Table of Contents

1. VHDL Design Modeling

1.1. Description of the Task

You are to implement a 1011 Moore sequence detector in VHDL. In addition to detecting the sequence, the circuit keeps track of modulo-256 count of the 1011 sequences ever detected. When the correct sequence is detected, the **w** output becomes 1 and at the same time an 8-bit counter is incremented. Include an asynchronous active-high reset input for all clocked components.

A. Show the state diagram for this circuit.

B. Write a behavioral VHDL description of this sequence detector using VHDL PROCESS statements (no gate-level implementation is needed). Use Huffman model.

C. Write a testbench for the circuit in VHDL that tests the correctness of your circuit.

1.2. Solution

1.2.1. Diagrams

In this chapter, the diagrams of the designed circuit of 1011-sequence detector are presented (Fig. 1, Fig. 2).

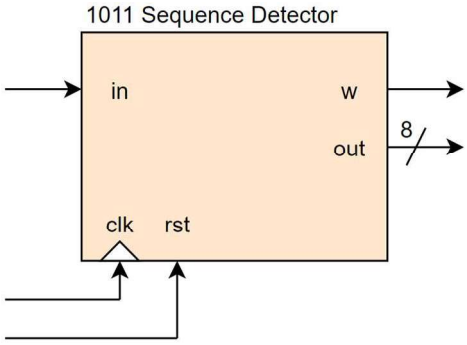

Fig. 1 – Sequence detector circuit

The detector is overlapping (Fig. 2).

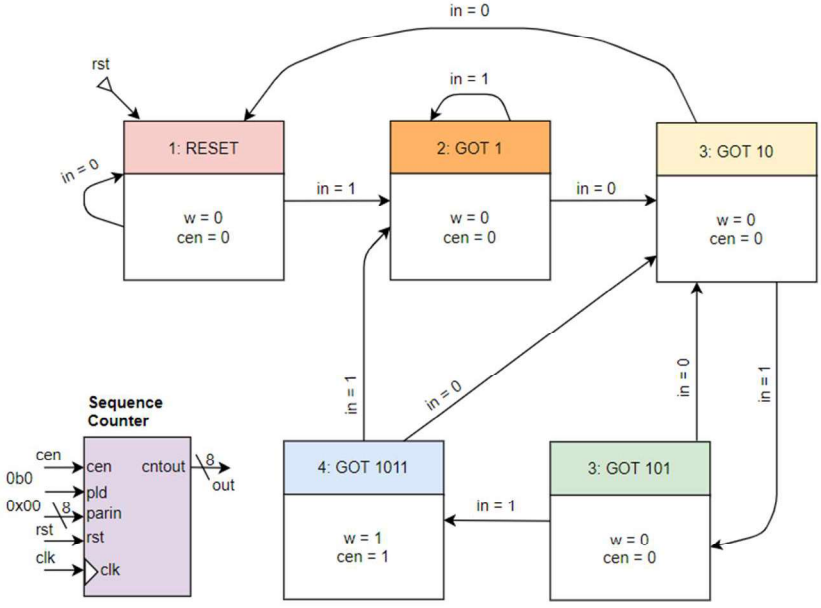

Fig. 2 – State diagram of the sequence detector

1.2.2. Test Results

In this chapter, the results of testing the designed circuit of 1011-sequence detector are presented (Fig. 3 - Fig. 6).

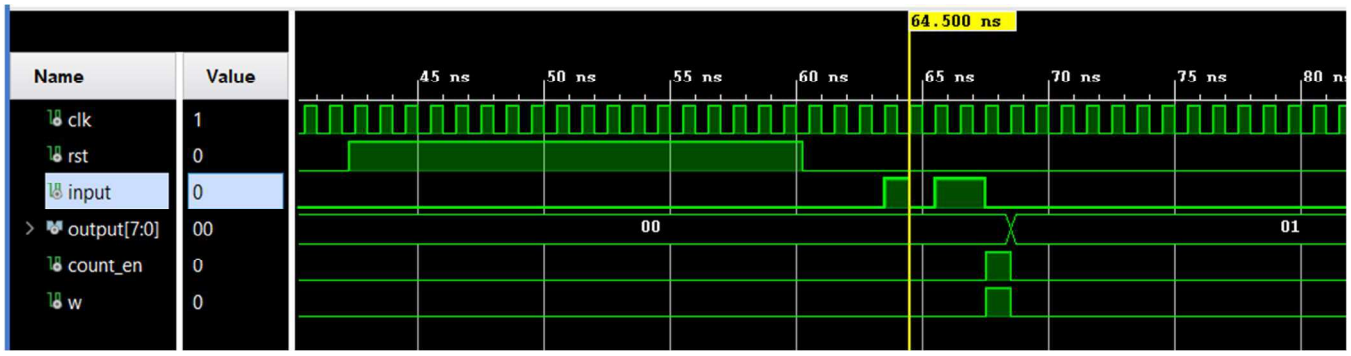

Fig. 3 – Test case 1: 1011 sequence was received, w becomes 1, and the counter is incremented (count_en = 1). The output of the counter changes on the next clock cycle after the sequence was detected.

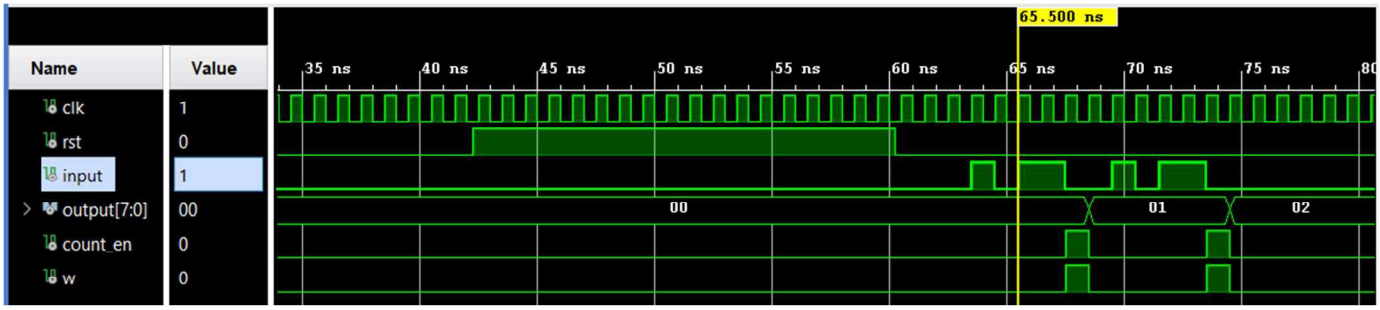

Fig. 4 – Test case 2: 2 non-overlapping 1011 sequences were received.

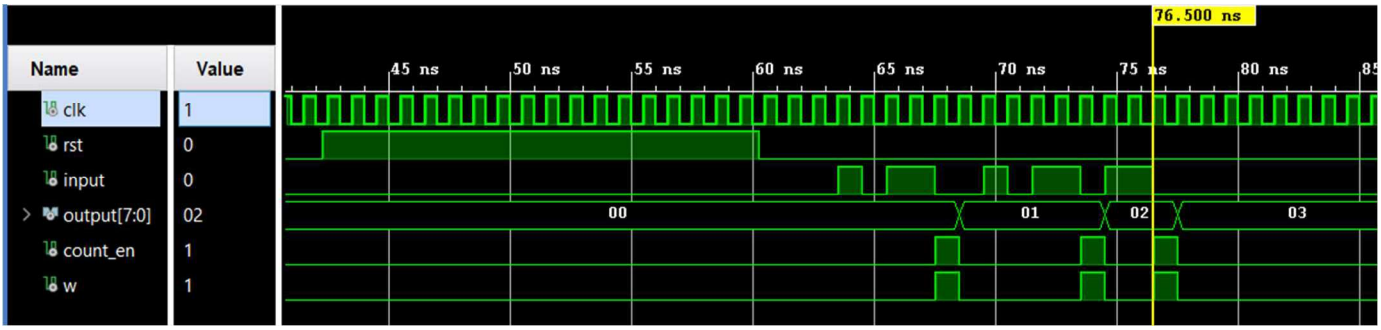

Fig. 5 – Test case 3: 2 non-overlapping and 1 overlapping 1011 sequences were received.

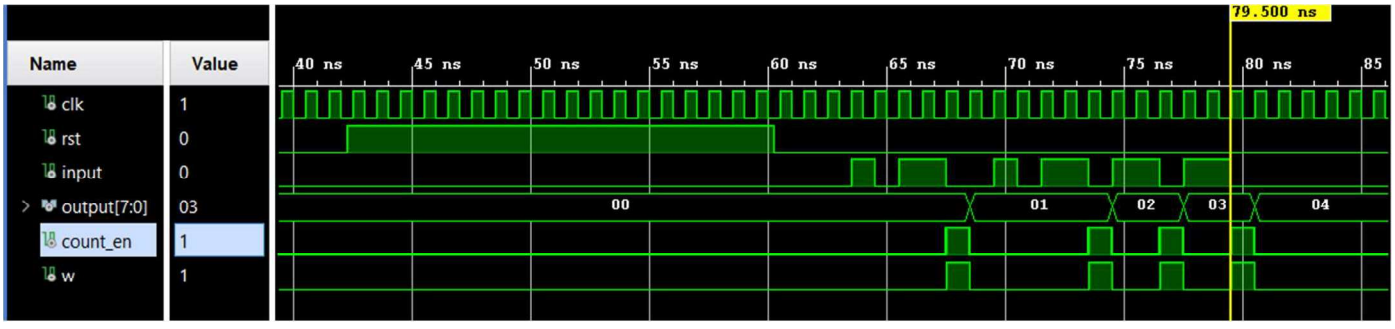

Fig. 6 – Test case 4: 2 non-overlapping and 2 overlapping 1011 sequences were received.

So, the tests show that the circuit operates correctly. The count_en output has a delay in 1 clock cycle, because cen becomes one in GOT1011 state, and the output of the counter changes its output on the next clock cycle.

2. C/C++ RTL Design Modeling

2.1. Description of the Task

An 8-bit, 4-function ALU is to be designed. The table below lists functions that the ALU can perform and their corresponding opcodes. The ALU inputs are data A[7:0] and B[7:0], and the 2-bit function F[1:0]. The ALU outputs are W[7:0], carry (c), and greater (g), where W is defined in the table below, c is the carry output for arithmetic operations, and g becomes 1 when A is greater than B.

Opcode	Function
00	W = A + B
01	W = Max (A, B)
10	W = 0
11	W = A | B

A. Show the block diagram of this circuit using adder, multiplexer, other RTL components discussed in lectures.
B. Write a C++ description that corresponds to the block diagram of Part A. Use the *bus* library discussed in lectures (Chapter 3).

2.2. Solution

2.2.1. Diagrams

In this chapter, the diagrams of the designed circuit of 4-function ALU are presented (Fig. 7, Fig. 8).

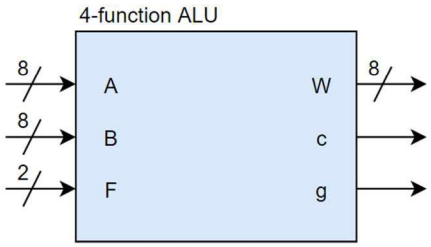

Fig. 7 – Four-function ALU circuit

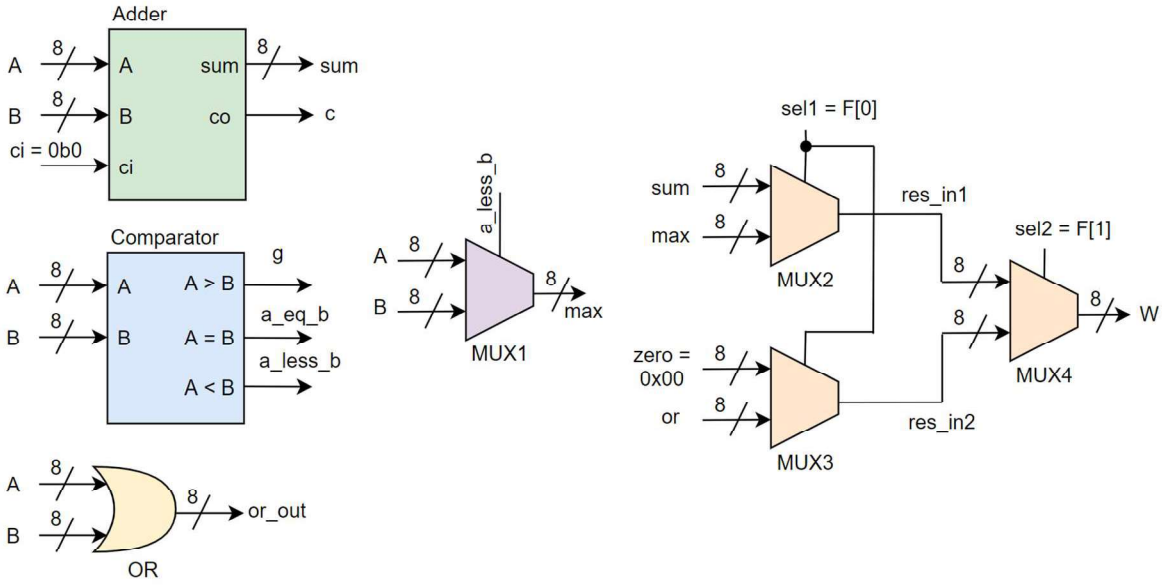

Fig. 8 – Block diagram of the 4-function ALU

2.2.2. Test Results

In this chapter, the results of testing the designed circuit of 4-function ALU are presented (Fig. 9).

