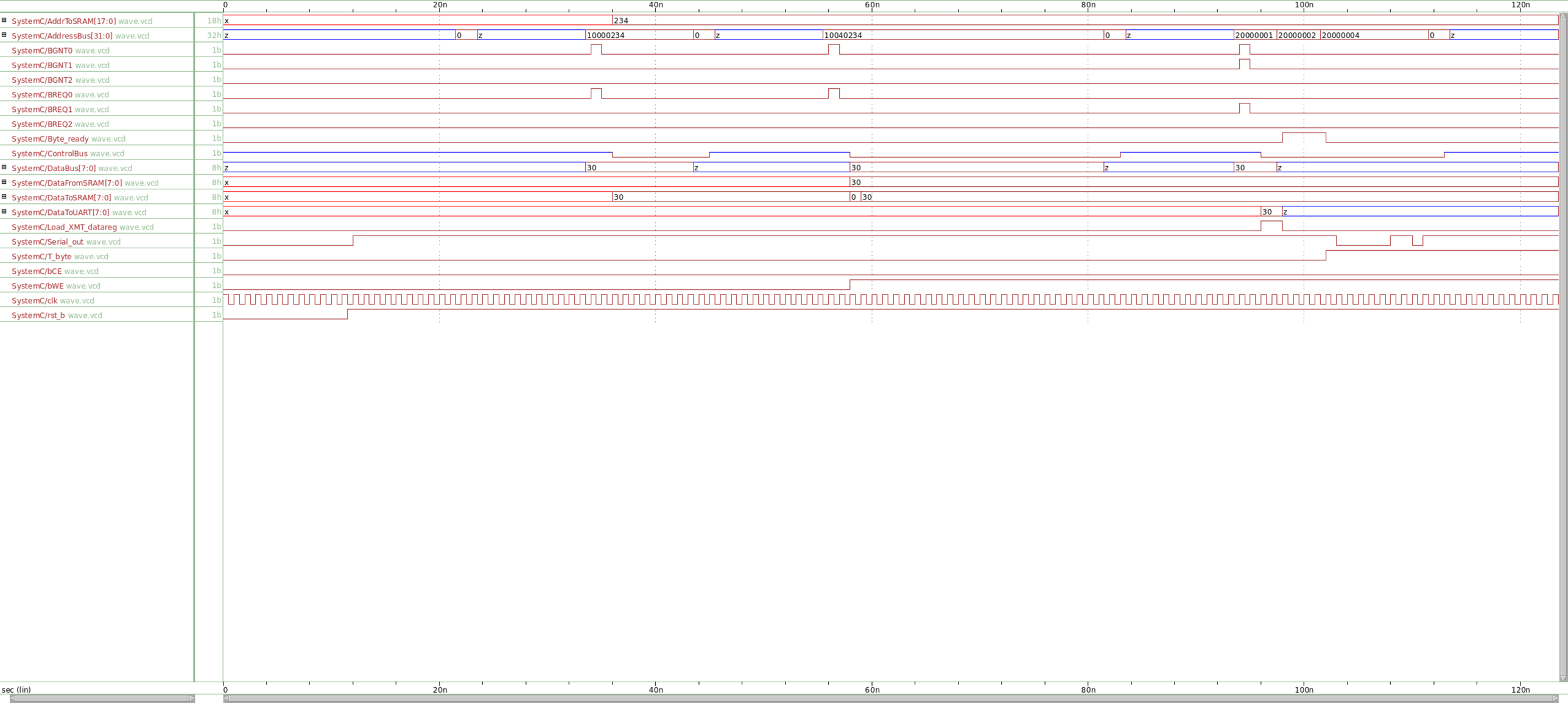
```
                AddressBus, ControlBus);

        // 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");

        // Time to simulate
        // Simulate until it meets sc_stop() if sc_start(-1) or sc_start()
        //sc_start(1000, SC_NS);        // Simulate for 1000ns
        sc_start();

        // Close the dump file
        sc_close_vcd_trace_file(wf);

        return 0;       // Terminate simulation
}
```

```
cd /home/ugrads/s/seth.barberee/ECEN468/Lab3/
gdb.shell.exe
spawn /opt/coe/mentorgraphics/vista312/linux64/tools.64bit/bin/gdb -quiet -interp=opengdb
 -runtcl="/opt/coe/mentorgraphics/vista312/generic/tcl/v2/gdb/gdb.tcl" --nx

(vista) BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.so.6: invalid relocation type 37
BFD: BFD 2.17.50 20061115 assertion fail elf64-x86-64.c:259
BFD: /usr/lib64/libm.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
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: TEST UART substituted by TEST_UART
In file: /vista/Vista32x/64bit/v2/Vista312release/src/tlm2.0.1/../CPP/systemc22/sysc/impl
/kernel/sc_object.cpp:267

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
Writing XXXXXXXX to location 262143
WARNING: Default time step is used for VCD tracing.

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

@33500 ps:: >>  Write data to SRAM: 0x30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30
Writing 030 to location 234
CE_b: 0
```

```
WE_b: 0
Addr: 00234
Data: 30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30
CE_b: 0
WE_b: 0
Addr: 00234
Data: 30

@55500 ps:: >>  Set SRAM to read mode
CE_b: 0
WE_b: 1
Addr: 00234
Data: 0
Reading 030 from 234
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
```

```
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30

@71500 ps:: >>  Read data from data bus: 0x030
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
CE_b: 0
WE_b: 1
Addr: 00234
Data: 30
Load_XMT_datareg: 1
Byte_ready: 0
T_Byte: 0

Load_XMT_datareg: 1
Byte_ready: 0
T_Byte: 0

Load_XMT_datareg: 0
Byte_ready: 1
T_Byte: 0
```

```
Load_XMT_datareg: 0
Byte_ready: 1
T_Byte: 0

Load_XMT_datareg: 0
Byte_ready: 1
T_Byte: 0

Load_XMT_datareg: 0
Byte_ready: 1
T_Byte: 0

Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 1
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 1
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 0
Load_XMT_datareg: 0
Byte_ready: 0
T_Byte: 1

Sending 1

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

@123500 ps:: >>>>>>>>>>>> Start Simulation
SystemC: simulation stopped by user.
Program is about to exit.
#0  0x00000000005040c0 in summit_sc::Runtime::atExitCallback ()
(vista)
```


0
20n
40n
60n
80n
100n
120n
SystemC/AddrToSRAM[17:0] wave.vcd
18h
x
234
SystemC/AddressBus[31:0] wave.vcd
32h
z
0
z
10000234
0
z
10040234
0
z
20000001
20000002
20000004
0
z
SystemC/BGNT0 wave.vcd
1b
SystemC/BGNT1 wave.vcd
1b
SystemC/BGNT2 wave.vcd
1b
SystemC/BREQ0 wave.vcd
1b
SystemC/BREQ1 wave.vcd
1b
SystemC/BREQ2 wave.vcd
1b
SystemC/Byte_ready wave.vcd
1b
SystemC/ControlBus wave.vcd
1b
SystemC/DataBus[7:0] wave.vcd
8h
z
30
z
30
z
30
z
SystemC/DataFromSRAM[7:0] wave.vcd
8h
x
30
SystemC/DataToSRAM[7:0] wave.vcd
8h
x
30
0
30
SystemC/DataToUART[7:0] wave.vcd
8h
x
30
z
SystemC/Load_XMT_datareg wave.vcd
1b
SystemC/Serial_out wave.vcd
1b
SystemC/T_byte wave.vcd
1b
SystemC/bCE wave.vcd
1b
SystemC/bWE wave.vcd
1b
SystemC/clk wave.vcd
1b
SystemC/rst_b wave.vcd
1b
sec (lin)
0
20n
40n
60n
80n
100n
120n