Homework 3 – C++ RT-level Design and Modeling

Taghizadeh,
810198373

I. Introduction

In this homework, we are about to make a 64-bit Adder from a 4-bit Adder. The algorithm is to split inputs into 16 packets of 4 bits and in 16 clock cycles, add each packet of inputs. So in the each cycle 4 bits of the *sum* output will be calculated. Carry out of the each addition of the 4-bits packets is used as carry in of the next partial addition. By repeating this about 16 times, the entire *sum* output and carry out will be calculated.

II. HOMEWORK PARTS REPORT

A. 64-bit Adder's Data Path Schematic

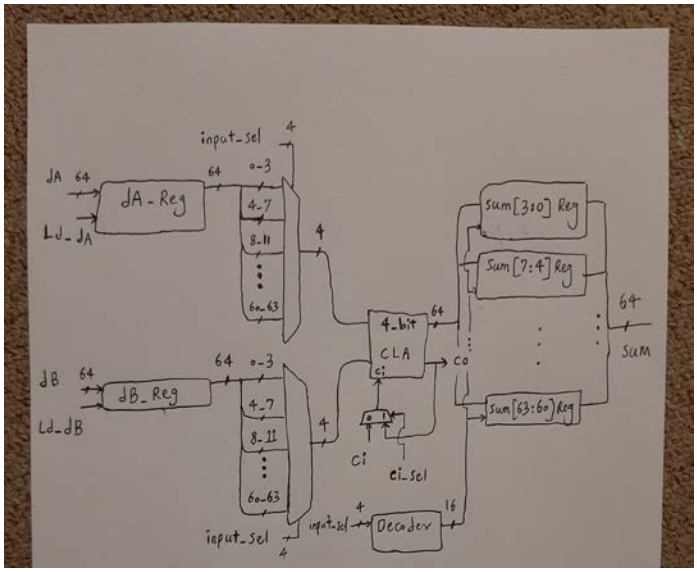

Fig. 1 64-bit Adder Data Path Schematic

B. 64-bit Adder's Data Path in C++

First by the *range()* method of the *bus* class, 4 bits of *a* and *b* inputs are selected by the *internal_4_bits_adder_inputs_selector* to added by the internal 4-bit adder. At first the carry in of the internal 4-bit adder is equal to the carry in of the 64-bit adder. Then in the next cycles, the carry in of the internal 4-bit adder will be equal to the carry out of the previous addition which is selected by the *internal_4_bits_adder_ci_selector* signal. After addition, 4 bits of the *sum* output of the 64-bit adder is ready. So by the *fill_given_range()* method of the *bus* method, 4 bits of the *sum* output will be filled by the sum result of the current addition.

```
5 class Adder_64_bits_DataPath{
6       bus *a, *b;
7       bus *ci;
8       bus *internal_4_bits_adder_inputs_selector;
9       bus *internal_4_bits_adder_ci_selector;
10      bus *co;
11      bus *sum;
12      public:
13              Adder_64_bits_DataPath(bus& a, bus& b, bus& ci, bus&
  internal_4_bits_adder_inputs_selector, bus&
  internal_4_bits_adder_ci_selector, bus& co, bus& sum);
14              void evl ();
15 };
16
```

Fig. 2 64-bit Adder Data Path Declaration

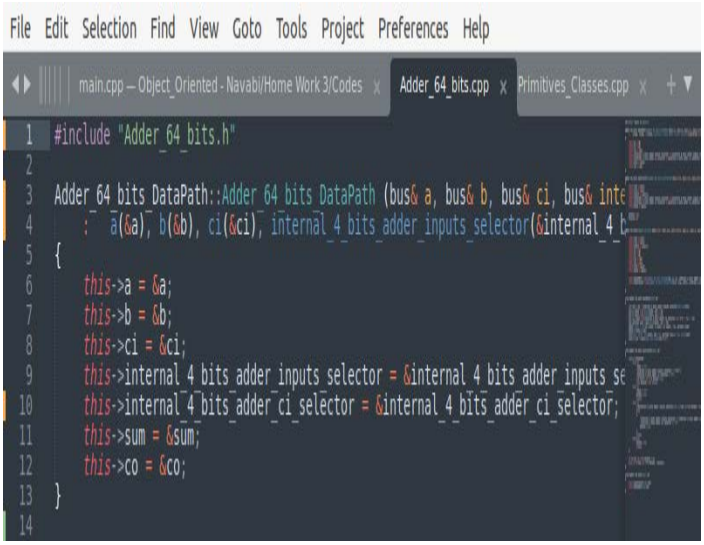

Fig. 3 64-bit Adder Data Path Constructor

```
55      void fill_given_range(int begin_index, int end_index, string s)
56      {
57              int end = v.length() - 1 - begin_index;
58              int begin = v.length() - 1 - end_index;
59              for(int i = begin; i <= end; i++)
60                      v.at(i) = s.at(i - begin);
61      }
62
63      string get_bus_values()
64      {
65              return v;
66      }
67
68      int to_uint()
69      {
70              int res = 0;
71              for(int i = 0; i < v.length(); i++){
72                      if(v.at(i) == '1')
73                              res += int(pow(2, v.length() - 1 -
  i));
74              }
75              return res;
76      }
```

Fig. 4 64-bit Adder Data Path Utility Bus Methods

```
45  void Adder_64_bits_DataPath::evl ()
46  {
47      int sel = (4 * internal_4_bits_adder_inputs_selector->to_uint());
48      bus a_input = a->range(sel, sel + 3);
49      bus b_input = b->range(sel, sel + 3);
50      bus ci_input = (*internal_4_bits_adder_ci_selector == "0") ? *ci : *co
51      bus partial_sum = sum->range(sel, sel + 3);
52      Adder adder_4_bits(a_input, b_input, ci_input, *co, partial_sum);
53      adder_4_bits.evl();
54      sum->fill_given_range(sel, sel + 3, partial_sum.get_bus_values());
55  }
56
```

Fig. 5 64-bit Adder Data Path Evaluation Method

C. 64-bit Adder's Controller Schematic

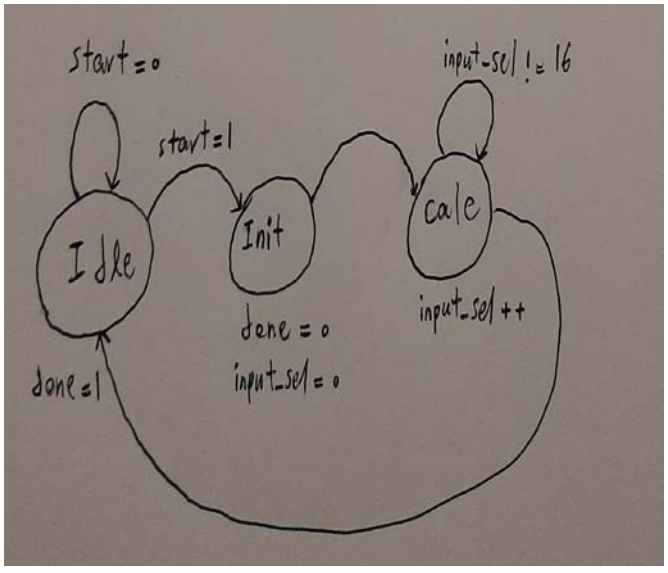

Fig. 6 64-bit Adder Controller Schematic

D. 64-bit Adder's Controller in C++

In the first state, *internal_4_bits_adder_inputs_selector*, *internal_4_bits_adder_ci_selector*, are initialized by 0. If the *start* signal is asserted we will go to the next state, else we will stay on the first state and the *done* signal will remain 1. In the second state the *done* signal is deactivated. In the final state *internal_4_bits_adder_inputs_selector is increased*. This state will be repeated about 16 times to calculate al l 4 packets of the *sum* output. By the end of these repetitions the *sum* output will be calculated, so the *done* signal is asserted and we will get back to the fist state. During the final state the *internal_4_bits_adder_ci_selector* signal will remain 1 to cascade the carry out of each partial addition to the carry in of the next partial addition to calculate the *sum* output correctly.

```
17 class Adder_64_bits_Controller{
18      bus *rst;
19      bus *clk;
20      bus *start;
21      bus *done;
22      bus *internal_4_bits_adder_inputs_selector;
23      bus *internal_4_bits_adder_ci_selector;
24      int Nstate, Pstate;
25      public:
26              Adder_64_bits_Controller(bus& rst, bus& clk, bus& start,
  bus& internal_4_bits_adder_inputs_selector, bus&
  internal_4_bits_adder_ci_selector, bus& done);
27              void evl ();
28 };
29
```

Fig. 7 64-bit Adder Controller Declaration

```
15  Adder_64_bits_Controller::Adder_64_bits_Controller (bus& rst, bus& clk, bus& start
16  {
17      this->rst = &rst;
18      this->clk = &clk;
19      this->start = &start;
20      this->internal_4_bits_adder_inputs_selector = &internal_4_bits_adder_inputs_se
21      this->internal_4_bits_adder_ci_selector = &internal_4_bits_adder_ci_selector;
22      this->done = &done;
23
24      Nstate = 0;
25      Pstate= 0;
26  }
27
```

Fig. 8 64-bit Adder Controller Constructor

```
void Adder_64_bits_Controller::evl ()
{
    switch (Pstate){
        case 0:
            *done = "1";
            *internal_4_bits_adder_inputs_selector = "0000";
            *internal_4_bits_adder_ci_selector = "0";
            if(*start == "1") Nstate = 1;
            else Nstate = 0; break;
        case 1:
            *done = "0";
            Nstate = 2;
            break;
        case 2:
            if(internal_4_bits_adder_inputs_selector->to_uint() == 17) Nstate = 0;
            else
            {
                *internal_4_bits_adder_inputs_selector = *internal_4_bits_adder_in
                *internal_4_bits_adder_ci_selector = "1";
                Nstate = 2;
            }
            break;
        default:
            Pstate = 0;
            break;
    }

    if (*rst == "1") Pstate = 0;
    else if (*clk == "P") Pstate = Nstate;
}
```

Fig. 9 64-bit Adder Controller Evaluation Method

E. 64-bit Adder's Top Level in C++

The 64-bit Adder is simply made by putting the DataPath and the Controller together. To evaluate the addition, first the Controller and then the DataPath is evaluated. An instance of DataPath and Controller is taken in the constructor of the *Adder_64_bits* class.

```
class Adder_64_bits{
        Adder_64_bits_DataPath *DataPath;
        Adder_64_bits_Controller *Controller;
        bus *rst, *clk;
        bus *start;
        bus *a, *b;
        bus *ci;
        bus internal_4_bits_adder_inputs_selector;
        bus internal_4_bits_adder_ci_selector;
        bus *co;
        bus *sum;
        bus *done;
        public:
                Adder_64_bits(bus& rst, bus& clk, bus& start, bus& a, bus&
  b, bus& ci, bus& co, bus& sum, bus& done);

                void set_rst(bus *given_rst) { rst = given_rst; };
                void set_clk(bus *given_clk) { clk = given_clk; };
                void set_start(bus *given_start) { start = given_start; };

                void evl ();
};
```

C/ObjC Header Tab Width: 8 Ln 33, Col 24 INS

Fig. 10 64-bit Adder Declaration

```
Adder_64_bits::Adder_64_bits (bus& rst, bus& clk, bus& start, bus& a, bus& b, bus&
{
    this->rst = &rst;
    this->clk = &clk;
    this->start = &start;
    this->a = &a;
    this->b = &b;
    this->ci = &ci;
    this->co = &co;
    this->sum = &sum;
    this->done = &done;

    this->DataPath = new Adder_64_bits_DataPath(a, b, ci, internal_4_bits_adder_in
    this->Controller = new Adder_64_bits_Controller(rst, clk, start, internal_4_bi

}
```

Fig. 11 64-bit Adder Constructor

```
void Adder_64_bits::evl ()
{
    this->Controller->evl();
    this->DataPath->evl();
}
```

Line 93, Column 2 Tab Size: 4 C++

Fig. 12 64-bit Adder Evaluation Method

F. 64-bit Adder Test Bench

First *rst* is set to 0. Adders' inputs are generated. Carry in of the Adder is set. *start* is asserted. *sum*, *co* and *done* are set to 0 because at first they are 0. An instance of 64-bit Adder is taken. The sequence of the clock signal is generated. In a loop the clock of the 64-bit Adder is set and Adder is evaluated. This will be repeated for the entire sequence of the clock. At the end, outputs are printed.

```
#include "Adder_64_bits.cpp"

int main()
{
    bus rst = "0";
    bus clk;

    // generate adder's inputs scenario 1
    bus a = "01101000000000000000000000000000000000000000000001100111100001000";
    bus b = "11001000000000000000000000000000000000000000000000000001100001000";

    bus ci = "1";
    bus start = "1";

    // At first sum, co and done are initialized to zero
    bus sum = "00000000000000000000000000000000000000000000000000000000000000000";
    bus co = "0";
    bus done = "0";

    Adder_64_bits adder(rst, clk, start, a, b, ci, co, sum, done);

    // generate sequence of clock signal
    string clock = "NPNPNPNPNPNPNPNPNPNPNPNPNPNPNP";

    for(int i = 0; i < clock.size(); i++)
    {
        string clock_state = "";
        clk = clock_state + clock.at(i);
        adder.set_clk(&clk);
        adder.evl();
    }

    //print the outputs
    cout << "A input: " << a.get_bus_values()   << "\n\n";
```

Line 1, Column 1 Tab Size: 4 C++

Fig. 13 64-bit Adder Test Bench

Fig. 14 Printing outputs of 64-bit Adder Test Bench

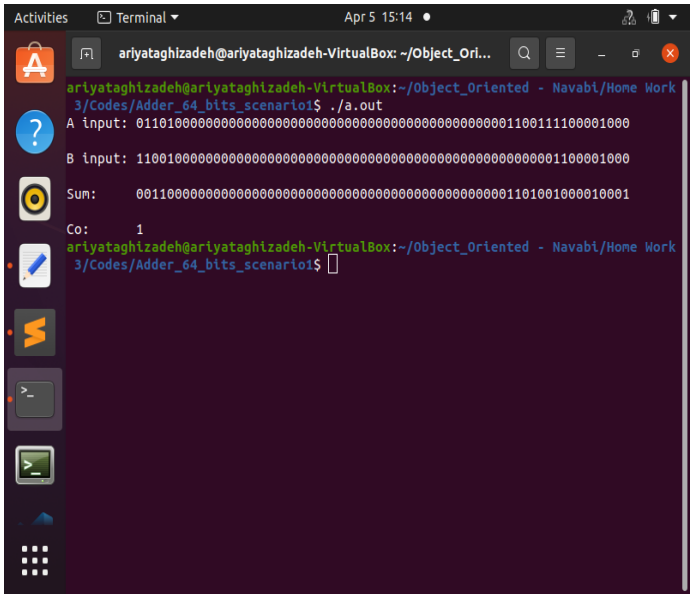

Fig. 15 64-bit Adder Test Bench Scenario 1 Outputs

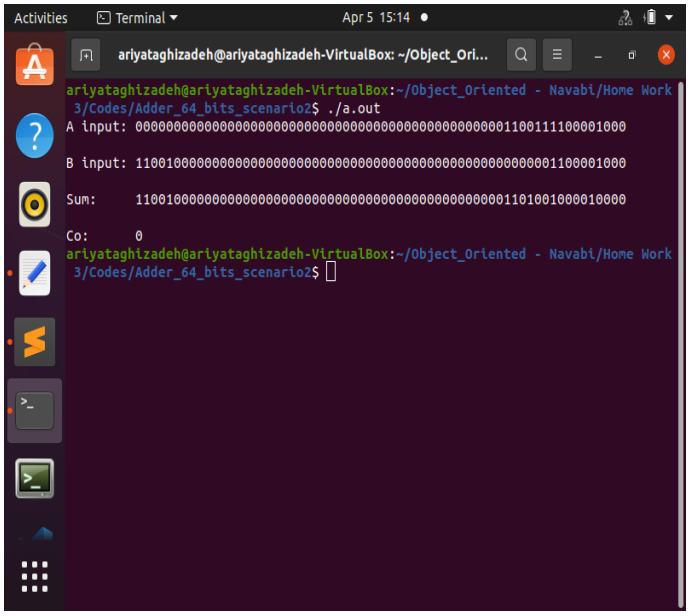

Fig. 16 64-bit Adder Test Bench Scenario 2 Outputs

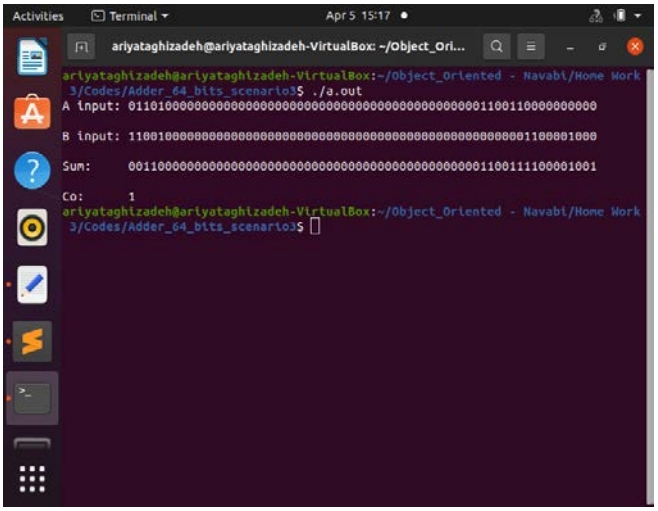

Fig. 17 64-bit Adder Test Bench Scenario 3 Outputs

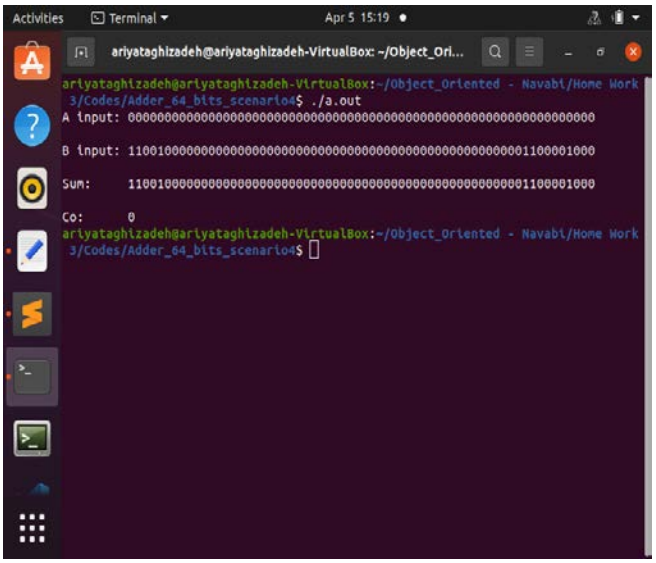

Fig. 18 64-bit Adder Test Bench Scenario 4 Outputs

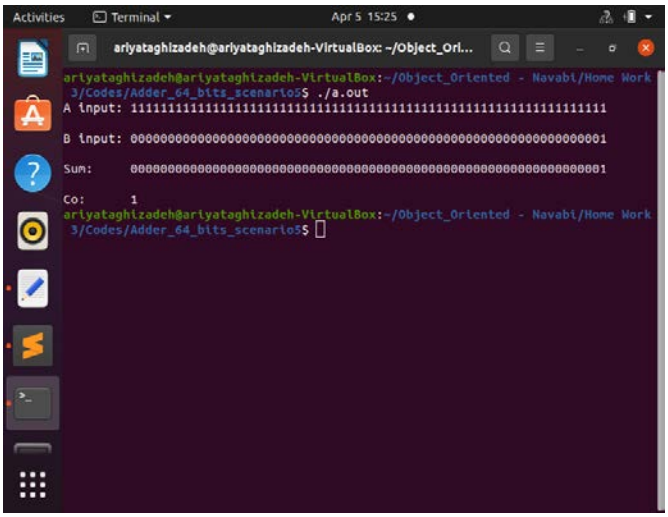

Fig. 19 64-bit Adder Test Bench Scenario 5 Outputs