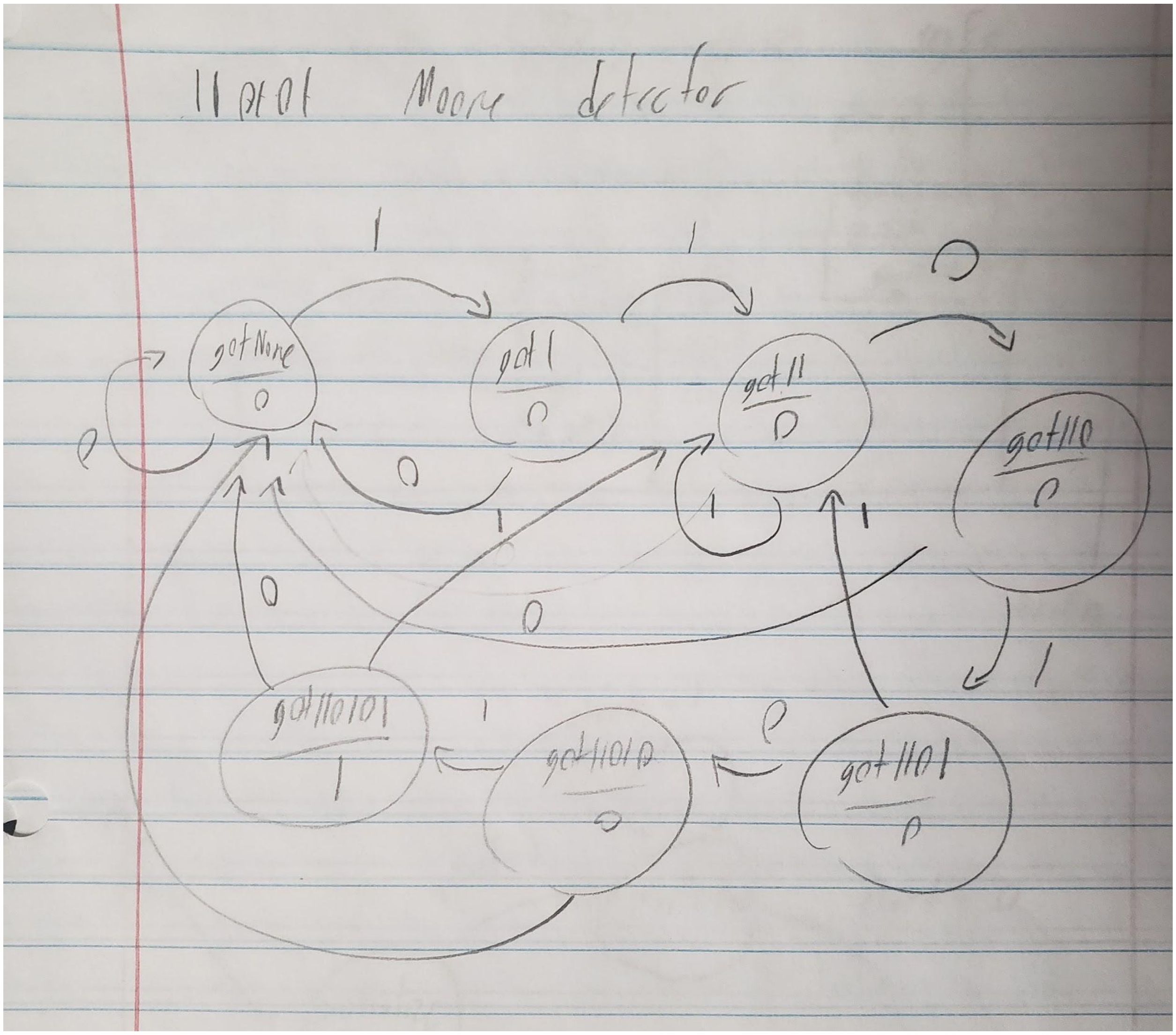
ECE 5723 Midterm

By Bruce Huynh

10/30/2021

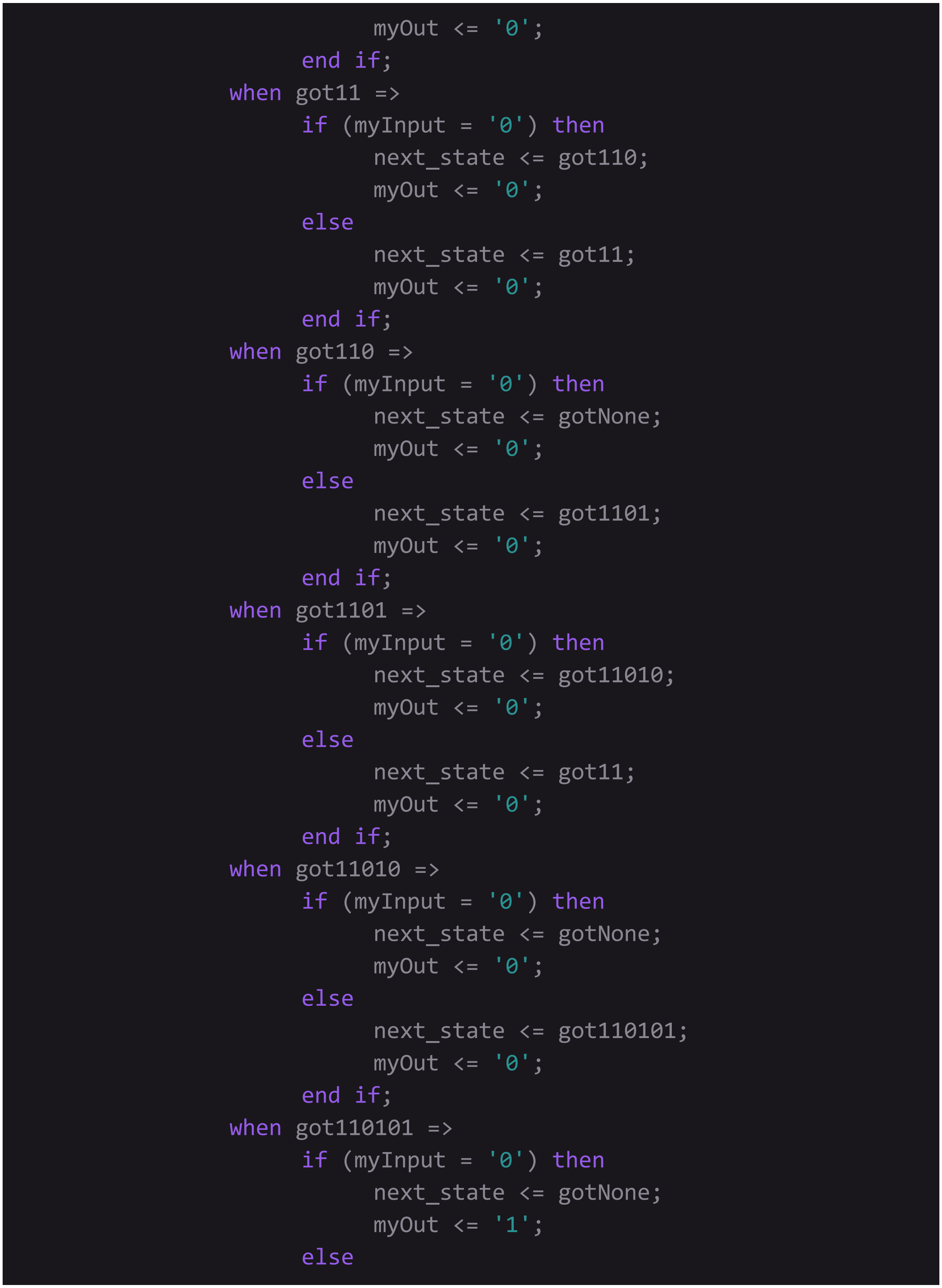
Problem 1:

In this problem, we were to create a simple sequence detector in VHDL that detects the sequence "110101". The following is the state diagram for the sequence detector:

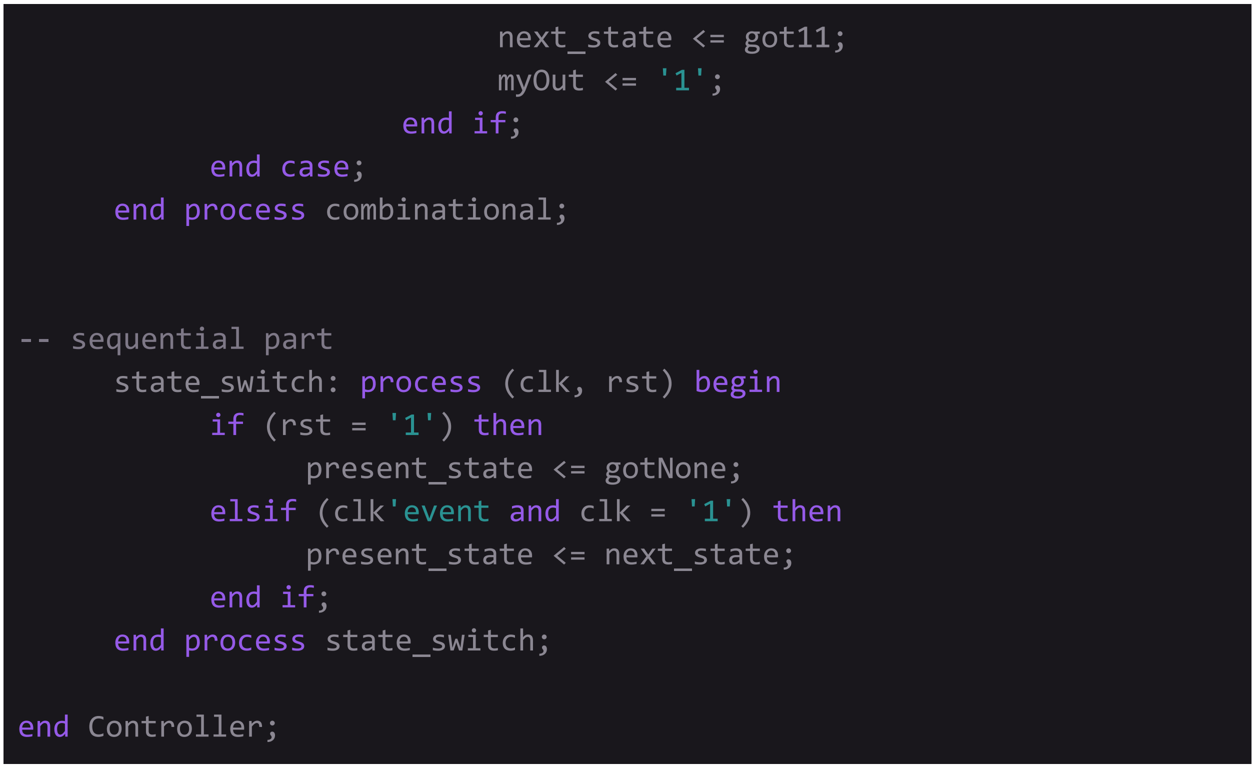


This state diagram is then used to create the sequence detector in VHDL:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sequence_detector is port(
      myInput: in std_logic;
      clk: in std_logic;
      rst: in std_logic;
      myOut: out std_logic);
end sequence_detector;

