Lab Report

Name: Dang Hung Thinh ID: 22119137

Class: Monday morning class (1-5)

Note: every individual must submit a unique lab report form.

1. Summarize (in your own words) the subject of this lab:

This lab focused on designing and simulating two digital components using SystemC: a 4-bit up/down counter with control signals, and an 8-bit parallel load/shift register. The goal was to understand how digital components behave under various control conditions and to validate their operation using testbenches and waveform simulations.

2. Describe the new concepts covered in this lab:

- Asynchronous control and parameterized behavior in digital circuits.
- Implementing bidirectional counting based on a control signal.
- Designing a register that can load data in parallel and shift data left/right.
- Writing comprehensive testbenches for SystemC modules.
- Visualizing digital signal transitions using GTKWave with VCD trace files.

3. Describe how this lab built upon previous ones:

This lab built on earlier SystemC labs where basic modules and signal manipulations were introduced. It extended those principles to implement more complex control logic, and it emphasized modular design and reusability. Additionally, it expanded the use of simulation tools to better understand timing behavior.

4. Describe the most difficult part of this lab for you:

The most challenging part was ensuring that the control signals (PE, UD, BD) correctly affected the counter behavior in all scenarios. It required careful planning and simulation to catch logic errors that weren't obvious from code alone. Writing a clear testbench that covers all use cases was also tricky.

5. Describe problems you faced and how you solved them:

One problem was that the counter didn't count properly when BD was set to 0 — it exceeded 9. I realized I needed to include logic to reset the counter to 0 once it reaches the upper limit (9 or 15 depending on the mode). I fixed it by adding condition checks before incrementing or decrementing. For the shift register, I initially forgot to handle the "hold" condition, which I fixed by preserving the output when *sh == 3.*

6. **Do you verify that the code included with this report is your's original work (yes/no)?**

YES

7. **Submit your source code, testbench, and simulation output.**

7.1. Up/down counter

- **LIBRARY:**

```
//============================
// Files: libudcounter.h
//============================
#ifndef UPDOWN_COUNTER_H
#define UPDOWN_COUNTER_H

#include <systemc.h>

SC_MODULE(UpDownCounter) {
  sc_in<bool> clk;
  sc_in<bool> pe;
  sc_in<bool> ud;
  sc_in<bool> bd;
  sc_in<sc_uint<4>> p;
  sc_out<sc_uint<4>> q;

  sc_uint<4> count;

  void process() {
    if (pe.read()) {
      count = p.read();
    } else if (clk.posedge()) {
      if (ud.read()) {
```

```
            // Count Up
            if (bd.read() == 0 && count < 9) count++;
            else if (bd.read() == 1 && count < 15) count++;
          } else {
            // Count Down
            if (count > 0) count--;
          }
        }
        q.write(count);
    }

    SC_CTOR(UpDownCounter) : count(0) {
        SC_METHOD(process);
        sensitive << clk.pos();
    }
};

#endif
```

- **SOURCE CODE:**

```
//=========================
// File: udcouter.cpp
//=========================
#include "libudcounter.h"
```

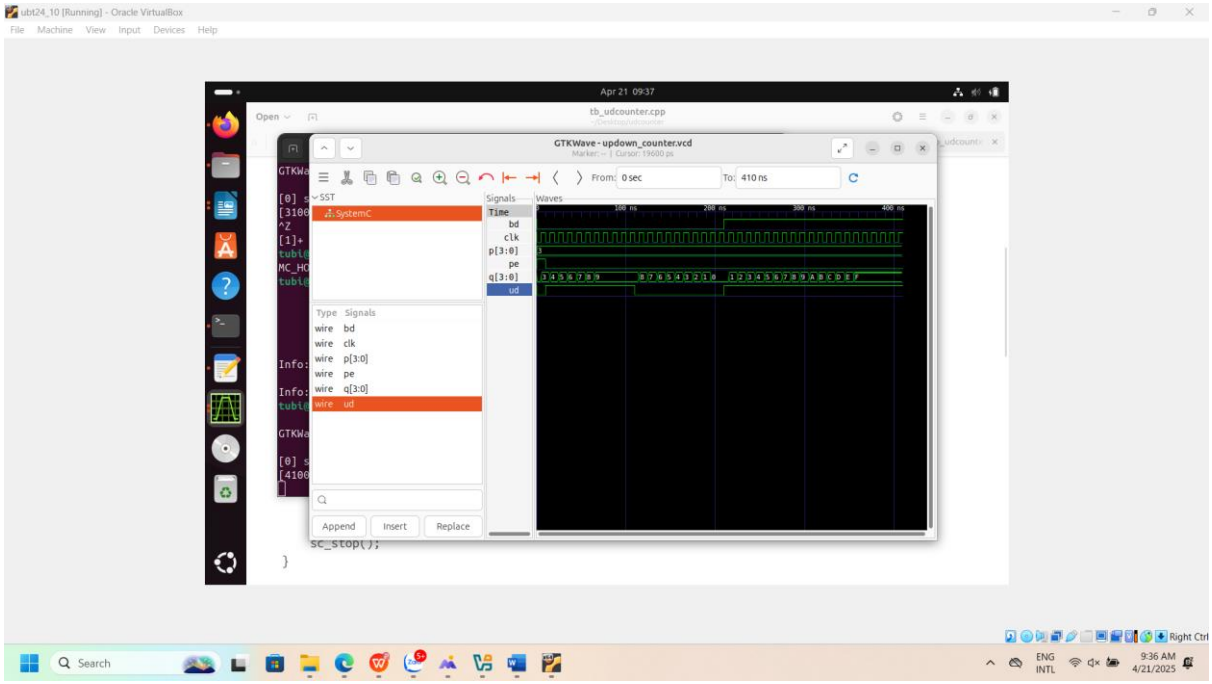
- **TEST BENCH:**

```
//============================
// Up/Down Counter Testbench
//============================
#include <systemc.h>
#include "libudcounter.h"

SC_MODULE(UpDownCounterTB) {
   sc_signal<bool> clk, pe, ud, bd;
   sc_signal<sc_uint<4>> p;
   sc_signal<sc_uint<4>> q;

   UpDownCounter* counter;

   void clk_gen() {
      while (true) {
         clk.write(false);
         wait(5, SC_NS);
         clk.write(true);
         wait(5, SC_NS);
      }
   }

   void stim() {
      pe.write(1);
      p.write(0x3);
      wait(10, SC_NS);
```

```
    pe.write(0);
    ud.write(1);
    bd.write(0);
    for (int i = 0; i < 10; i++) wait(10, SC_NS);

    ud.write(0);
    for (int i = 0; i < 10; i++) wait(10, SC_NS);

    bd.write(1);
    ud.write(1);
    for (int i = 0; i < 20; i++) wait(10, SC_NS);

    sc_stop();
  }

  SC_CTOR(UpDownCounterTB) {
    counter = new UpDownCounter("counter");
    counter->clk(clk);
    counter->pe(pe);
    counter->ud(ud);
    counter->bd(bd);
    counter->p(p);
    counter->q(q);

    SC_THREAD(clk_gen);
    SC_THREAD(stim);
  }
};
```

```
int sc_main(int argc, char* argv[]) {
   UpDownCounterTB tb("tb");
   sc_trace_file* tf = sc_create_vcd_trace_file("updown_counter");

   sc_trace(tf, tb.clk, "clk");
   sc_trace(tf, tb.pe, "pe");
   sc_trace(tf, tb.ud, "ud");
   sc_trace(tf, tb.bd, "bd");
   sc_trace(tf, tb.p, "p");
   sc_trace(tf, tb.q, "q");

   sc_start();
   sc_close_vcd_trace_file(tf);
   return 0;
}
```

-SIMUMATION WAVEFORM:



7.2. Parallel Load/Shift Register

- **LIBRARY:**

```
//==========================
// File: libpalsres.h
//==========================
#ifndef SHIFT_REGISTER_H
#define SHIFT_REGISTER_H


#include <systemc.h>


SC_MODULE(ShiftRegister) {
   sc_in<bool> clk;
   sc_in<sc_uint<8>> input;
   sc_in<sc_uint<2>> sh; // 00: shift right, 01: shift left, 10: load, 11: hold
   sc_out<sc_uint<8>> output;
```

```
  sc_uint<8> reg;

  void process();

  SC_CTOR(ShiftRegister) {
    SC_METHOD(process);
    sensitive << clk.pos();
    reg = 0;
  }
};

#endif
```

- **SOURCE CODE:**

```
//==========================
// File: palsres.cpp
//==========================
#include "libpalsres.h"

void ShiftRegister::process() {
  switch (sh.read()) {
    case 0: // shift right
      reg = reg >> 1;
      break;
    case 1: // shift left
      reg = reg << 1;
      break;
    case 2: // load input
```

```
        reg = input.read();
        break;
    case 3: // hold
        // do nothing
        break;
  }
  output.write(reg);
}
```

- **TEST BENCH:**

```
//==============================
// File: tb_palsres.cpp
//==============================
#include <systemc.h>
#include "palsres.h"

SC_MODULE(ShiftRegisterTB) {
  sc_signal<bool> clk;
  sc_signal<sc_uint<8>> input;
  sc_signal<sc_uint<2>> sh;
  sc_signal<sc_uint<8>> output;

  ShiftRegister* reg;

  void clk_gen() {
    while (true) {
      clk.write(false);
      wait(5, SC_NS);
      clk.write(true);
```

```
        wait(5, SC_NS);
    }
}

void stim() {
    input.write(0xA5); // 10100101
    sh.write(2); // load
    wait(10, SC_NS);

    sh.write(1); // shift left
    for (int i = 0; i < 3; i++) wait(10, SC_NS);

    sh.write(0); // shift right
    for (int i = 0; i < 3; i++) wait(10, SC_NS);

    sh.write(3); // hold
    for (int i = 0; i < 3; i++) wait(10, SC_NS);

    sc_stop();
}

SC_CTOR(ShiftRegisterTB) {
    reg = new ShiftRegister("shiftreg");
    reg->clk(clk);
    reg->input(input);
    reg->sh(sh);
    reg->output(output);
```

```
    SC_THREAD(clk_gen);
    SC_THREAD(stim);
  }
};

int sc_main(int argc, char* argv[]) {
  ShiftRegisterTB tb("tb");

  sc_trace_file* tf = sc_create_vcd_trace_file("shift_register");
  sc_trace(tf, tb.clk, "clk");
  sc_trace(tf, tb.input, "input");
  sc_trace(tf, tb.sh, "sh");
  sc_trace(tf, tb.output, "output");

  sc_start();
  sc_close_vcd_trace_file(tf);
  return 0;
}
```

mode	Input[7:0]	Present-state	Output[7:0]	Description
10	10100101 (A5)	00000000 (00)	10100101 (A5)	Load mode
01	10100101 (A5)	10100101 (A5)	01001010 (4A)	Shift left
01	01001010 (4A)	01001010 (4A)	10010100 (94)	Shift left
01	10010100 (94)	10010100 (94)	00101000 (28)	Shift left
00	00101000 (28)	00101000 (28)	00010100 (14)	Shift right
00	00010100 (14)	00010100 (14)	00001010 (0A)	Shift right
00	00001010 (0A)	00001010 (0A)	00000101 (05)	Shift right
11	00000101 (05)	00000101 (05)	00000101 (05)	Hold

-SIMUMATION: