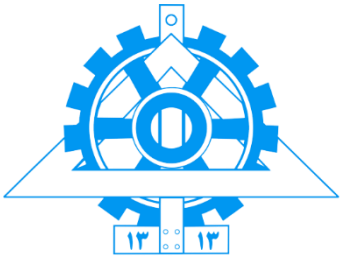
باسمه‌ی تعالی



Object-Oriented Modeling of Electronic Circuits

Computer Assignment 5

System Modeling with CPU

Student Name: Moein Maleki
Student Number: 810197591

Electrical and Computer Engineering Department
Spring 1401

Table of Contents

Developed systemc/ams modules:

Square source:

Header:

```
SC_MODULE(squareWave) {
    sc_out <double> out;

    SC_CTOR(squareWave) :
        out("out") {

        SC_THREAD(squareWaveGeneration);
    }

    void squareWaveGeneration();
};
```

Source:

```
void squareWave::squareWaveGeneration() {
    int halfPeriod = (SQUARE_WAVE_PERIOD_US) / 2;
    while(1) {
        wait(halfPeriod, SC_US);
        out->write(SQUARE_WAVE_MAX_VALUE);
        wait(halfPeriod, SC_US);
        out->write(SQUARE_WAVE_MIN_VALUE);
    }
}
```

Lowpass filter:

Header:

```
SC_MODULE(lowpassFilter) {
    sc_in<double> in;

    sca_tdf::sca_out<double> out;

    sca_eln::sca_r* resUnit;
    sca_eln::sca_c* capUnit;

    sca_eln::sca_de::sca_vsource* inConverter;
    sca_eln::sca_tdf::sca_vsink* outConverter;

    SC_HAS_PROCESS(sc_module_name);
    lowpassFilter(sc_module_name);

    private:
        sca_eln::sca_node_ref gndNode;
        sca_eln::sca_node inNode;
        sca_eln::sca_node outNode;
};
```

Source:

```
lowpassFilter::lowpassFilter(sc_module_name) {
    inConverter = new sca_eln::sca_de::sca_vsource("converter_de2eln", 1.0);
    inConverter->inp(in);
    inConverter->p(inNode);
    inConverter->n(gndNode);
    inConverter->set_timestep(GLOBAL_TIMESTEP_NS, SC_NS);

    resUnit = new sca_eln::sca_r("lowpassFilter_res1", LOWPASS_FILTER_RES_1_VALUE);
    resUnit->n(inNode);
    resUnit->p(outNode);

    capUnit = new sca_eln::sca_c("lowpassFilter_cap1", LOWPASS_FILTER_CAP_1_VALUE);
    capUnit->n(outNode);
    capUnit->p(gndNode);

    outConverter = new sca_eln::sca_tdf::sca_vsink("converter_eln2tdf", 1.0);
    outConverter->outp(out);
    outConverter->p(outNode);
    outConverter->n(gndNode);
}
```

ADC:

Header:

```
SCA_TDF_MODULE(adc) {
    sca_tdf::sca_in<double> in;

    sca_tdf::sca_out<sc_dt::sc_int<ADC_OUTPUT_BITS>> out;

    adc(sc_core::sc_module_name) {}

    void set_attributes();
    void processing();
};
```

Source:

```
void adc::set_attributes() {
    set_timestep(sc_time(GLOBAL_TIMESTEP_NS, SC_NS));
}

void adc::processing() {
    int threshold = (int)pow(2, ADC_OUTPUT_BITS - 1) - 1;

    if(in.read() > threshold)
        out.write(ADC_MAXIMUM_VALUE);
    else if(in.read() < -1 * threshold)
        out.write(-ADC_MAXIMUM_VALUE);
    else {
        out.write(lround(in.read() * (threshold/ADC_MAXIMUM_VALUE)));
    }
}
```

Explaining V_th choice:

ما میخواهیم یک عدد double را در ۱۶ بیت جای دهیم. چون عملا این عدد double، کران بالایی ندارد، کاری که ما میکنیم این است که نسبت عدد double را به مقدار ماکسیممی که در ۱۶ بیت میخواهیم نشان دهیم (ADC_MAXIMUM_VALUE=1000) را پیدا میکنیم، که وقتی دو حالت in < 1000 و in > 1000 را isolate بکنیم، این نسبت عددی کمتر از یک میشود. سپس آنرا ضرب در عددی باید بکنیم که اگر نسبت ۱ بود، تبدیل به ماکسیمم عدد در ۱۶ بیت شود. منطقا آنرا باید ضرب در $2^{16-1} - 1$ که ماکسیمم ما در ۱۶ بیت است.

TDF2DE:

Header:

```
SCA_TDF_MODULE(tdf2De) {
    sca_tdf::sca_in<sc_dt::sc_int<ADC_OUTPUT_BITS>> in;

    sca_tdf::sca_de::sca_out<sc_lv<ADC_OUTPUT_BITS>> out;

    tdf2De(sc_module_name) {}

    void set_attributes();
    void processing();
};
```

Source:

```
void tdf2De::set_attributes() {
    set_timestep(sc_time(GLOBAL_TIMESTEP_NS, SC_NS));
}

void tdf2De::processing() {
    out = in.read().to_int();
}
```

Sensor frontend:

Header:

```
SC_MODULE(sensorFrontend) {
    sc_core::sc_in <double> in;

    sc_core::sc_out <sc_dt::sc_lv <ADC_OUTPUT_BITS>> out;

    lowpassFilter* lowpassFilterUnit;
    adc* adcUnit;
    tdf2De* tdf2DeUnit;

    SC_HAS_PROCESS(sc_module_name);
    sensorFrontend(sc_module_name);

    sca_tdf::sca_signal<double> sineSignal;
    sca_tdf::sca_signal<sc_dt::sc_int<ADC_OUTPUT_BITS>> adcSineSignal;
};
```

Source:

```
sensorFrontend::sensorFrontend(sc_module_name) {
    lowpassFilterUnit = new lowpassFilter("lowpassFilter_inst");
    adcUnit = new adc("adc_inst");
    tdf2DeUnit = new tdf2De("tdf2De_inst");

    lowpassFilterUnit->in(in);
    lowpassFilterUnit->out(sineSignal);

    adcUnit->in(sineSignal);
    adcUnit->out(adcSineSignal);

    tdf2DeUnit->in(adcSineSignal);
    tdf2DeUnit->out(out);
}
```

Register:

Header:

```
SC_MODULE(regizter) {
    sc_in <sc_logic> clk;
    sc_in <sc_logic> outEnable;
    sc_in <sc_lv<16>> in;

    sc_out_rv <16> out;

    sc_signal <sc_lv<16>> temp;

    SC_CTOR(regizter) :
        clk("clk"),
        outEnable("outEnable"),
        in("in"),
        out("out") {

        SC_THREAD(registering);
            sensitive << clk.pos();
        SC_THREAD(tristate);
            sensitive << outEnable << temp;
    }

    void registering();
    void tristate();
};
```

Source:

```
void regizter::registering() {
    while(1) {
        temp = in;
        wait();
    }
}

void regizter::tristate() {
    out.write(REGISTER_LOGIC_Z_VALUE);
    while(1) {
        if(outEnable == SC_LOGIC_1) {
            out.write(temp.read());
        }
        else {
            out.write(REGISTER_LOGIC_Z_VALUE);
        }
        wait();
    }
}
```

Timer:

Header:

```
SC_MODULE(timer) {
    sc_in <sc_logic> clk;
    sc_in <sc_logic> startTimer;
    sc_in <sc_logic> outEnable;

    sc_out_rv <16> timeOut;

    sc_signal <sc_lv<16>> microsPassed;
    sc_signal <sc_logic> timerActive;

    SC_CTOR(timer) {
        SC_THREAD(activating);
            sensitive << clk.pos();
        SC_THREAD(counting);
            sensitive << clk.pos();
        SC_THREAD(tristate);
            sensitive << outEnable << microsPassed;
    }

    void activating();
    void counting();
    void tristate();
};
```