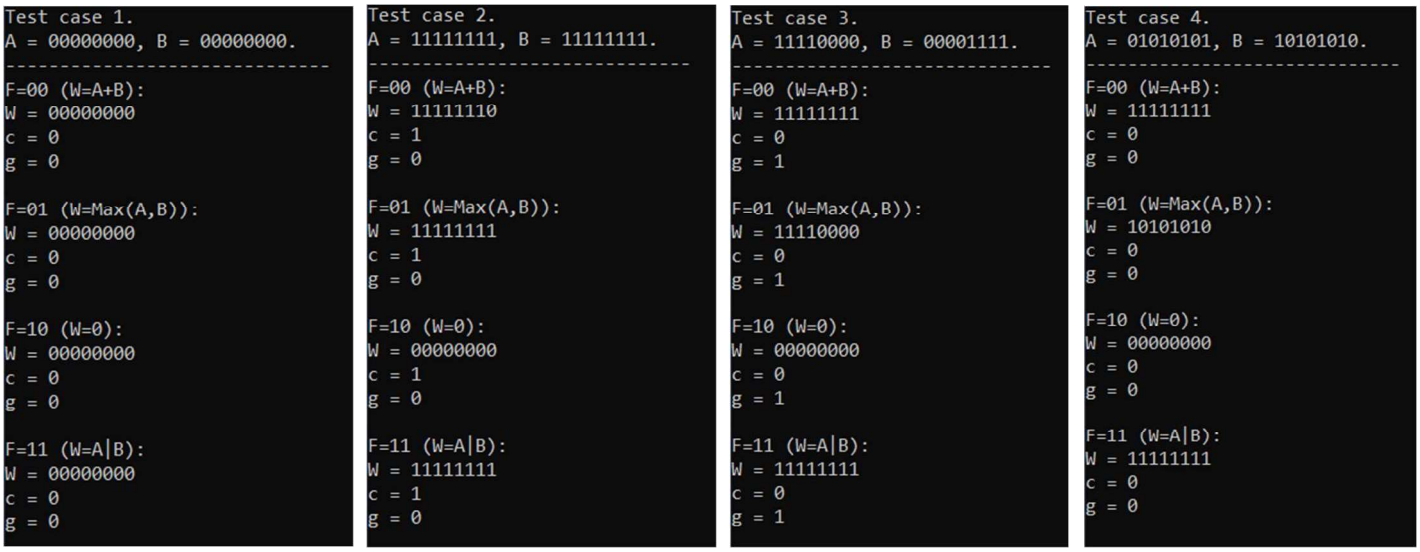

Fig. 9 – Test cases 1 to 4 for the designed circuit of 4-function ALU

So, the tests show that the circuit operates correctly.

3. SystemC RTL Modeling

3.1. Description of the Task

A processing element for approximate calculation of *1/(1-x)* is provided below. The input range is *0≤ x ≤0.5*. The circuit data input and output are 18-bit fixed-point numbers with two integer bits and sixteen fractional bits. The circuit receives its fractional-only data input via its 18-bit *XBus* input. This happens after the circuit receives a complete positive pulse on the *go* input. When the calculation is complete, the circuit issues a *ready* pulse. This happens after *sixteen* iterations.

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \cdots$$

3.1.1. RTL Structural Modeling

A. Develop the complete controller (CU) using SystemC. Use Huffman model.
B. Implement the datapath (DP) of the circuit using SystemC.
C. Complete the design of the circuit by wiring CU and DP in a top-level module.
D. Write a testbench and verify your design using different scenarios. Show VCD viewer results.

3.1.2. Functional Modeling

A. Develop a cycle-accurate behavioral model (BFM: Bus Functional Model) for the circuit.
B. Write a testbench and verify your design using the same scenarios as in Part D.

3.2. Solution

3.2.1 Diagrams and Pseudo Code

In this chapter, the diagrams of the designed circuit of 1/(1-x) calculator are presented (Fig. 10, Fig. 11, Fig. 12)

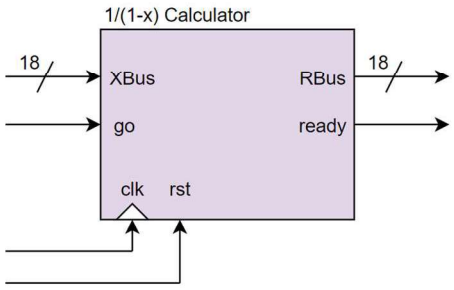

Fig. 10 – 1/(1-x) approximate calculator circuit

Pseudo code (Taylor's series):

```
term = 1;                    // T-register (term)
res = 1;                     // R-register (result)
for (i = 1; i < n; i++) {    // modulo-4 counter (n – number of terms)
        term = term * x;     // Multiplier, X-register (Xbus - input)
        res = res + term;    // Adder
}
```

The benefit of fixed-point arithmetic is that we can reuse the hardware built for integer arithmetic to perform real numbers arithmetic.

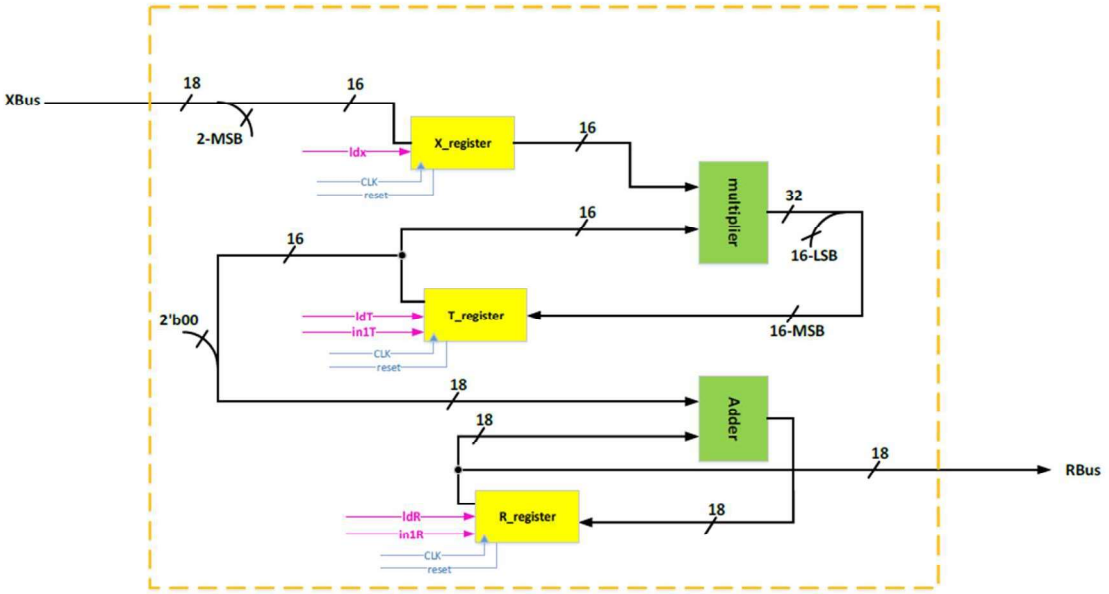

Fig. 11 – Datapath of 1/(1-x) approximate calculator

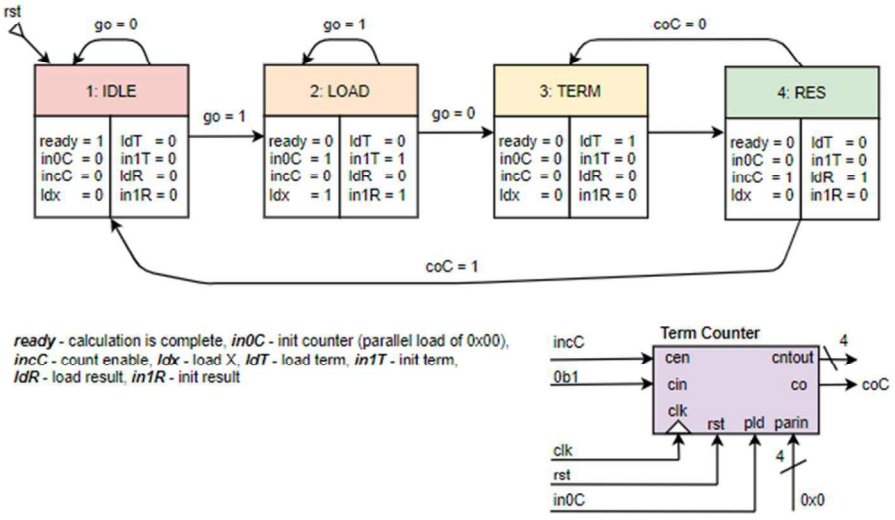

Fig. 12 – Controller (Moore machine) state diagram of 1/(1-x) approximate calculator

3.2.2. Test Results for the Design

In this chapter, the results of testing the designed **circuit** of 1/(1-x) calculator are presented (Fig. 13 - Fig. 16).

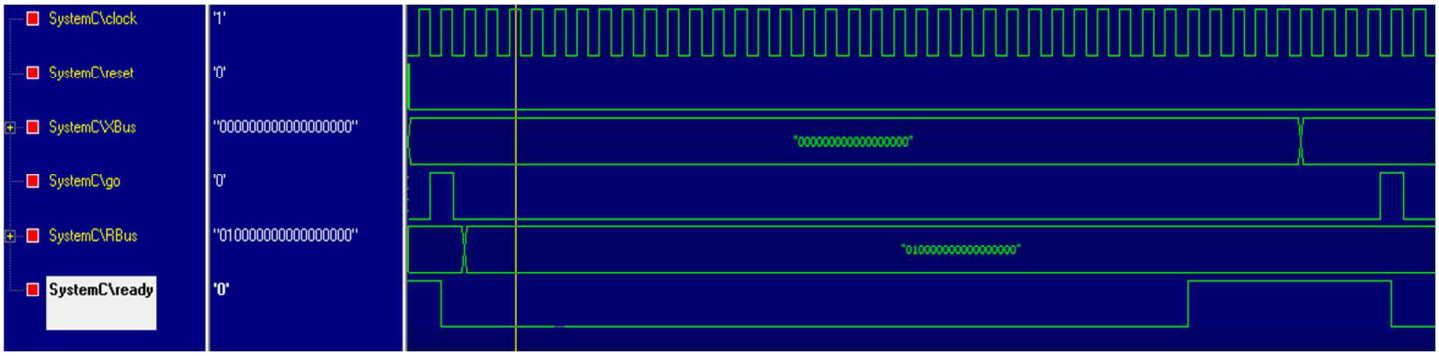

Fig. 13 – Test case 1: XBus = 0 (dec), RBus = 0100000000000000 (fixed point bin) = 1.0 (dec). Precise result = 1.0 (dec).

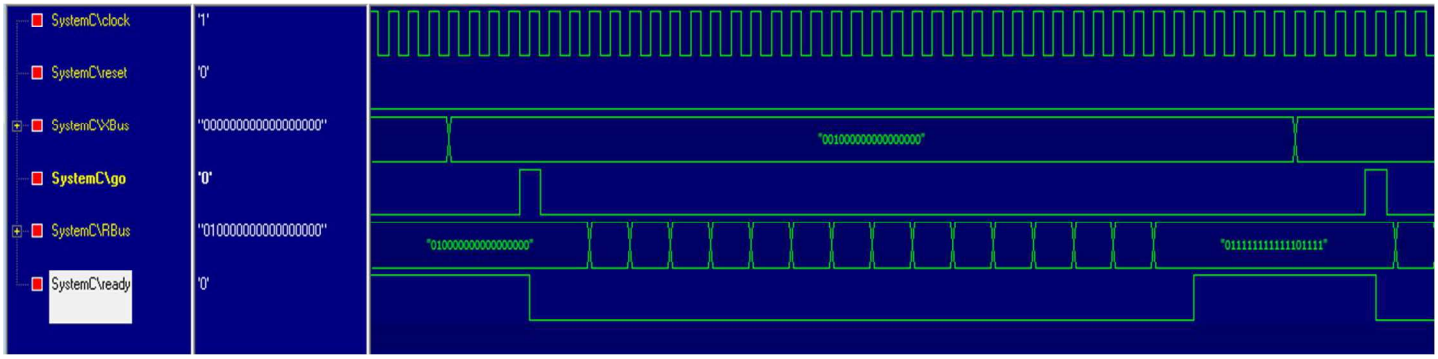

Fig. 14 – Test case 2: XBus = 0.5 (dec), RBus = 01111111111101111 (fixed point bin) = 1.9997406005859375 (dec). Precise result = 2 (dec).

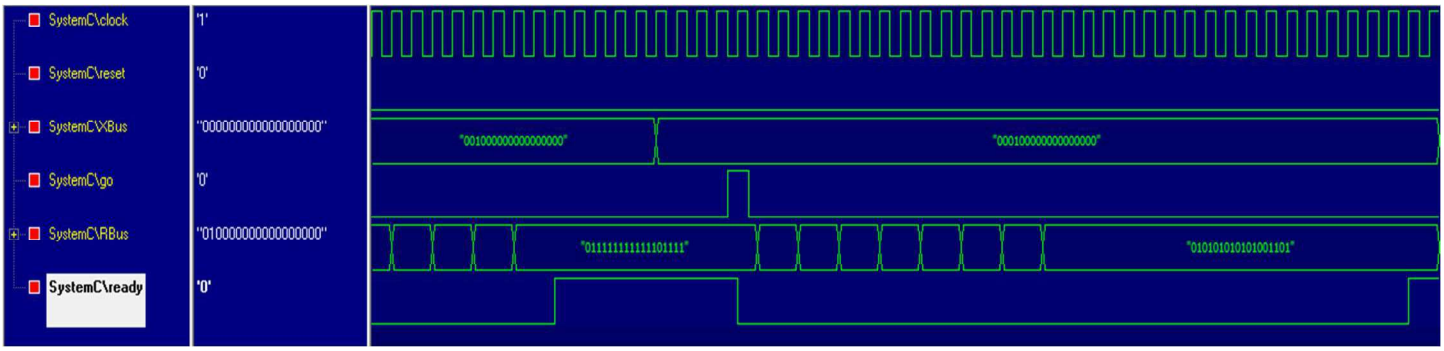

Fig. 15 – Test case 3: XBus = 0.25 (dec), RBus = 010101010101001101 (fixed point bin) = 1.3332061767578125 (dec). Precise result = 1.(33) (dec).

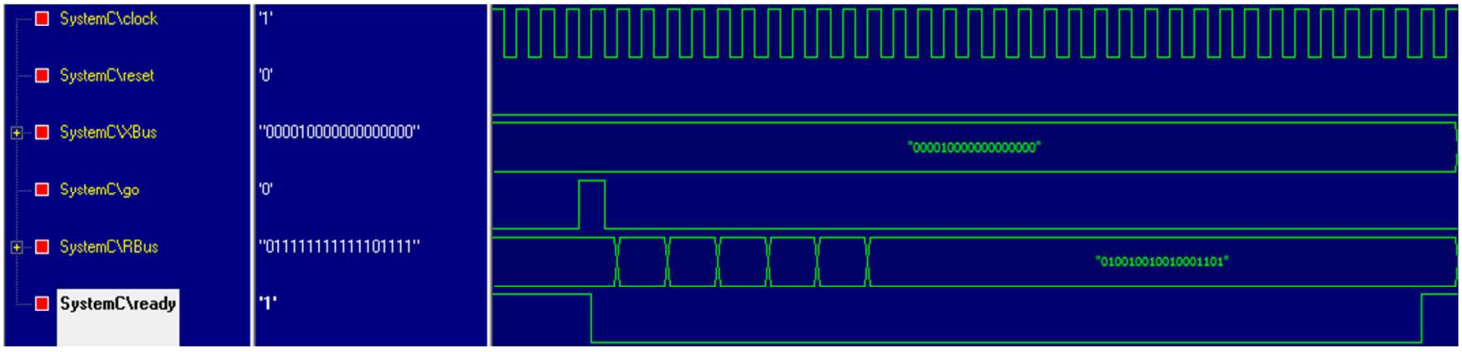

Fig. 16 – Test case 4: XBus = 0.125 (dec), RBus = 010010010010001101 (fixed point bin) = 1.1427764892578125 (dec). Precise result = 1.14285714 (dec).

So, the tests show that the circuit works correctly and produces the result with high precision.

3.2.3. Test Results for the BFM

The same tests, as in the previous chapter, were performed for the BFM of the circuit (Fig. 17 - Fig. 21).

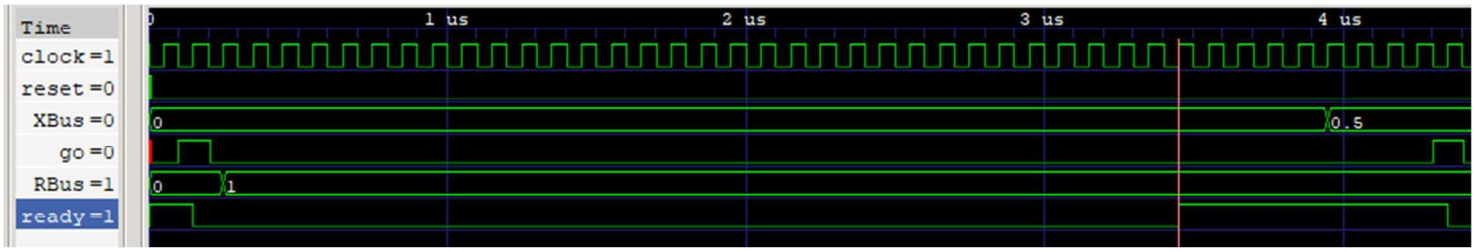

Fig. 17 - Test case 1: XBus = 0, RBus = 1.0 (dec). Precise result = 1.0.

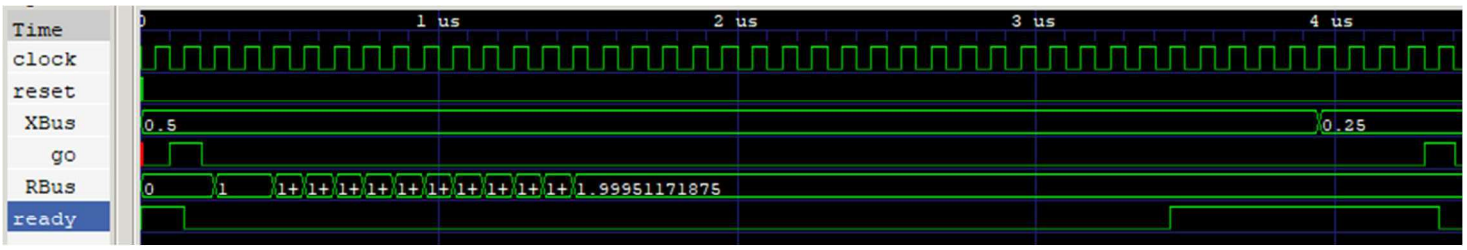

Fig. 18 - Fig. 19 – Test case 2: XBus = 0.5, RBus = 1.99951171875. Precise result = 2.

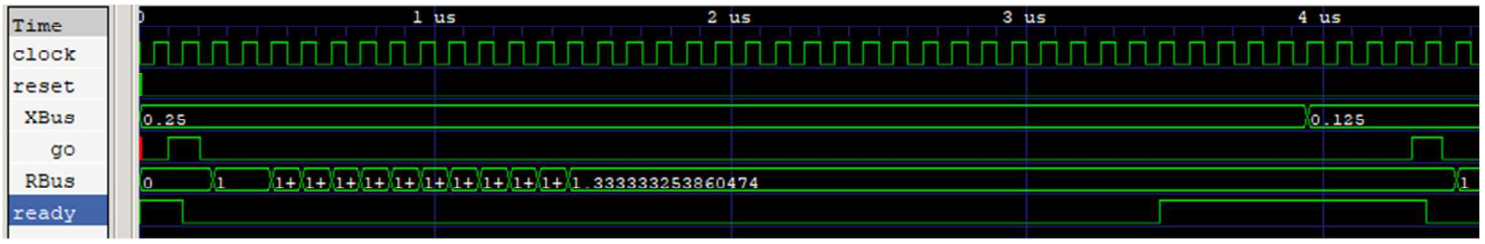

Fig. 20 - Test case 3: XBus = 0.25, RBus = 1.333333253860474. Precise result = 1.(33).

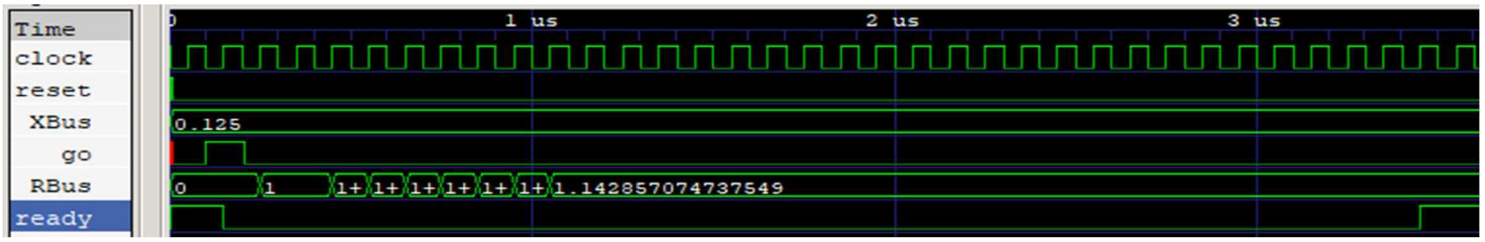

Fig. 21 - Test case 4: XBus = 0.125, RBus = 1.142857074737549. Precise result = 1.14285714 (dec).

So, the tests show that the BFM works correctly. The difference in the results between the design and the BFM are due to the use of the 16-bit fixed-point terms and the float datatype accordingly.

Also, it is of interest to compare the execution time for the design and for the BFM. For this purpose, conditional compilation statements (#ifdef - #endif) were added to *main.cpp*. So, when TIME_TEST is defined, the execution time is outputted in the file *time.txt* located in the current working (project) directory. The results were as following:

(1) for the design, it was 0.200000 sec

(2) for the BFM, it was 0.007000 sec

So, the BFM is 28.5 times faster.