architecture Controller of sequence_detector is
      type states is (gotNone, got1, got11, got110, got1101,
got11010, got110101);
      signal present_state : states;
      signal next_state : states;

begin
-- combinational part
      combinational: process(myInput, present_state) begin
            case present_state is
                  when gotNone =>
                        if (myInput = '0') then
                              next_state <= gotNone;
                              myOut <= '0';
                        else
                              next_state <= got1;
                              myOut <= '0';
                        end if;

                  when got1 =>
                        if (myInput = '0') then
                              next_state <= gotNone;
                              myOut <= '0';
                        else
                              next_state <= got11;
```

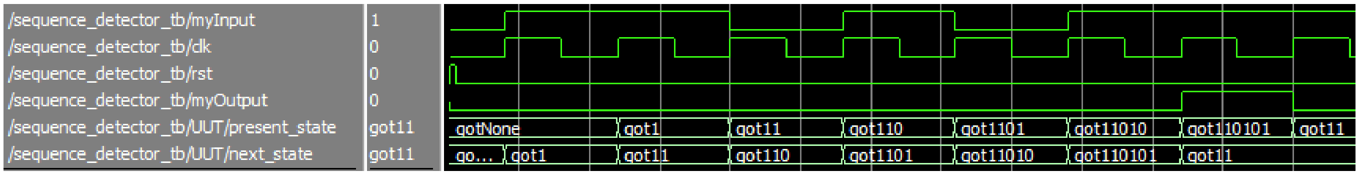


```
                myOut <= '0';
            end if;
        when got11 =>
            if (myInput = '0') then
                next_state <= got110;
                myOut <= '0';
            else
                next_state <= got11;
                myOut <= '0';
            end if;
        when got110 =>
            if (myInput = '0') then
                next_state <= gotNone;
                myOut <= '0';
            else
                next_state <= got1101;
                myOut <= '0';
            end if;
        when got1101 =>
            if (myInput = '0') then
                next_state <= got11010;
                myOut <= '0';
            else
                next_state <= got11;
                myOut <= '0';
            end if;
        when got11010 =>
            if (myInput = '0') then
                next_state <= gotNone;
                myOut <= '0';
            else
                next_state <= got110101;
                myOut <= '0';
            end if;
        when got110101 =>
            if (myInput = '0') then
                next_state <= gotNone;
                myOut <= '1';
            else
```



```
                    next_state <= got11;
                    myOut <= '1';
                end if;
        end case;
    end process combinational;


-- sequential part
    state_switch: process (clk, rst) begin
        if (rst = '1') then
            present_state <= gotNone;
        elsif (clk'event and clk = '1') then
            present_state <= next_state;
        end if;
    end process state_switch;

end Controller;
```

Here is the testbench that simply checks the correctness of the circuit:

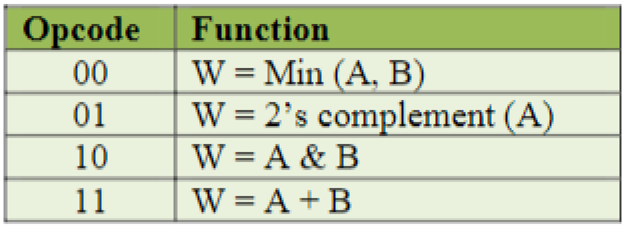


```
/sequence_detector_tb/myInput            1
/sequence_detector_tb/clk                0
/sequence_detector_tb/rst                0
/sequence_detector_tb/myOutput           0
/sequence_detector_tb/UUT/present_state  got11
/sequence_detector_tb/UUT/next_state     got11
```

The states change with every clock cycle and when the present state is “got110101”, the output goes high, showing that it's a Moore model.

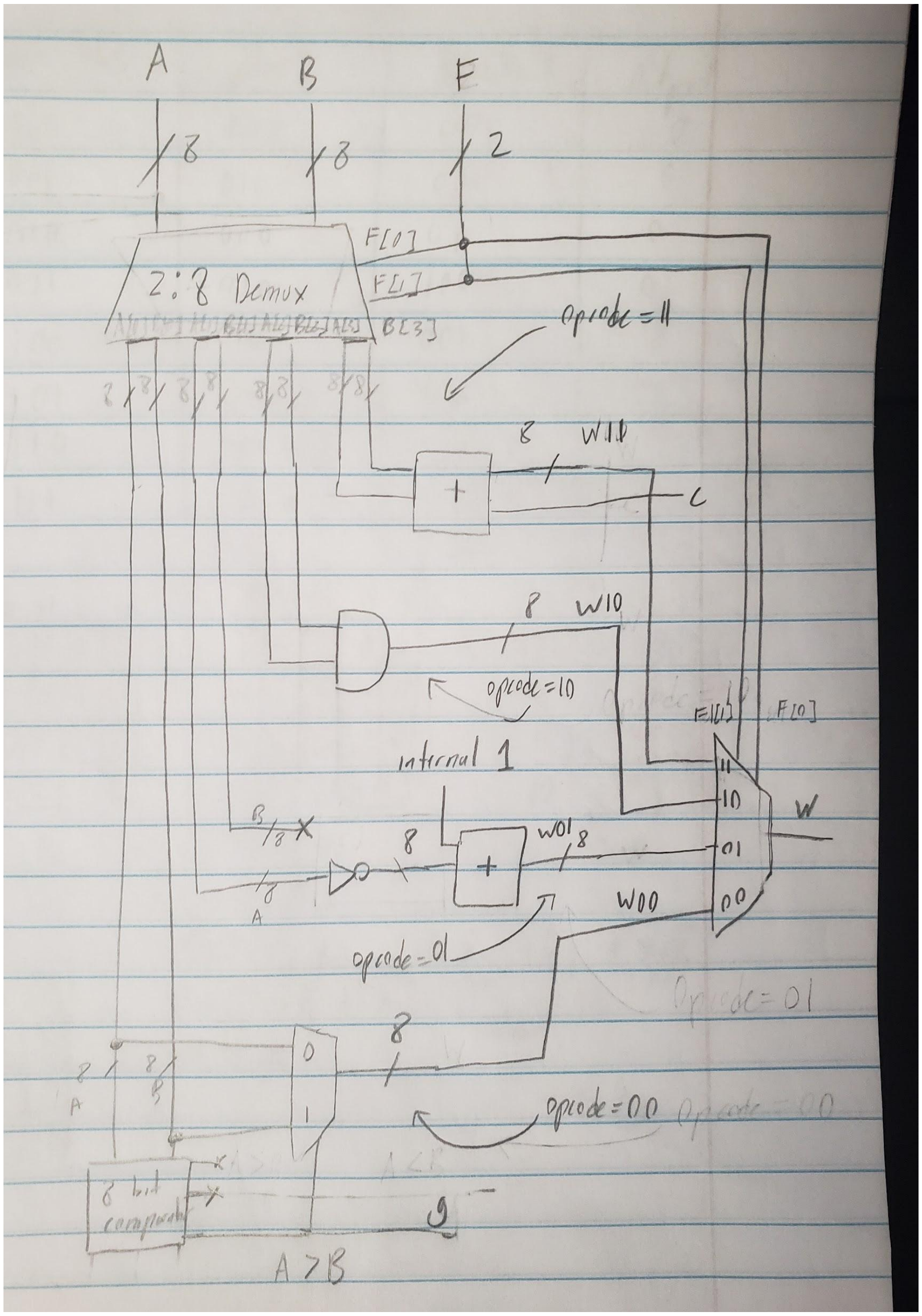
Problem 2:

In this problem, we create an 8-bit, four function ALU. The functions (controlled by F[1:0]) is shown below:



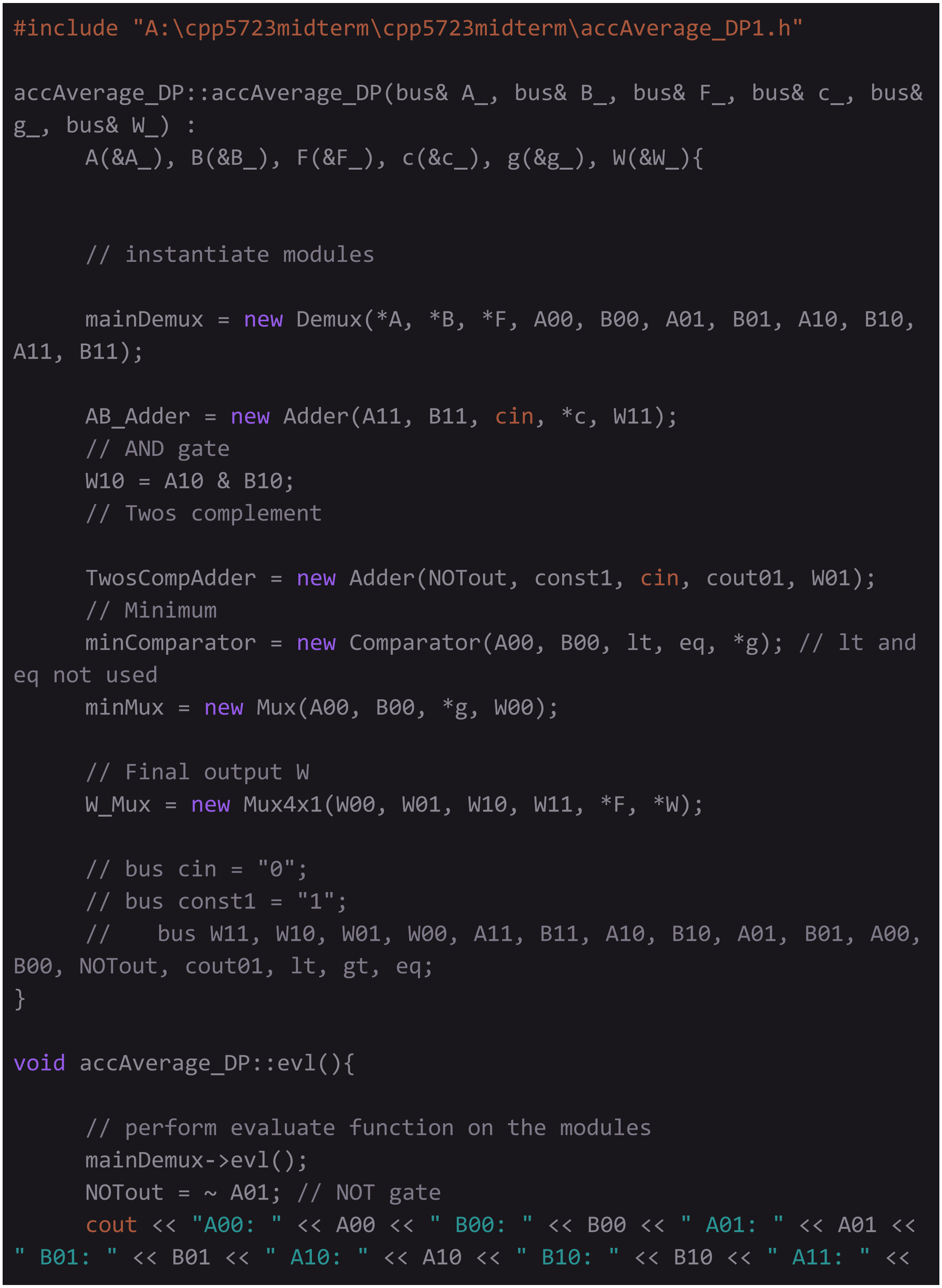
Opcode	Function
00	W = Min (A, B)
01	W = 2's complement (A)
10	W = A & B
11	W = A + B

The following is the block diagram of the circuit:

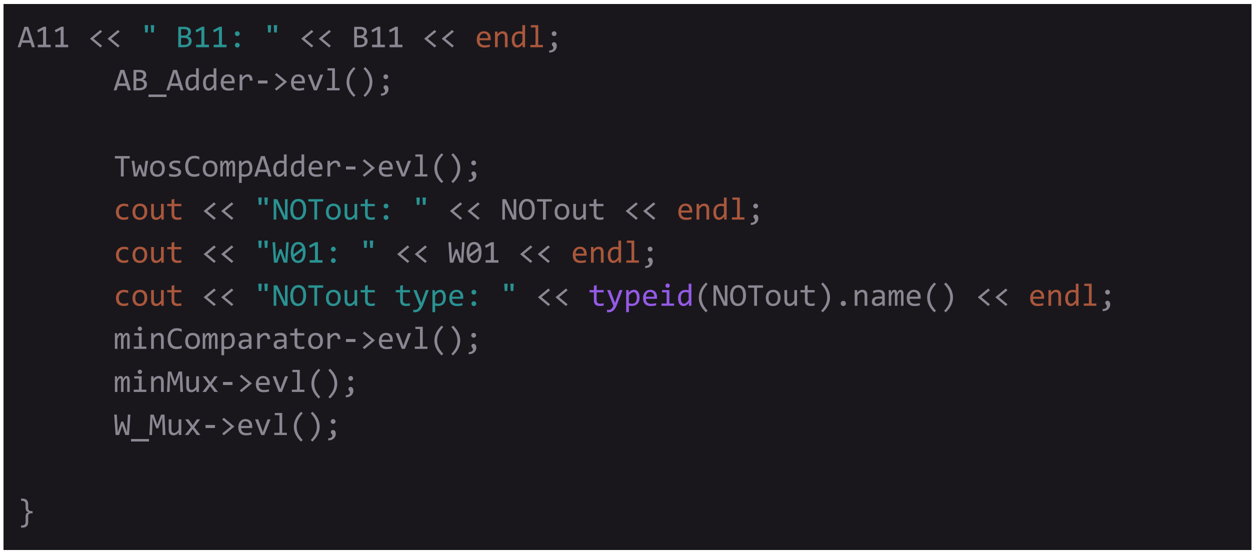


To create this in C++, I had to modify the classVectorPrimitives library a bit to add the comparator, the 2-bit multiplexer, and the demultiplexer. The following is the code to create this circuit:

```
#include "A:\cpp5723midterm\cpp5723midterm\classVectorPrimitives.h"

class accAverage_DP {
	bus *A;
	bus* B;
	bus *F;
	bus *W;
	bus* c;
	bus* g;

	Adder* AB_Adder;
	Adder* TwosCompAdder;
	Demux* mainDemux;
	Mux* minMux;
	Mux4x1* W_Mux;
	Comparator* minComparator;

	bus cin = "0";
	bus const1 = "00000001";
	bus W11, W10, W01, W00, A11, B11, A10, B10, A01, B01, A00, B00,
NOTout, cout01, lt, gt, eq;

public:
	accAverage_DP(bus& A, bus& B, bus& F, bus& c, bus& g, bus& W);
	~accAverage_DP();
	void evl();
};
```



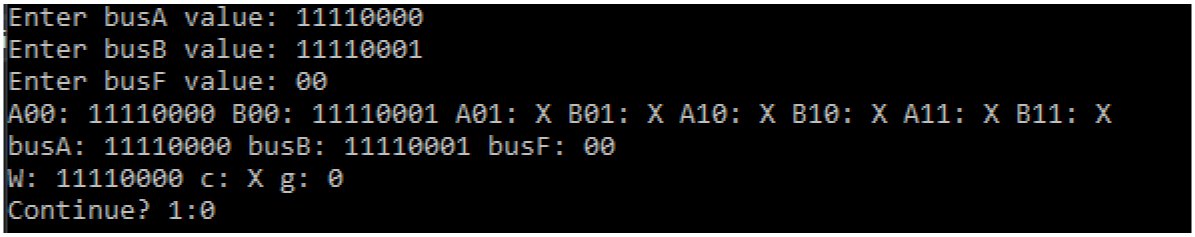
```
#include "A:\cpp5723midterm\cpp5723midterm\accAverage_DP1.h"

accAverage_DP::accAverage_DP(bus& A_, bus& B_, bus& F_, bus& c_, bus&
g_, bus& W_) :
      A(&A_), B(&B_), F(&F_), c(&c_), g(&g_), W(&W_){


      // instantiate modules

      mainDemux = new Demux(*A, *B, *F, A00, B00, A01, B01, A10, B10,
A11, B11);

      AB_Adder = new Adder(A11, B11, cin, *c, W11);
      // AND gate
      W10 = A10 & B10;
      // Twos complement

      TwosCompAdder = new Adder(NOTout, const1, cin, cout01, W01);
      // Minimum
      minComparator = new Comparator(A00, B00, lt, eq, *g); // lt and
eq not used
      minMux = new Mux(A00, B00, *g, W00);

      // Final output W
      W_Mux = new Mux4x1(W00, W01, W10, W11, *F, *W);

      // bus cin = "0";
      // bus const1 = "1";
      //   bus W11, W10, W01, W00, A11, B11, A10, B10, A01, B01, A00,
B00, NOTout, cout01, lt, gt, eq;
}

void accAverage_DP::evl(){

      // perform evaluate function on the modules
      mainDemux->evl();
      NOTout = ~ A01; // NOT gate
      cout << "A00: " << A00 << " B00: " << B00 << " A01: " << A01 <<
" B01: " << B01 << " A10: " << A10 << " B10: " << B10 << " A11: " <<
```



```
A11 << " B11: " << B11 << endl;
      AB_Adder->evl();

      TwosCompAdder->evl();
      cout << "NOTout: " << NOTout << endl;
      cout << "W01: " << W01 << endl;
      cout << "NOTout type: " << typeid(NOTout).name() << endl;
      minComparator->evl();
      minMux->evl();
      W_Mux->evl();

}
```

Testbench:

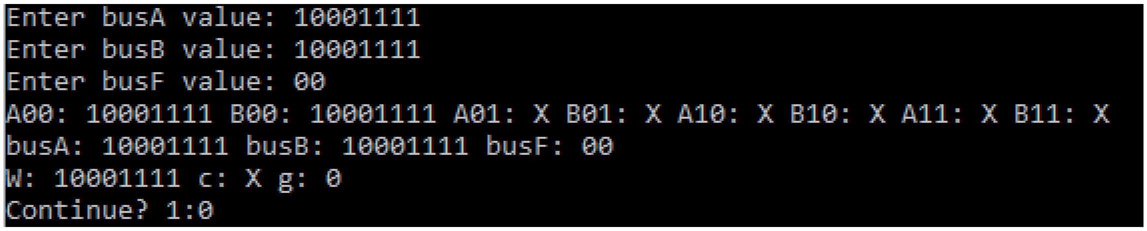
Scenario 1: Comparing 11110000 and 11110001 (Opcode 00)



```
Enter busA value: 11110000
Enter busB value: 11110001
Enter busF value: 00
A00: 11110000 B00: 11110001 A01: X B01: X A10: X B10: X A11: X B11: X
busA: 11110000 busB: 11110001 busF: 00
W: 11110000 c: X g: 0
Continue? 1:0
```

11110000 is smaller and therefore is W. g is 0 as busA is not greater than busB.

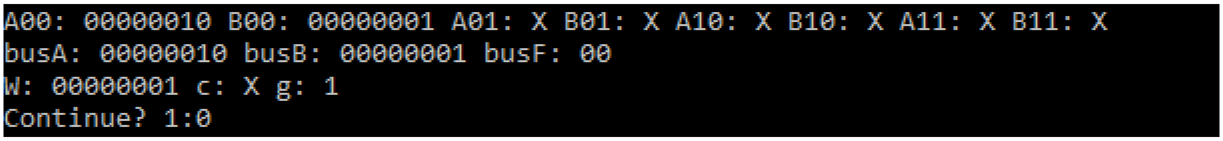
Scenario 2: Comparing 10001111 and 10001111 (Opcode 00)



```
Enter busA value: 10001111
Enter busB value: 10001111
Enter busF value: 00
A00: 10001111 B00: 10001111 A01: X B01: X A10: X B10: X A11: X B11: X
busA: 10001111 busB: 10001111 busF: 00
W: 10001111 c: X g: 0
Continue? 1:0
```

Both outputs are considered the smallest so it is passed through W. Since busA still is not greater than busB, g is 0.

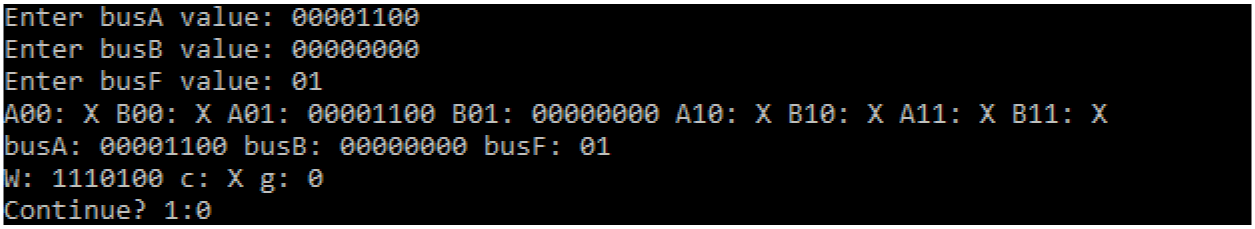
Scenario 3: busA > busB (Opcode 00)



```
A00: 00000010 B00: 00000001 A01: X B01: X A10: X B10: X A11: X B11: X
busA: 00000010 busB: 00000001 busF: 00
W: 00000001 c: X g: 1
Continue? 1:0
```

busA = 2 while busB = 1. This causes busB to be outputted to W. Since busA was greater than busB, g = 1.

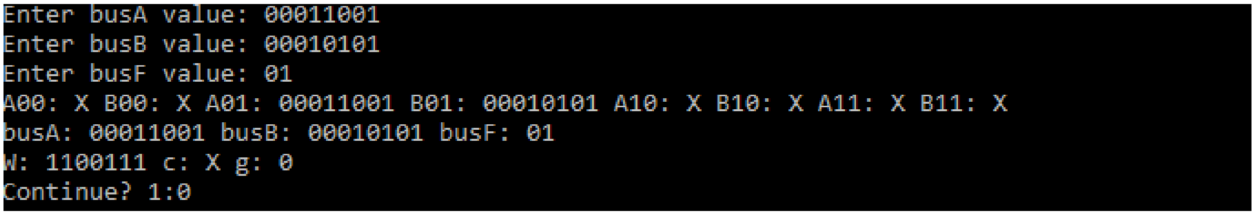
Scenario 4: Two's Complement of 12 (Opcode 01)



```
Enter busA value: 00001100
Enter busB value: 00000000
Enter busF value: 01
A00: X B00: X A01: 00001100 B01: 00000000 A10: X B10: X A11: X B11: X
busA: 00001100 busB: 00000000 busF: 01
W: 1110100 c: X g: 0
Continue? 1:0
```

The two's complement of 12 is 1110100 which the circuit correctly displays.

Scenario 5: Two's Complement of 25 (Opcode 01)

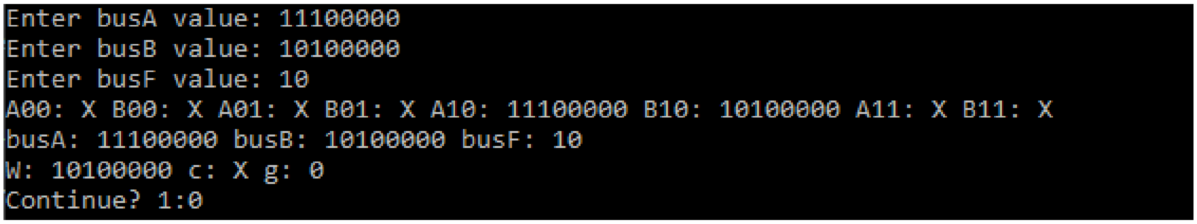


```
Enter busA value: 00011001
Enter busB value: 00010101
Enter busF value: 01
A00: X B00: X A01: 00011001 B01: 00010101 A10: X B10: X A11: X B11: X
busA: 00011001 busB: 00010101 busF: 01
W: 1100111 c: X g: 0
Continue? 1:0
```

The two's complement of 25 is 1100111 which the circuit correctly displays. Also inputs into busB does not matter.

Scenario 6: 11100000 AND 10100000 (Opcode 10)

The predicted result should be 10100000.

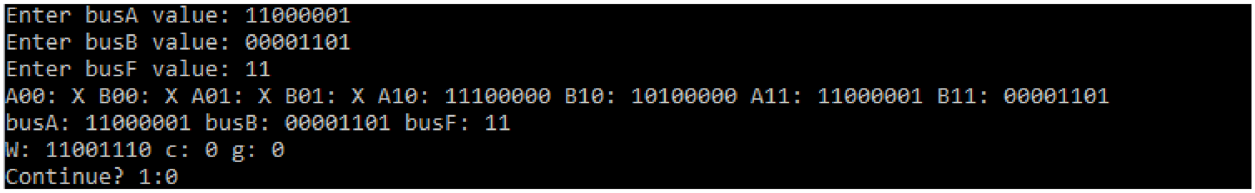


```
Enter busA value: 11100000
Enter busB value: 10100000
Enter busF value: 10
A00: X B00: X A01: X B01: X A10: 11100000 B10: 10100000 A11: X B11: X
busA: 11100000 busB: 10100000 busF: 10
W: 10100000 c: X g: 0
Continue? 1:0
```

Our predicted result is correct.

Scenario 7: Adding 11000001 and 00001101 (193 + 13) (Opcode 11)

Our predicted result is 11001110 or 206.

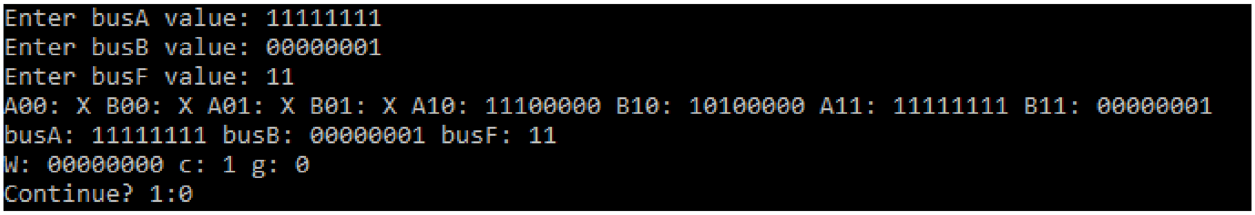


```
Enter busA value: 11000001
Enter busB value: 00001101
Enter busF value: 11
A00: X B00: X A01: X B01: X A10: 11100000 B10: 10100000 A11: 11000001 B11: 00001101
busA: 11000001 busB: 00001101 busF: 11
W: 11001110 c: 0 g: 0
Continue? 1:0
```

Our predicted result is correct and there is no carry out for this one.

Scenario 8: Adding 11111111 and 000000001 (255 + 1) (Opcode 11)

Our predicted result is 00000000 with a carry out of 1 due to overflow.



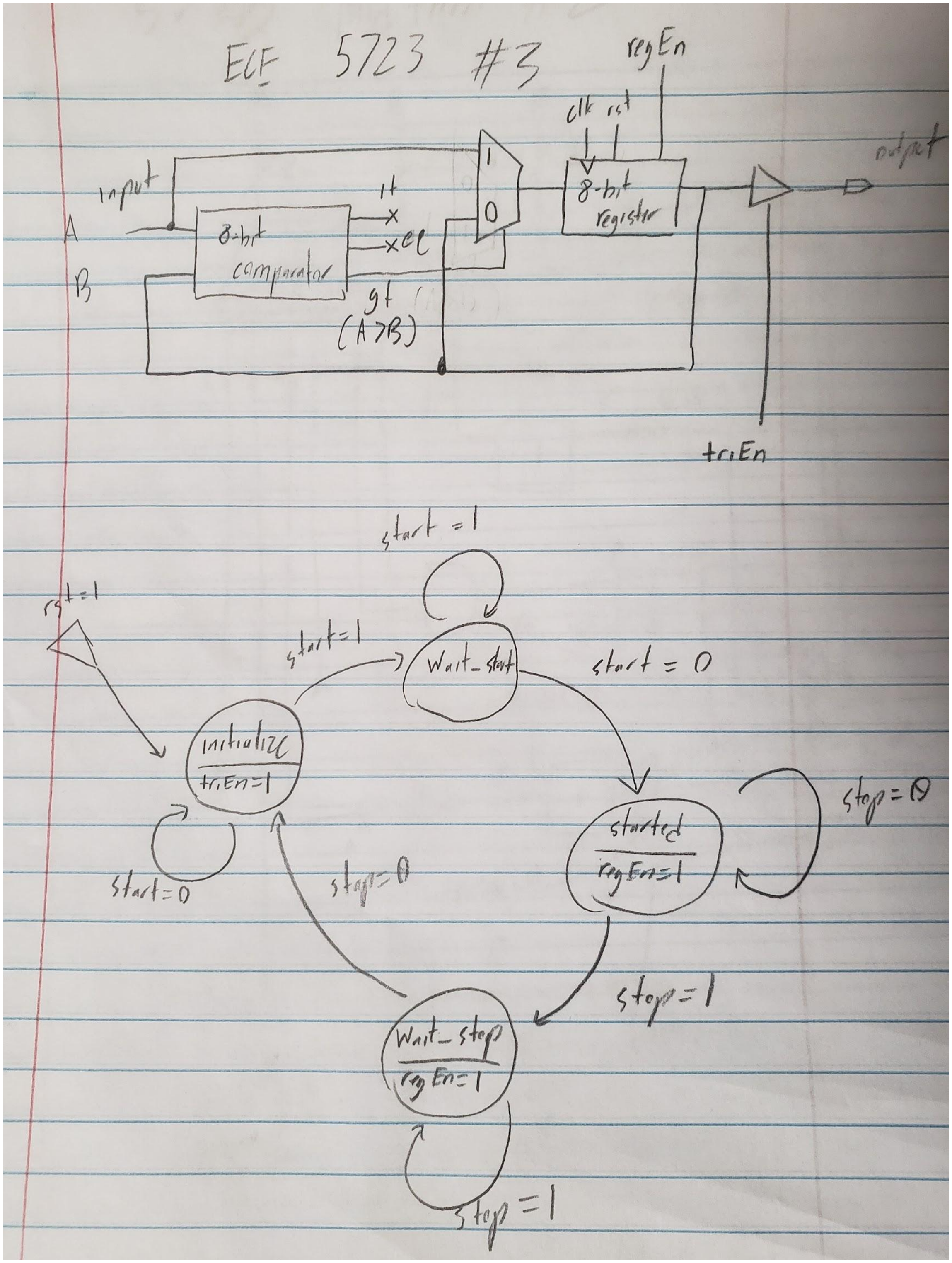
```
Enter busA value: 11111111
Enter busB value: 00000001
Enter busF value: 11
A00: X B00: X A01: X B01: X A10: 11100000 B10: 10100000 A11: 11111111 B11: 00000001
busA: 11111111 busB: 00000001 busF: 11
W: 00000000 c: 1 g: 0
Continue? 1:0
```

Our predictions on both W and c are correct.

Problem 3:

In this problem, we have to build a circuit that finds the biggest value on its input bus. The process starts with a start pulse and the results are shown with the stop pulse.

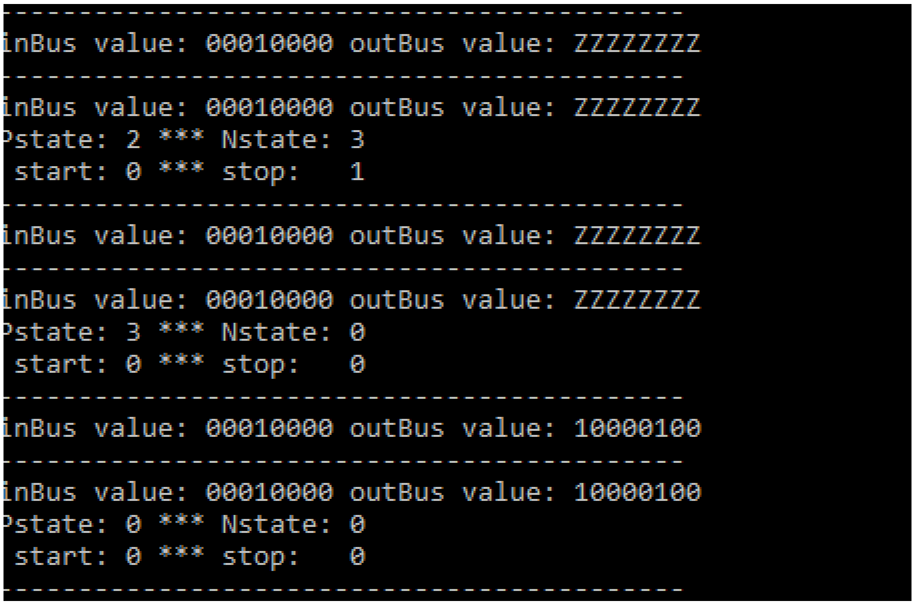
The following is the datapath and the controller for the circuit:



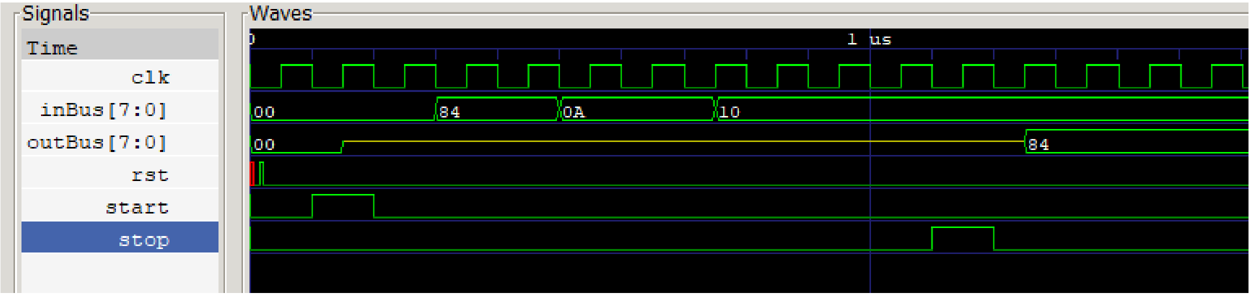
Code will be provided in the .zip file.

Testbench:

Scenario 1: Testing the largest of 00000000, 10000100, 00001010, 00010000 in that order

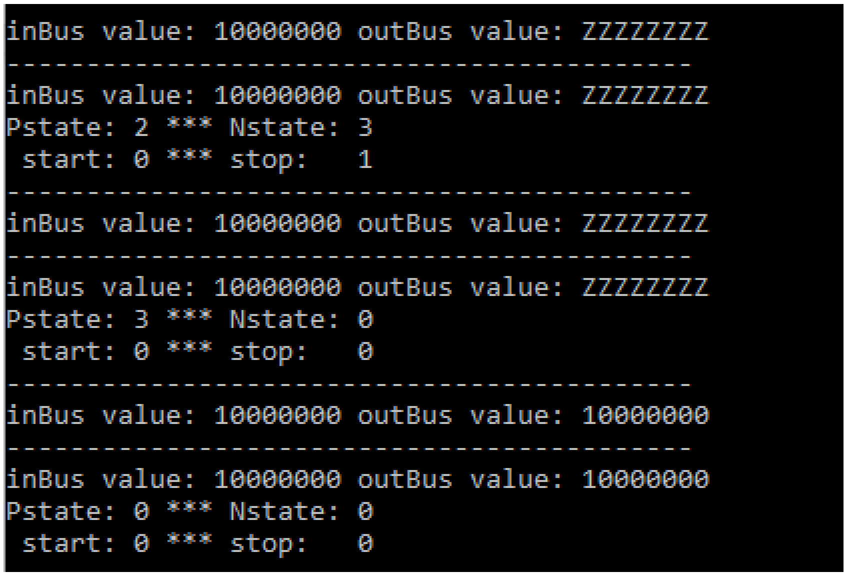


```
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   1
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
------------------------------------------
inBus value: 00010000 outBus value: 10000100
------------------------------------------
inBus value: 00010000 outBus value: 10000100
Pstate: 0 *** Nstate: 0
 start: 0 *** stop:   0
------------------------------------------
```

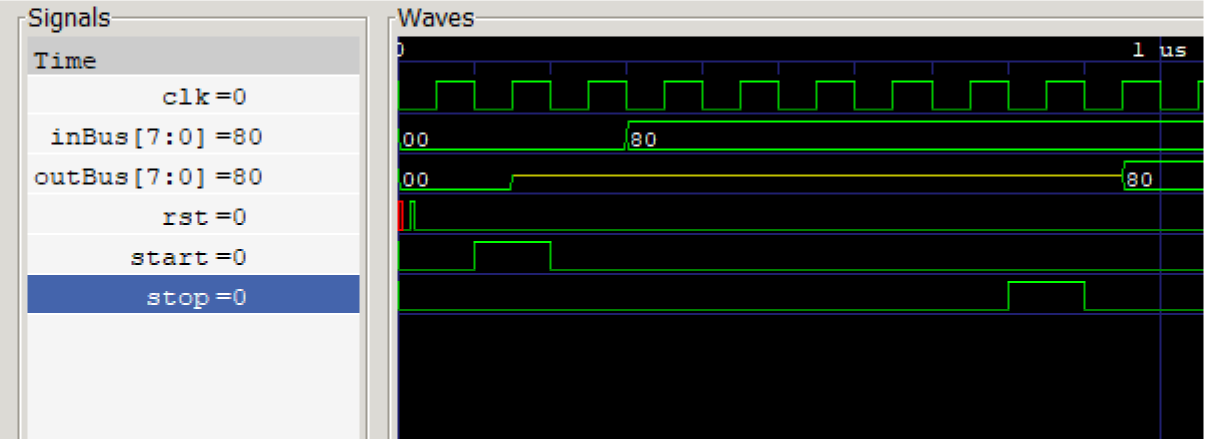


Once the stop signal is given, it outputs the largest held value which is 10000100.

Scenario 2: Testing largest of 10000000 and 10000000 (equal values)

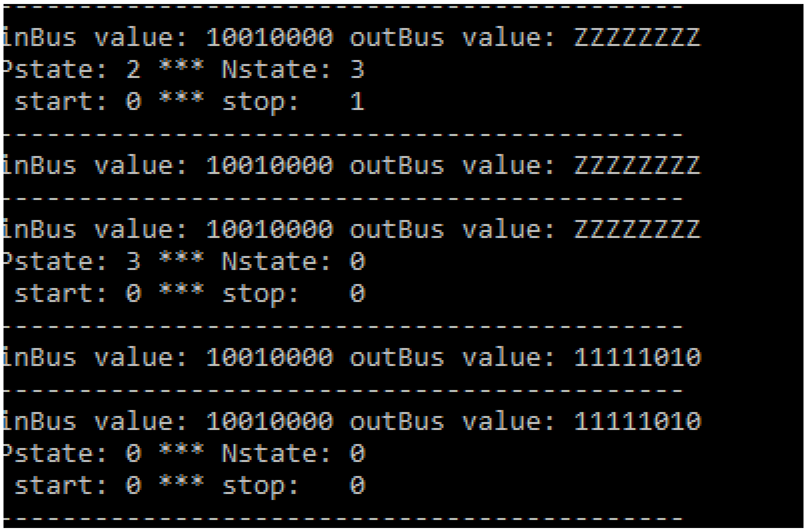


```
inBus value: 10000000 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   1
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
-------------------------------------------
inBus value: 10000000 outBus value: 10000000
-------------------------------------------
inBus value: 10000000 outBus value: 10000000
Pstate: 0 *** Nstate: 0
 start: 0 *** stop:   0
```

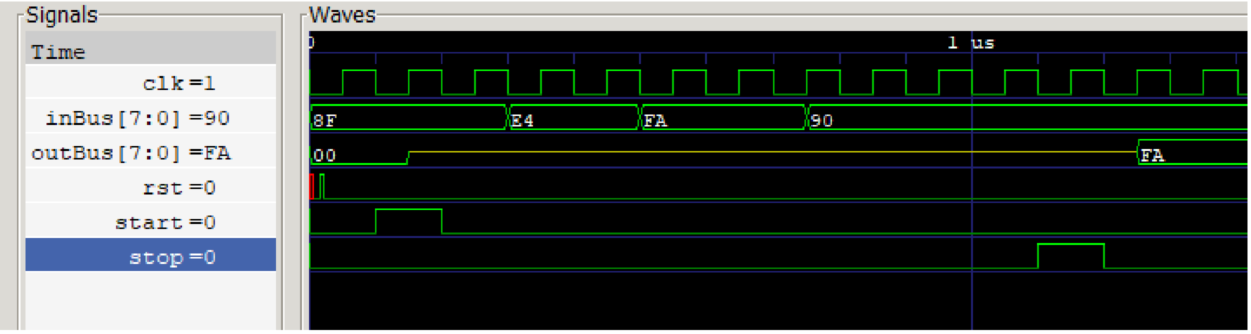


If there is only ever 1 value on the input bus, it is automatically made the max.

Scenario 3: Testing the largest of 10001111, 11100100, 11111010, 10010000 in that order

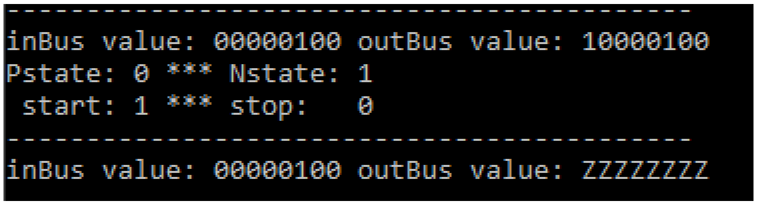


```
------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   1
------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
------------------------------------------
inBus value: 10010000 outBus value: 11111010
------------------------------------------
inBus value: 10010000 outBus value: 11111010
Pstate: 0 *** Nstate: 0
 start: 0 *** stop:   0
------------------------------------------
```



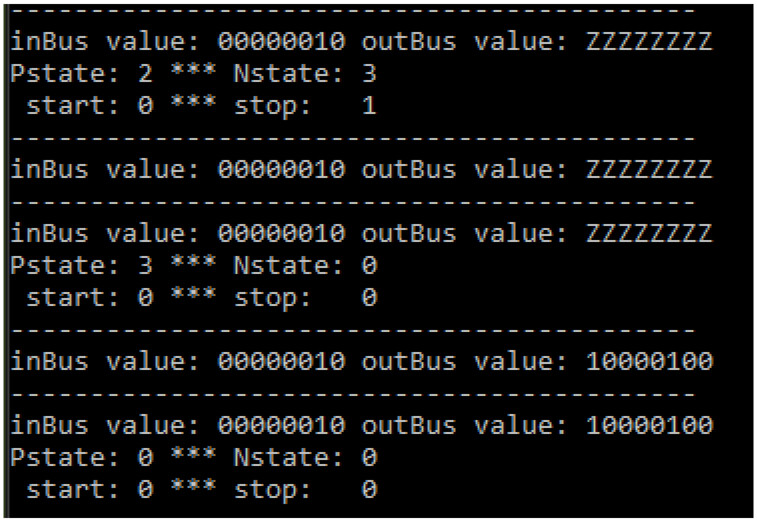
This scenario just tests the circuit function again with different values and the largest value (11111010) is correctly displayed.

Scenario 4: Scenario 1 starting after it was stopped but now with smaller values



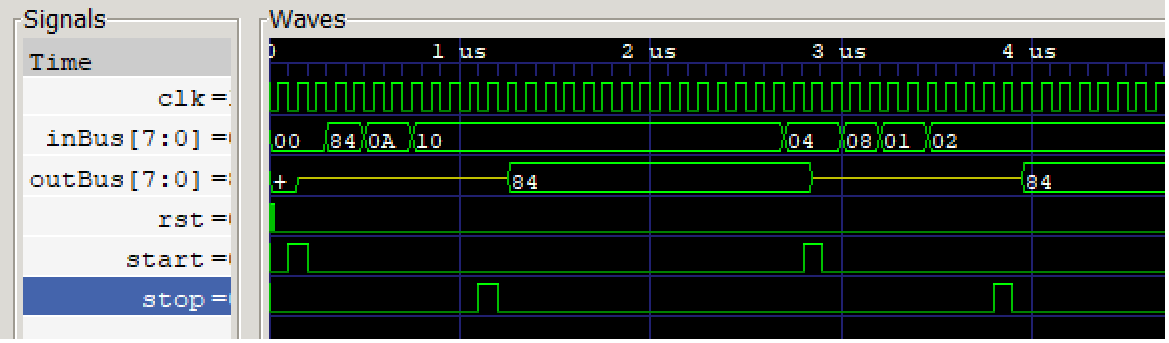
```
------------------------------------------
inBus value: 00000100 outBus value: 10000100
Pstate: 0 *** Nstate: 1
 start: 1 *** stop:   0
------------------------------------------
inBus value: 00000100 outBus value: ZZZZZZZZ
```

Output becomes high impedance when “start” is detected.



```
-------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   1
-------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
-------------------------------------------
inBus value: 00000010 outBus value: 10000100
-------------------------------------------
inBus value: 00000010 outBus value: 10000100
Pstate: 0 *** Nstate: 0
 start: 0 *** stop:   0
```

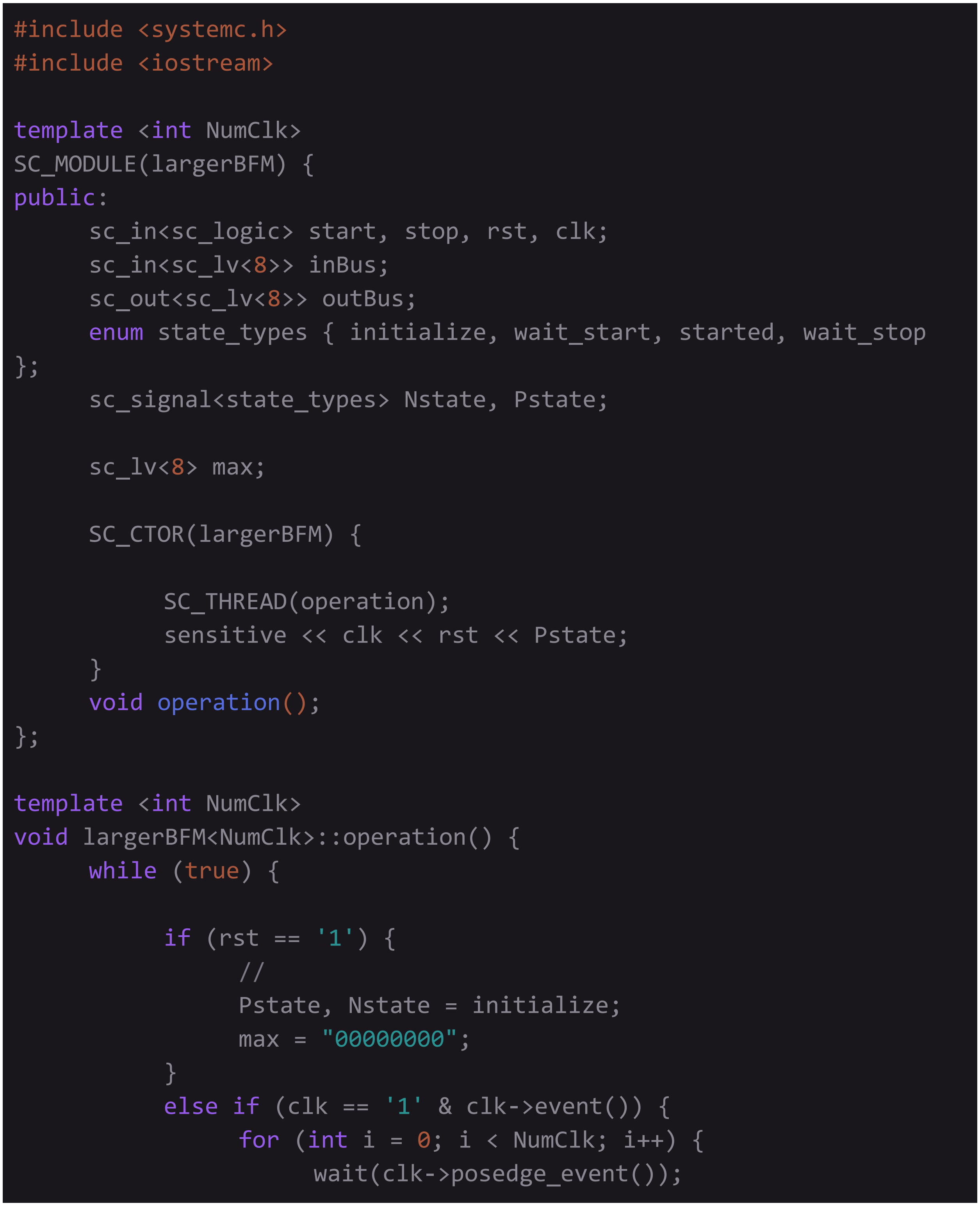
Without a reset pulse, the largest remains the same from the first start.



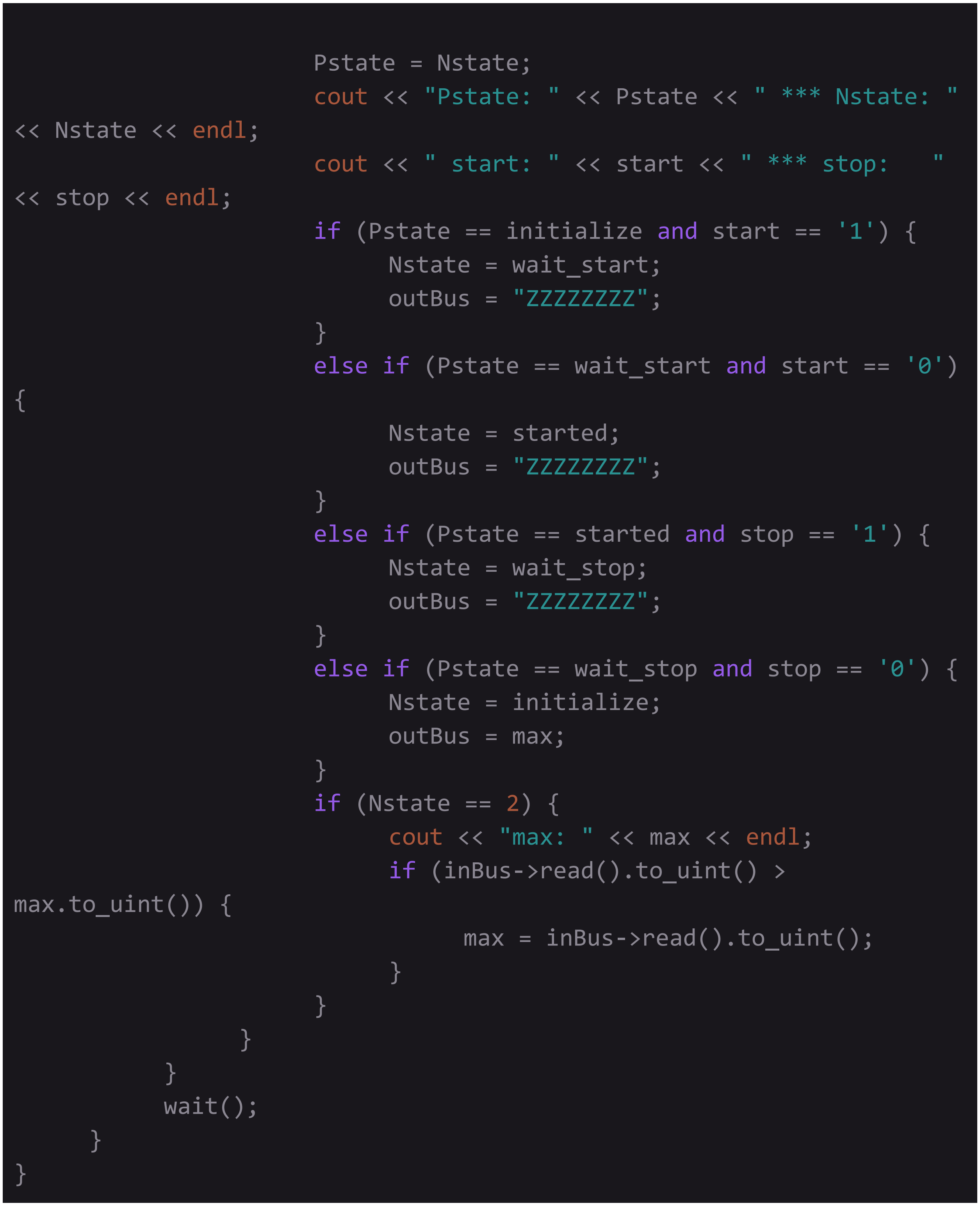
As seen from this waveform, the previous largest value of 10000100 is retained.

BFM Model:

The following is the code used to make the BFM model:

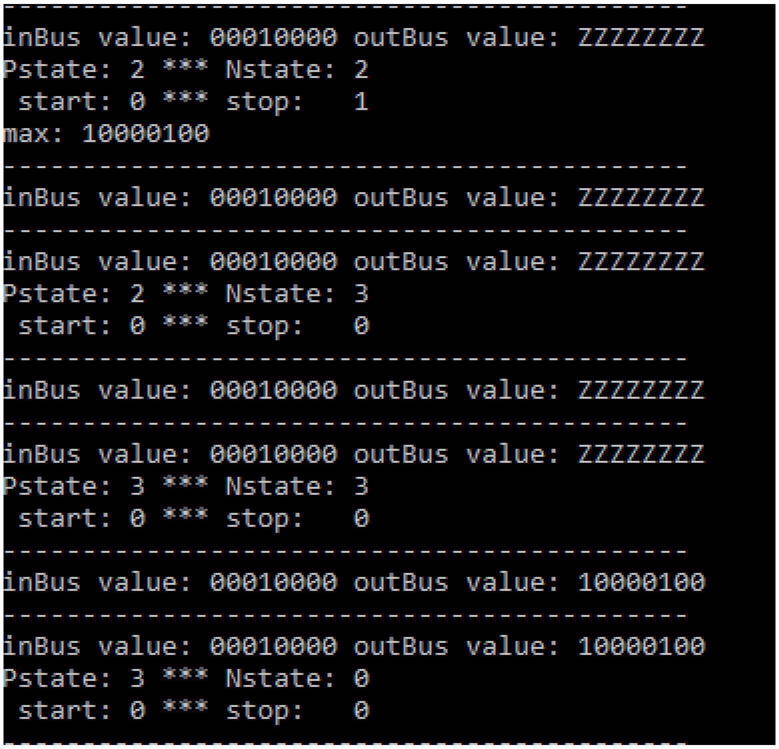


```
#include <systemc.h>
#include <iostream>

template <int NumClk>
SC_MODULE(largerBFM) {
public:
    sc_in<sc_logic> start, stop, rst, clk;
    sc_in<sc_lv<8>> inBus;
    sc_out<sc_lv<8>> outBus;
    enum state_types { initialize, wait_start, started, wait_stop
};
    sc_signal<state_types> Nstate, Pstate;

    sc_lv<8> max;

    SC_CTOR(largerBFM) {

        SC_THREAD(operation);
        sensitive << clk << rst << Pstate;
    }
    void operation();
};

template <int NumClk>
void largerBFM<NumClk>::operation() {
    while (true) {

        if (rst == '1') {
            //
            Pstate, Nstate = initialize;
            max = "00000000";
        }
        else if (clk == '1' & clk->event()) {
            for (int i = 0; i < NumClk; i++) {
                wait(clk->posedge_event());
```

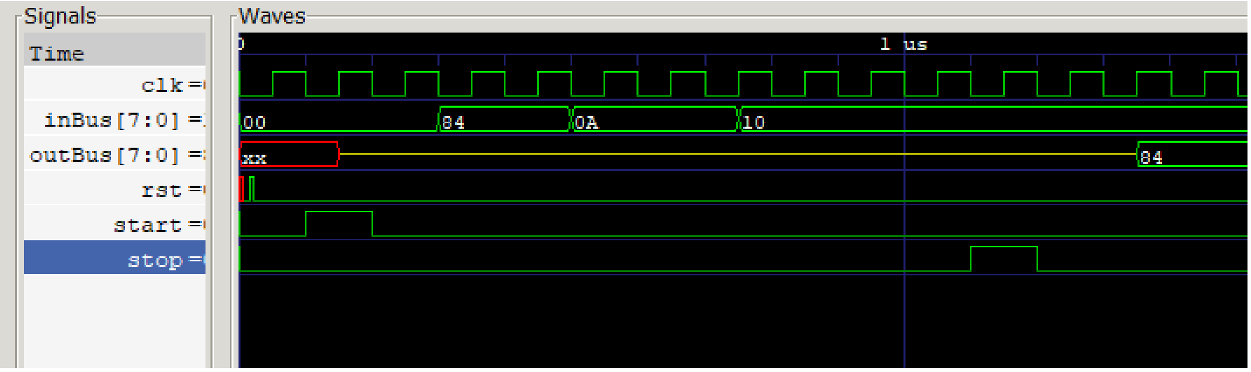


```
                Pstate = Nstate;
                cout << "Pstate: " << Pstate << " *** Nstate: "
<< Nstate << endl;
                cout << " start: " << start << " *** stop:   "
<< stop << endl;
                if (Pstate == initialize and start == '1') {
                    Nstate = wait_start;
                    outBus = "ZZZZZZZZ";
                }
                else if (Pstate == wait_start and start == '0')
{
                    Nstate = started;
                    outBus = "ZZZZZZZZ";
                }
                else if (Pstate == started and stop == '1') {
                    Nstate = wait_stop;
                    outBus = "ZZZZZZZZ";
                }
                else if (Pstate == wait_stop and stop == '0') {
                    Nstate = initialize;
                    outBus = max;
                }
                if (Nstate == 2) {
                    cout << "max: " << max << endl;
                    if (inBus->read().to_uint() >
max.to_uint()) {
                        max = inBus->read().to_uint();
                    }
                }
            }
        }
        wait();
    }
}
```

BFM Scenario 1: Testing the largest of 00000000, 10000100, 00001010, 00010000 in that order

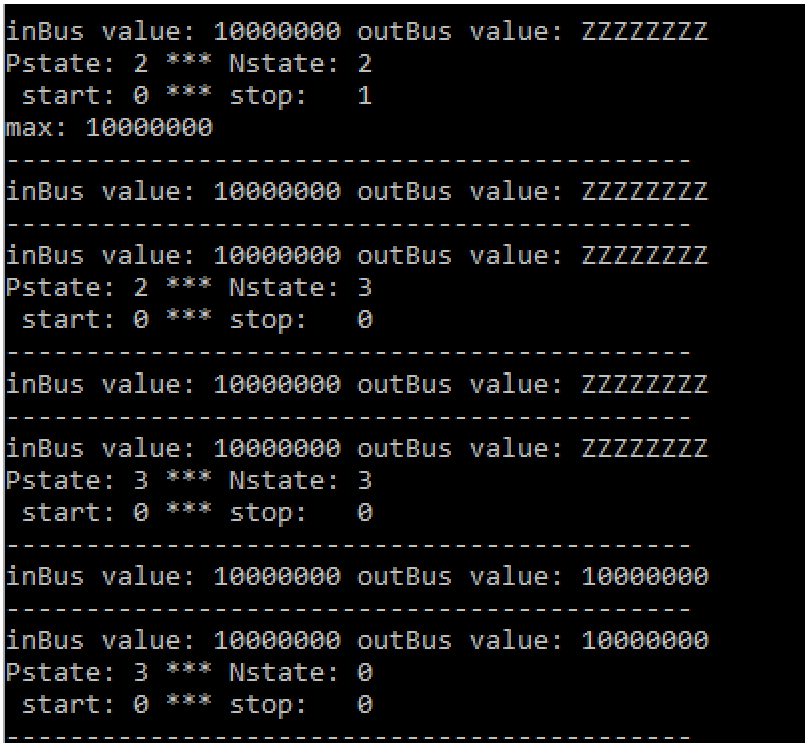


```
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 2
 start: 0 *** stop:   1
max: 10000100
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   0
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00010000 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 3
 start: 0 *** stop:   0
------------------------------------------
inBus value: 00010000 outBus value: 10000100
------------------------------------------
inBus value: 00010000 outBus value: 10000100
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
------------------------------------------
```

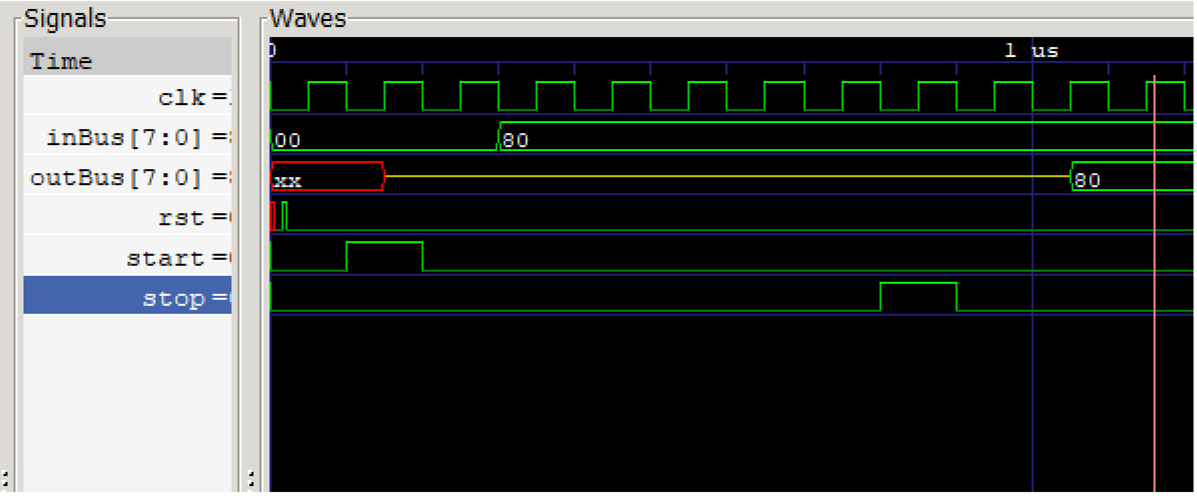


The circuit reads the inputs and holds the largest value until a stop pulse is detected. It correctly identifies and outputs 10000100 as the largest value.

BFM Scenario 2: Testing largest of 10000000 and 10000000 (equal values)

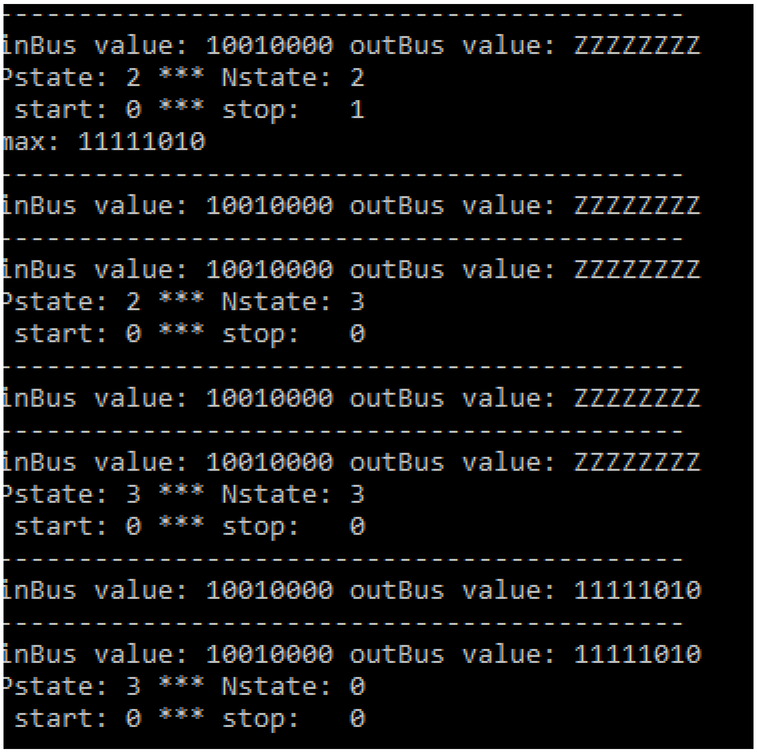


```
inBus value: 10000000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 2
 start: 0 *** stop:   1
max: 10000000
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   0
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 10000000 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 3
 start: 0 *** stop:   0
-------------------------------------------
inBus value: 10000000 outBus value: 10000000
-------------------------------------------
inBus value: 10000000 outBus value: 10000000
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
-------------------------------------------
```

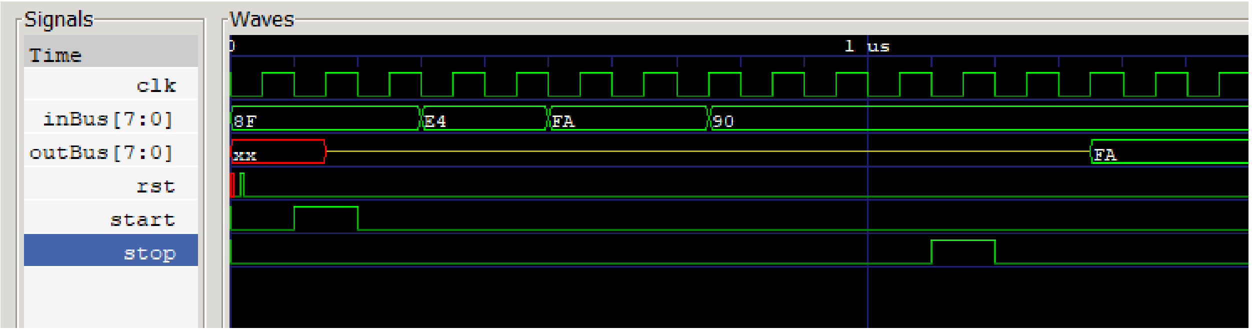


The BFM model will make 10000000 its largest held value if its the only value in the input bus.

BFM Scenario 3: Testing the largest of 10001111, 11100100, 11111010, 10010000 in that order

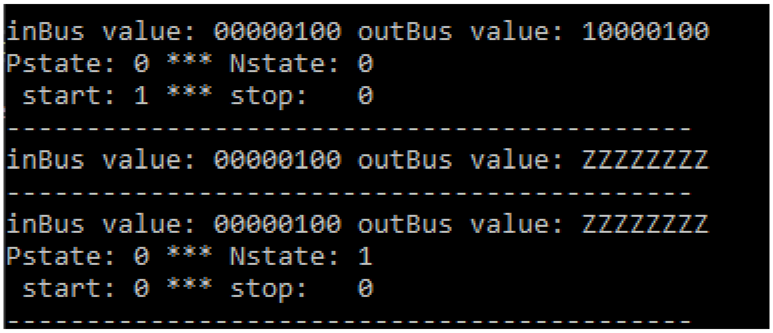


```
-------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 2
 start: 0 *** stop:   1
max: 11111010
-------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   0
-------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
-------------------------------------------
inBus value: 10010000 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 3
 start: 0 *** stop:   0
-------------------------------------------
inBus value: 10010000 outBus value: 11111010
-------------------------------------------
inBus value: 10010000 outBus value: 11111010
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
```



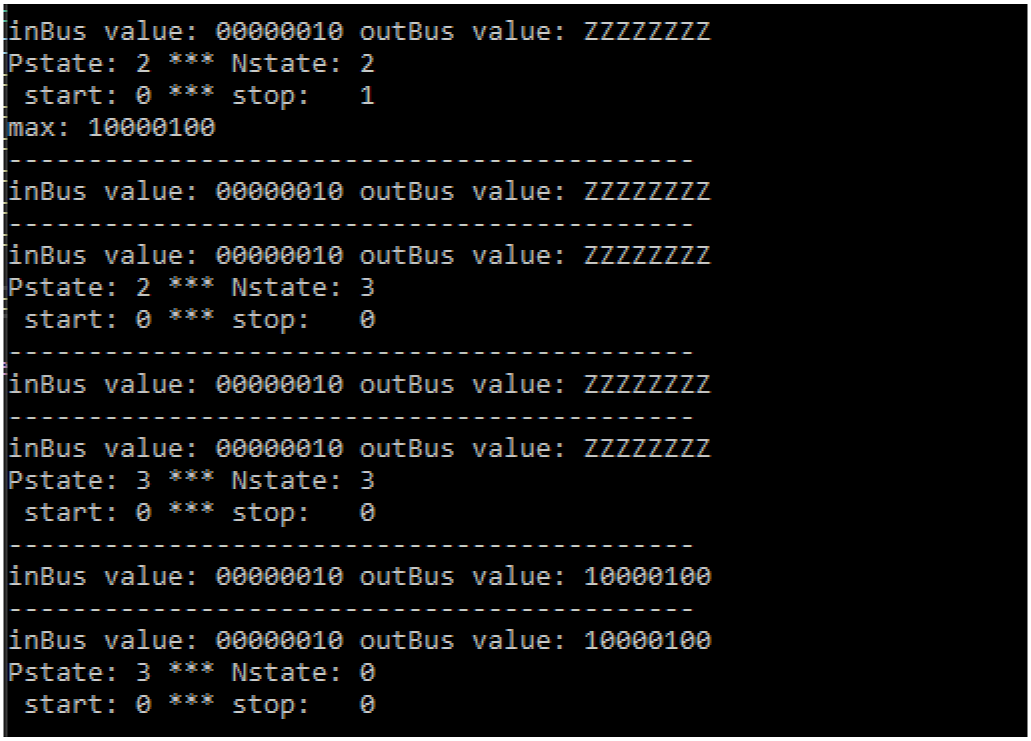
The BFM correctly outputs 11111010 as its highest value.

BFM Scenario 4: Scenario 1 starting after it was stopped but now with smaller values



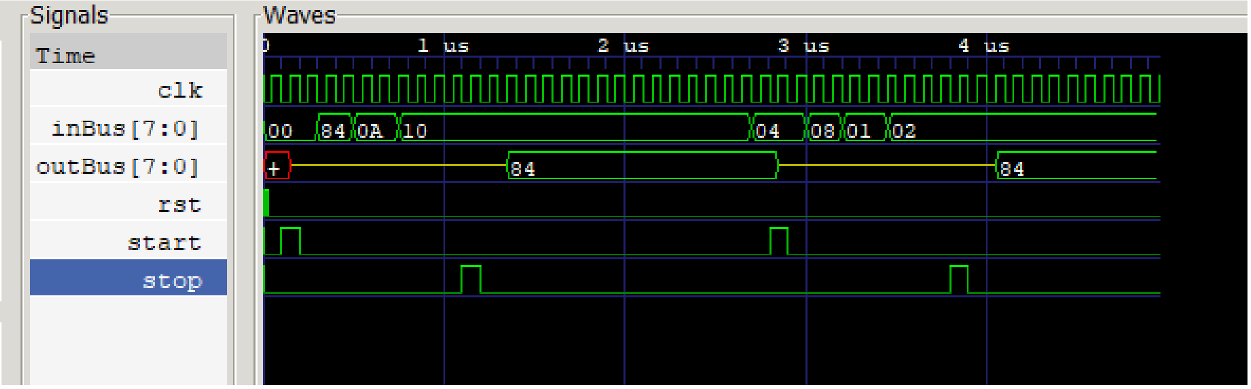
```
inBus value: 00000100 outBus value: 10000100
Pstate: 0 *** Nstate: 0
 start: 1 *** stop:   0
------------------------------------------
inBus value: 00000100 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00000100 outBus value: ZZZZZZZZ
Pstate: 0 *** Nstate: 1
 start: 0 *** stop:   0
------------------------------------------
```

The BFM output also goes high when a start pulse is detected.



```
inBus value: 00000010 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 2
 start: 0 *** stop:   1
max: 10000100
------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
Pstate: 2 *** Nstate: 3
 start: 0 *** stop:   0
------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
------------------------------------------
inBus value: 00000010 outBus value: ZZZZZZZZ
Pstate: 3 *** Nstate: 3
 start: 0 *** stop:   0
------------------------------------------
inBus value: 00000010 outBus value: 10000100
------------------------------------------
inBus value: 00000010 outBus value: 10000100
Pstate: 3 *** Nstate: 0
 start: 0 *** stop:   0
```

Without a reset, the BFM will also still hold the previous largest values.


Signals
Waves
Time
clk
inBus[7:0]
outBus[7:0]
rst
start
stop
0
1 us
2 us
3 us
4 us
00
84
0A
10
04
08
01
02
+
84
84