Source:

```
void timer::activating() {
    timerActive = SC_LOGIC_0;
    while(1) {
        if(startTimer == SC_LOGIC_0 && microsPassed.read().to_uint() == TIMER_TIMEOUT_INTERVAL_US) {
            timerActive = SC_LOGIC_0;
        }
        else if(startTimer == SC_LOGIC_1) {
            while(1) {
                wait();
                if(startTimer == SC_LOGIC_0) {
                    timerActive = SC_LOGIC_1;
                    break;
                }
            }
        }
        wait();
    }
}
void timer::counting() {
    microsPassed = 0;
    while(1) {
        if(timerActive == SC_LOGIC_1) {
            if(microsPassed.read().to_uint() == TIMER_TIMEOUT_INTERVAL_US) {
                // hold the microsPassed as it is, to keep the timeOut on
            }
            else {
                microsPassed = (sc_lv<16>) (microsPassed.read().to_uint() + 1);
            }
        }
        if(startTimer == SC_LOGIC_1) {
            microsPassed = (sc_lv<16>) 0;
        }
        wait();
    }
}
```

```
void timer::tristate() {
    while(1) {
        if(outEnable == SC_LOGIC_1){
            if((timerActive == SC_LOGIC_1) && (microsPassed.read().to_uint() == TIMER_TIMEOUT_INTERVAL_US)) {
                timeOut.write(TIMER_TIMEOUT_SET_VALUE);
            }
            else {
                timeOut.write(TIMER_TIMEOUT_CLEAR_VALUE);
            }
        }
        else {
            timeOut.write(TIMER_TIMEOUT_Z_VALUE);
        }
        wait();
    }
}
```

Bus interface:

Header:

```
SC_MODULE(busInterface) {
    sc_in <sc_logic> readIO;
    sc_in <sc_logic> writeIO;
    sc_in <sc_lv<16>> addrBus;

    sc_out <sc_logic> timerEnable;
    sc_out <sc_logic> sensorEnable;
    sc_out <sc_logic> startTimer;

    SC_CTOR(busInterface) :
        readIO("readIO"),
        writeIO("writeIO"),
        addrBus("addrBus"),
        timerEnable("timerEnable"),
        sensorEnable("sensorEnable"),
        startTimer("startTimer") {

        SC_THREAD(timerDecoder);
            sensitive << readIO << writeIO << addrBus;
        SC_THREAD(sensorDecoder);
            sensitive << readIO << writeIO << addrBus;
    }

    void timerDecoder();
    void sensorDecoder();
};
```

Source:

```
void busInterface::timerDecoder() {
    while(1) {
        timerEnable = SC_LOGIC_0;
        startTimer = SC_LOGIC_0;
        if((addrBus.read().to_uint() & BUS_ADDRESS_IO_MASK) == (TIMER_BASE_ADDRESS & BUS_ADDRESS_IO_MASK)) {
            if(readIO == SC_LOGIC_1) {
                timerEnable = SC_LOGIC_1;
            }
            else if(writeIO == SC_LOGIC_1) {
                startTimer = SC_LOGIC_1;
            }
        }
        wait();
    }
}
```

```
void busInterface::sensorDecoder() {
    while(1) {
        sensorEnable = SC_LOGIC_0;
        if((addrBus.read().to_uint() & BUS_ADDRESS_IO_MASK) == (SENSOR_BASE_ADDRESS & BUS_ADDRESS_IO_MASK)) {
            if(readIO == SC_LOGIC_1) {
                sensorEnable = SC_LOGIC_1;
            }
        }
        wait();
    }
}
```

Processor:

Header:

```
SC_MODULE(processor) {
    sc_in <sc_logic> clk;

    sc_out <sc_logic> readIO;
    sc_out <sc_logic> writeIO;
    sc_out <sc_lv<16>> addrBus;

    sc_inout_rv <16> dataBus;

    SC_CTOR(processor) :
        clk("clk"),
        readIO("readIO"),
        writeIO("writeIO"),
        addrBus("addrBus"),
        dataBus("dataBus") {

        SC_THREAD(bracketing);
            sensitive << clk.pos();
        //dont_initialize();
    }

    void bracketing();
};
```

Source:

```
void processor::bracketing() {
    std::vector<int> dataFrame(100);
    wait();
    while(1) {
        for(int iter = 0; iter < 100; iter++) {
            // starting the timer
            writeIO = SC_LOGIC_1;
            readIO = SC_LOGIC_0;
            addrBus = TIMER_BASE_ADDRESS;
            wait();

            // waiting for the 1ms interval to pass
            writeIO = SC_LOGIC_0;
            readIO = SC_LOGIC_1;
            wait();
            while(dataBus.read().to_uint() != 1) {
                wait();
            }

            //reading the sensor value
            writeIO = SC_LOGIC_0;
            readIO = SC_LOGIC_1;
            addrBus = SENSOR_BASE_ADDRESS;
            wait();
```

```
            dataFrame[iter] = dataBus.read().to_int();
        }
        double avg = std::accumulate(dataFrame.begin(), dataFrame.end(), 0.0) / dataFrame.size();
        std::cout << "average = " << avg << endl;
    }
}
```

Main:

Header:

```
#include <systemc.h>
#include "squareWave.h"
#include "sensorFrontend.h"
#include "clockGenerator.h"
#include "regizter.h"
#include "timer.h"
#include "pulseGenerator.h"
#include "busInterface.h"
#include "processor.h"
#include "interconnect.h"
```

Source:

```
    // signal instantiation
    sc_signal <double>              squareWaveSig;
    sc_signal <sc_logic>            clk;
    sc_signal <sc_logic>            readIO;
    sc_signal <sc_logic>            writeIO;
    sc_signal <sc_logic>            sensorEnable;
    sc_signal <sc_logic>            timerEnable;
    sc_signal <sc_logic>            startTimer;
    sc_signal <sc_lv<16>>           sensorOut;
    sc_signal <sc_lv<16>>           addrBus;
    sc_signal <sc_lv<16>>           timeOut;
    sc_signal <sc_lv<16>>           registerOut;
    sc_signal_rv <16>               dataBus;
```

```
    // component instantiation
    squareWave                      squareWaveUnit(                 "squareWave_inst");
    sensorFrontend                  sensorFrontendUnit(             "sensorFrontend_inst");
    clockGenerator                  clockGeneratorUnit(             "clockGenerator_inst");
    regizter                        registerUnit(                   "register_inst");
    timer                           timerUnit(                      "timer_inst");
    busInterface                    busInterfaceUnit(               "busInterface_inst");
    processor                       processorUnit(                  "processor_inst");
```

```
    //creating connections
    squareWaveUnit.out(             squareWaveSig);
    clockGeneratorUnit.out(         clk);

    sensorFrontendUnit.in(          squareWaveSig);
    sensorFrontendUnit.out(         sensorOut);

    registerUnit.clk(               clk);
    registerUnit.outEnable(         sensorEnable);
    registerUnit.in(                sensorOut);
    registerUnit.out(               dataBus);

    timerUnit.clk(                  clk);
    timerUnit.startTimer(           startTimer);
    timerUnit.outEnable(            timerEnable);
    timerUnit.timeOut(              dataBus);

    busInterfaceUnit.readIO(        readIO);
    busInterfaceUnit.writeIO(       writeIO);
    busInterfaceUnit.addrBus(       addrBus);
```

```
    busInterfaceUnit.timerEnable(   timerEnable);
    busInterfaceUnit.sensorEnable(  sensorEnable);
    busInterfaceUnit.startTimer(    startTimer);

    processorUnit.clk(              clk);
    processorUnit.dataBus(          dataBus);
    processorUnit.readIO(           readIO);
    processorUnit.writeIO(          writeIO);
    processorUnit.addrBus(          addrBus);
```

```
    // tracing signals
    sca_util::sca_trace(trace,      squareWaveUnit.out,                "squareWave");
    sca_util::sca_trace(trace,      clockGeneratorUnit.out,            "clk");

    sca_util::sca_trace(trace,      sensorFrontendUnit.sineSignal,     "lowPassFilterOut");
    sca_util::sca_trace(trace,      sensorFrontendUnit.adcSineSignal,  "adcOut");
    sca_util::sca_trace(trace,      sensorFrontendUnit.out,            "tdf2DeOut");

    sca_util::sca_trace(trace,      registerUnit.outEnable,            "sensorEnable");

    sca_util::sca_trace(trace,      timerUnit.outEnable,               "timerEnable");
    sca_util::sca_trace(trace,      timerUnit.startTimer,              "startTimer");
    sca_util::sca_trace(trace,      timerUnit.microsPassed,            "microsPassed");
    sca_util::sca_trace(trace,      timerUnit.timerActive,             "timerActive");

    sca_util::sca_trace(trace,      busInterfaceUnit.timerEnable,      "timerEnable");
    sca_util::sca_trace(trace,      busInterfaceUnit.sensorEnable,     "sensorEnable");
    sca_util::sca_trace(trace,      busInterfaceUnit.startTimer,       "startTimer");

    sca_util::sca_trace(trace,      processorUnit.readIO,              "readIO");
    sca_util::sca_trace(trace,      processorUnit.writeIO,             "writeIO");
    sca_util::sca_trace(trace,      processorUnit.addrBus,             "addrBus");
```

```
    sc_start(1200, SC_MS);
```

Results:

Terminal:

```
moein@Vivo:~/Documents/5$ ./executable

        SystemC 2.3.3-Accellera --- Jul  8 2022 08:08:41
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED


        SystemC AMS extensions 2.3.0-COSEDA Release date: 20200312 2138
        Copyright (c) 2010-2014  by Fraunhofer-Gesellschaft IIS/EAS
        Copyright (c) 2015-2020  by COSEDA Technologies GmbH
        Licensed under the Apache License, Version 2.0


Info: SystemC-AMS:
        6 SystemC-AMS modules instantiated
        2 SystemC-AMS views created
        3 SystemC-AMS synchronization objects/solvers instantiated


Info: SystemC-AMS:
        1 dataflow clusters instantiated
          cluster 0:
                3 dataflow modules/solver, contains e.g. module: sca_linear_solver_0 containing modules:
                sensorFrontend_inst.lowpassFilter_inst.converter_de2eln
                sensorFrontend_inst.lowpassFilter_inst.lowpassFilter_res1
                sensorFrontend_inst.lowpassFilter_inst.lowpassFilter_cap1
                sensorFrontend_inst.lowpassFilter_inst.converter_eln2tdf

                3 elements in schedule list,
                100 ns cluster period,
                ratio to lowest:  1             e.g. module: sca_linear_solver_0
                ratio to highest: 1 sample time  e.g. module: sca_linear_solver_0
                1 connections to SystemC de, 1 connections from SystemC de


Warning: (W206) vector contains 4-value logic
In file: ../../../src/sysc/datatypes/bit/sc_proxy.h:1457
In process: busInterface_inst.timerDecoder @ 0 s

Warning: (W206) vector contains 4-value logic
In file: ../../../src/sysc/datatypes/bit/sc_proxy.h:1457
In process: busInterface_inst.sensorDecoder @ 0 s

Info: SystemC-AMS:
        ELN solver instance: sca_linear_solver_0 (cluster 0)
                has 4 equations for 4 modules (e.g. sensorFrontend_inst.lowpassFilter_inst.converter_de2eln),
                0 inputs and 1 outputs to other (TDF) SystemC-AMS domains,
                1 inputs and 0 outputs to SystemC de.
                100 ns initial time step

average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
average = -4.8
moein@Vivo:~/Documents/5$
```

VCD waveform: