Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Systems on Chip Design
Course Number:	COE838
Semester/Year (e.g.F2016)	W2024

Instructor:	Dr. Khan

Assignment/Lab Number:	Lab 1
Assignment/Lab Title:	Introduction to SystemC

Submission Date:	01-24-2024
Due Date:	01-25-2024

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Zelenovic	Danilo	501032542	04	DZ

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Introduction

The purpose of this lab is to become familiarized with SystemC and any other development environments that may be used. The submission required a barrel shifter to be added in series with an ALU which will compute several different operations, based on the values given in the table in the lab manual.

Design

The lab was completed with the use of the provided example functions. The barrel shifter was used to be the barrel shifter, however, the appropriate functions were added such as the enable, left and right inputs. This value was selected to be the B input to the ALU, while the A value was directly transferred. Based on several inputs, the ALU will be turned on or off, to allow either a subtraction, addition, or a simple barrel shift with no further computation. From there, the makefile was also edited, simply to add every required file to be compiled and run. The alu code was created as well. The alu code is in charge of controlling, of course, the ALU. The ALU code required includes the opcode, which will determine which operation is being done between A and B; an addition, or a subtraction. The signals required were all initialized in order for the data to be able to communicate. Next, required ports were declared as they were requirements to determine entries to modules. The modules were also created, where the majority of the code is. In the modules, the requirements for the shifter and ALU were incorporated. All of the opcode and functionality was created, as specified by the lab manual. For instance, 0 opcode would cause the ALU to do an addition, while 1 would cause subtraction. From there, the waveforms were also generated for the multiple test cases provided. All results can be seen below.

Results

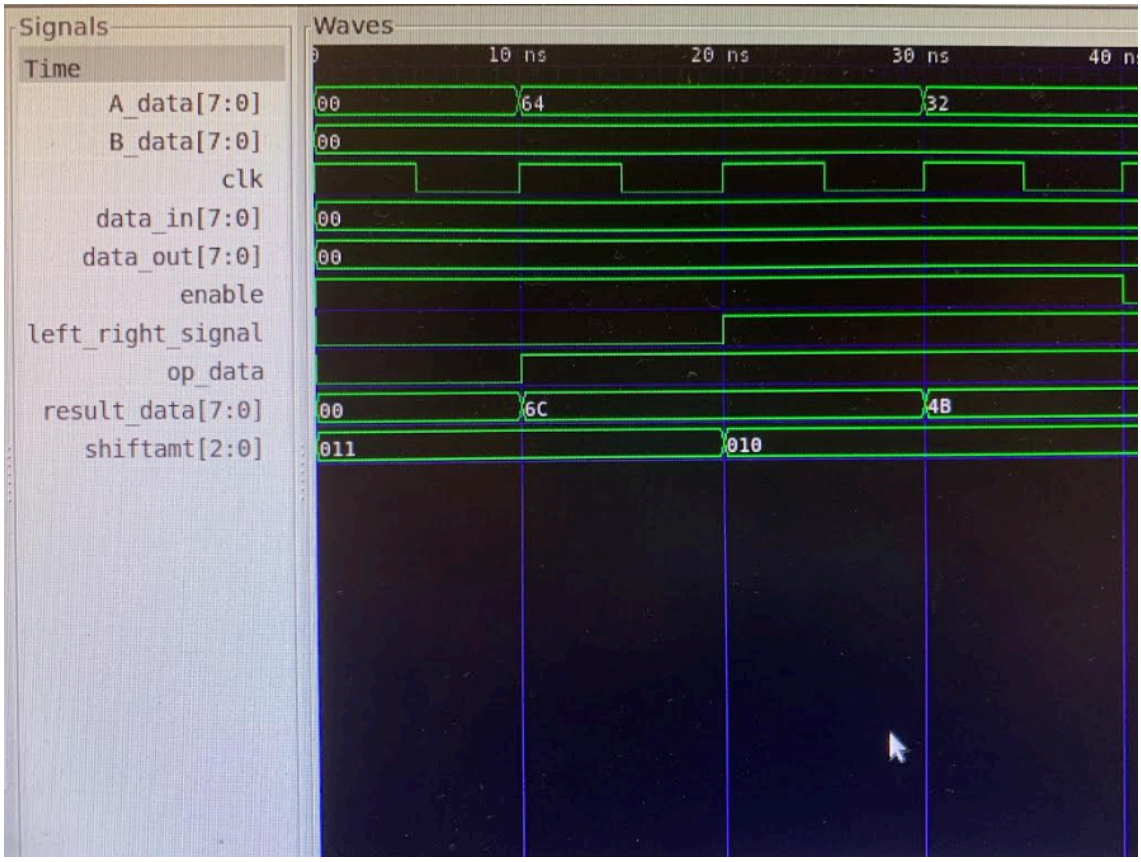

Figure 1. Waveform output

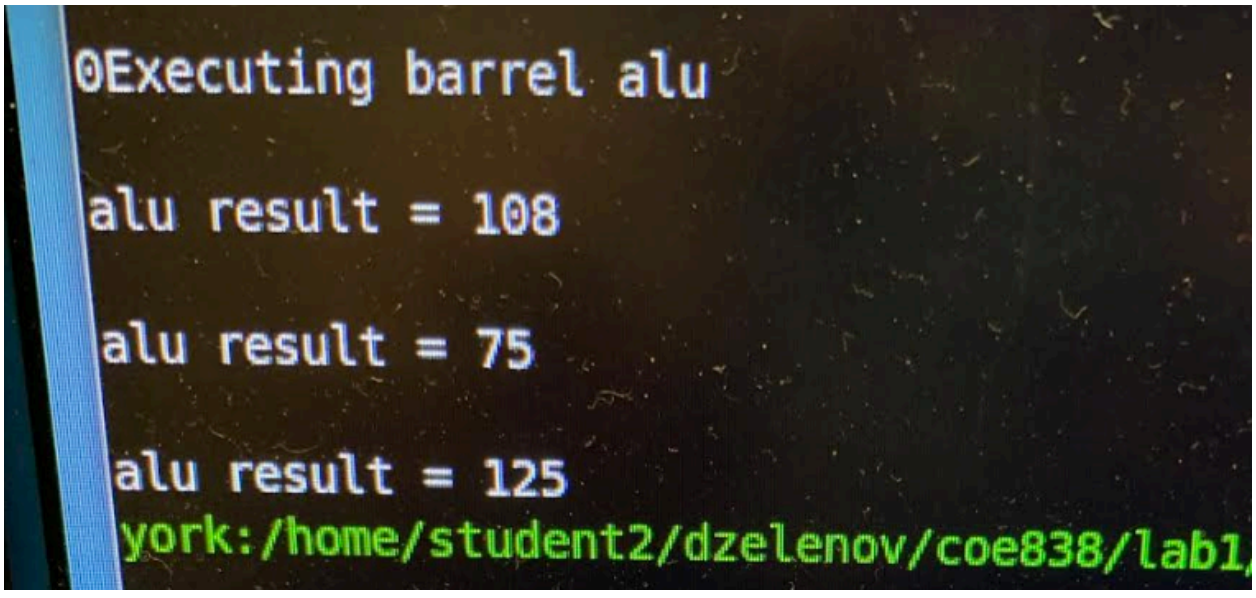

Figure 2. Command line output execution values

Appendix

Alu.cpp

```
#include "alu.h"

void ALUModule::performALUOperation() {
    sc_int<8> resultData;

    switch (operation.read()) {
        case 0:
            resultData = operandA.read() - operandB.read();
            break;
        case 1:
            resultData = operandA.read() + operandB.read();
            break;
    }

    result.write(resultData);
}
```

Alu.h

```
#ifndef ALU_H
#define ALU_H

#include <systemc.h>

void initializeALU();

SC_MODULE(ALUModule) {
    sc_in<bool> clock;
    sc_in<bool> operation;
    sc_in<sc_int<8>> operandA;
    sc_in<sc_int<8>> operandB;
    sc_out<sc_int<8>> result;

    void performALUOperation();

    SC_CTOR(ALUModule) {
        SC_METHOD(performALUOperation);
        dont_initialize();
        sensitive << clock.pos();
    }
};

#endif
```

Barrelalu.cpp

```
#include "barrelalu.h"

void BarrelALU::performBarrelALUOperation() {
    sc_int<8> resultData = barrelShiftOut.read();
    B.write(resultData);
}

void BarrelALU::initialize() {
    barrelShift.leftRight(leftRight);
    barrelShift.shiftAmount(shiftAmount);
    barrelShift.clock(clock);
    barrelShift.enable(enable);
    barrelShift.dataIn(dataIn);
    barrelShift.dataOut(barrelShiftResult);

    aluModule.operation(operation);
    aluModule.operandA(operandA);
    aluModule.operandB(barrelShiftResult);
    aluModule.result(result);
    aluModule.clock(clock);

    SC_METHOD(performBarrelALUOperation);
    dont_initialize();
    sensitive << clock.pos();
}

SC_HAS_PROCESS(BarrelALU);

BarrelALU::BarrelALU(sc_module_name name) : sc_module(name), aluModule("ALUModule"), barrelShift("BarrelShift"),
                                            clock("clock"), enable("enable"), leftRight("leftRight"),
                                            shiftAmount("shiftAmount"), dataIn("dataIn"), dataOut("dataOut"),
                                            operation("operation"), operandA("operandA"), operandB("operandB"), result("result"),
                                            barrelShiftResult("barrelShiftResult") {

    initialize();
}
```

Barrelalu.h

```
#include <systemc.h>
#include "alu.h"
#include "barrelshift.h"

void initializeBarrelALU();

SC_MODULE(BarrelALU) {
    BarrelShift barrelShift;
    ALUModule aluModule;

    sc_in<bool> clock;
    sc_in<bool> enable;
    sc_in<bool> leftRight;
    sc_in<sc_int<3>> shiftAmount;
    sc_in<sc_int<8>> dataIn;
    sc_out<sc_int<8>> dataOut;
    sc_in<bool> operation;
    sc_in<sc_int<8>> operandA;
    sc_inout<sc_int<8>> operandB;
    sc_out<sc_int<8>> result;
    sc_signal<sc_int<8>> barrelShiftResult;

    void performBarrelALUOperation();

    SC_CTOR(BarrelALU) : aluModule("ALUModule"), barrelShift("BarrelShift"),
                         clock("clock"), enable("enable"), leftRight("leftRight"),
                         shiftAmount("shiftAmount"), dataIn("dataIn"), dataOut("dataOut"),
                         operation("operation"), operandA("operandA"), operandB("operandB"), result("result"),
                         barrelShiftResult("barrelShiftResult") {

        barrelShift.leftRight(leftRight);
        barrelShift.shiftAmount(shiftAmount);
        barrelShift.clock(clock);
        barrelShift.enable(enable);
        barrelShift.dataIn(dataIn);
        barrelShift.dataOut(barrelShiftResult);

        aluModule.operation(operation);
        aluModule.operandA(operandA);
        aluModule.operandB(barrelShiftResult);
        aluModule.result(result);
        aluModule.clock(clock);

        SC_METHOD(performBarrelALUOperation);
        dont_initialize();
        sensitive << clock.pos();
    }
};

#endif
```

Sc_main.cpp

```
#include <systemc.h>
#include "barrelshift.h"
#include "alu.h"
#include "barrelalu.h"

void initializeBarrelALU();

int sc_main(int argc, char* argv[]) {
    initializeBarrelALU();
    return 0;
}

void initializeBarrelALU() {
    sc_trace_file *traceFile;  // Create VCD file for tracing

    // ALU SIGNALS
    sc_signal<bool> operationSignal;
    sc_signal<sc_int<8>> operandASignal, operandBSignal, resultSignal;
    sc_clock clock("clock", 10, SC_NS, 0.5);  // Create a clock signal

    // BARRELSHIFT SIGNALS
    sc_signal<bool> enableSignal, leftRightSignal;
    sc_signal<sc_int<3>> shiftAmountSignal;
    sc_signal<sc_int<8>> dataOutSignal, dataInSignal;

    // BIND BARRELALU SIGNALS TO PORTS
    BarrelALU barrelALU("BarrelALU");
    barrelALU.clock(clock);
    barrelALU.enable(enableSignal);
    barrelALU.leftRight(leftRightSignal);
    barrelALU.shiftAmount(shiftAmountSignal);
    barrelALU.dataIn(dataInSignal);
    barrelALU.dataOut(dataOutSignal);
    barrelALU.operation(operationSignal);
    barrelALU.operandA(operandASignal);
    barrelALU.operandB(operandBSignal);
    barrelALU.result(resultSignal);

    // Create wave file and trace the signals executing
    traceFile = sc_create_vcd_trace_file("trace_file");
    traceFile->set_time_unit(1, SC_NS);
    sc_trace(traceFile, clock, "clock");
    sc_trace(traceFile, enableSignal, "enable");
    sc_trace(traceFile, shiftAmountSignal, "shiftAmount");
    sc_trace(traceFile, leftRightSignal, "leftRightSignal");
    sc_trace(traceFile, dataOutSignal, "dataIn");
    sc_trace(traceFile, dataOutSignal, "dataOut");

    cout << "\nExecuting Barrel ALU" << endl;

    // TEST CASE 1
    enableSignal.write(1);  // initialize
    shiftAmountSignal.write(3);
    dataInSignal.write(1);
    leftRightSignal.write(0);
    sc_start(10, SC_NS);

    cout << endl;

    operationSignal.write(1);
    operandASignal.write(100);

    sc_start(10, SC_NS);
    cout << "ALU result = " << resultSignal.read() << endl;

    // TEST CASE 2
    enableSignal.write(1);  // initialize
    shiftAmountSignal.write(2);
    dataInSignal.write(100);
    leftRightSignal.write(1);
    sc_start(10, SC_NS);

    cout << endl;

    operationSignal.write(1);
    operandASignal.write(50);

    sc_start(10, SC_NS);
    cout << "ALU result = " << resultSignal.read() << endl;

    // TEST CASE 3
    enableSignal.write(0);  // initialize
    shiftAmountSignal.write(2);
    dataInSignal.write(100);
    leftRightSignal.write(1);
    sc_start(10, SC_NS);

    cout << endl;

    operationSignal.write(1);
    operandASignal.write(25);

    sc_start(10, SC_NS);
    cout << "ALU result = " << resultSignal.read() << endl;

    sc_close_vcd_trace_file(traceFile);
}
```