UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

DEPARTMENT OF COMPUTER AND COMMUNICATION TECHNOLOGY

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

COMPUTER ENGINEERING

YEAR ONE SEMESTER THREE

Session: 202502

UCCE2023 DIGITAL SYSTEMS DESIGN

Practical Assignment:

Specifications

Instructor: Mok Kai Ming

Date of Submission: 9 May 2025

Group Members	Tutorial Group	Task (component)	Mobile Contact
Teamwork		Full Chip (Chip spec and verification)	
1 Ng Yu Heng	1	Hex keypad code generator, Operator keypad code generator synchronizer	011-55611695
2 Chow Bin Lin	1	ALU	017-9916614
3 Chow Bin Lin	1	Barrel shifter	017-9916614
4 Ng Jing Xuan	1	Input / output converter	017-5759279

Table of Contents

1 Architecture Specification: Written Spec 4
1.1 Functionality / feature 4
1.2 Operating Procedures and Human-Machine Interface 4
1.2.1 Example of application 4
1.2.2 Step-by-step way to use the 8-bit integer calculator 4
1.2.3 Flow chart (operation of the system) 5
1.3 Naming convention 6
1.4 Calculator Interface and I/O Pin Description 6
1.4.1 block interface 6
1.4.2 I/O Pin Description 7
1.4.3 Examples of Application with I/O Devices in Block Diagram 8
1.4.4 Timing diagram 9
1.5 Internal Operation 11
1.5.1 Logical/Functional View 11
1.6 System register 12
2 Micro-Architecture Specification 13
2.1 Design hierarchy 13
2.1.1 Major Components Function 13
2.2 Block-level functional partitioning 14
2.3 Full Chip Verilog Model 15
3 Architecture Specification: Verification Spec 17
3.1 Test plan 17
3.2 Testbench and Simulation result 30
3.2.1 Testbench 30
3.2.2 Simulation result 48
4 Micro-Architecture Specification (Block level) 99
4.1 Synchronizer 99
4.1.1 Functionality/features 99
4.1.2 Block interface and I/O pin description 99
4.1.2.1 Synchronizer block interface 99
4.1.2.2 I/O pin description 99
4.1.3 Internal Operation: Function Table 100
4.1.4 Timing Requirement 100

4.1.5 Schematic Diagram 101
4.1.6 System Verilog Model 103
4.1.7 Test Plan 105
4.1.8 Testbench and Simulation result.....106
4.1.8.1 Testbench 106
4.1.8.2 Simulation Result.....108
4.2 Operator keypad code generator.....110
4.2.1 Functionality/features 110
4.2.2 Block interface and I/O pin description 110
4.2.2.1 Operator keypad code generator block interface 110
4.2.2.2 I/O pin description 110
4.2.3 Internal Operation: Function Table 111
4.2.4 Timing Requirement 112
4.2.5 Schematic Diagram 113
4.2.6 System Verilog Model 115
4.2.7 Test Plan 118
4.2.8 Testbench and Simulation results 121
4.2.8.1 Testbench 121
4.2.8.2 Simulation result 125
4.3 Hex keypad code generator 129
4.3.1 Functionality/features 129
4.3.2 Block interface and I/O pin description 129
4.3.2.1 Hex keypad code generator block interface 129
4.3.2.2 I/O pin description 129
4.3.3 Internal Operation: Function Table 130
4.3.4 Timing Requirement 130
4.3.5 Schematic Diagram 132
4.3.6 System Verilog Model 137
4.3.7 Test Plan 140
4.3.8 Testbench and Simulation results 142
4.3.8.1 Testbench 142
4.3.8.2 Simulation result 145
4.4 ALU calculator 148
4.4.1 Functionality / features 148
4.4.2 Block Interface and I/O Pin Description 148
4.4.2.1 block interface 148

4.4.2.2 I/O pin description 149
4.4.3 Internal Operation: Function Table 150
4.4.4 Timing Requirement 151
4.4.5 Schematic Diagram 152
4.4.6 System Verilog Model 167
4.4.7 Test Plan 171
4.4.8 Testbench and Simulation results 176
4.4.8.1 Testbench 176
4.4.8.2 Simulation result 181
4.5 7-segment display decoder 187
4.5.1 Functionality/features 187
4.5.2 Block interface and I/O pin description 187
4.5.2.1 7-segment display decoder block interface 187
4.5.2.2 I/O pin description 187
4.5.4 Timing Requirement 189
4.5.5 Schematic Diagram 191
4.5.6 System Verilog Model 193
4.5.7 Test Plan 195
4.5.8 Testbench and Simulation result 198
4.5.8.1 Testbench 198
4.5.7.2 Simulation Result 202

1 Architecture Specification: Written Spec

1.1 Functionality / feature

- 8-bit integer-based computation system using programmable logic.
- Allow users to enter two operands and choose operations.
- Allow user to clear or reset the input or computation.
- Real-time display of entered digits using a 7-segment display.
- Preform arithmetic and bitwise operations such as: Addition, Subtraction, Multiplication, Division and also AND, OR, XOR, NOT.
- Perform bit-level shift and rotate operations.
- Display of the results using 7-segment display.
- Support digit from -255 to 255.

1.2 Operating Procedures and Human-Machine Interface

1.2.1 Example of application

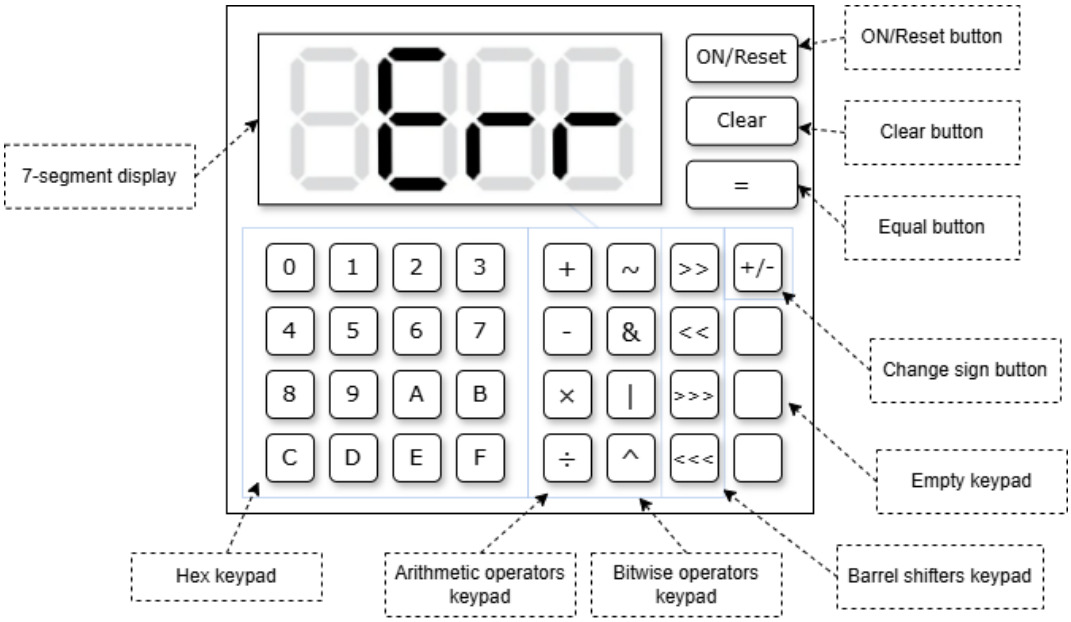

Diagram 1.2.1: Entire 8-bit integer calculator displaying error message when the display value overflowed or invalid button is pressed

1.2.2 Step-by-step way to use the 8-bit integer calculator

1) Press the **ON/Reset** button to start or reset the calculator.
2) Use the **hexadecimal keypad** to enter the first operand.
3) Use **operator keypad** to select the operation to perform on operand a and operand b.
4) Use the **hexadecimal keypad** to enter the second operand.
5) User can use the **Clear** button to clear the inputted operand before selecting operation or pressing the **Equal** button if they wish to change the operand value.
6) Once both operands and selected operation are entered, press the **Equal** button to execute the operation between two operands.
7) User can choose to continue the calculation by pressing a **operator key** and repeat steps above.
8) Press the **ON/Reset** button to initialize the 8-bit integer calculator to perform a new calculation.

1.2.3 Flow chart (operation of the system)

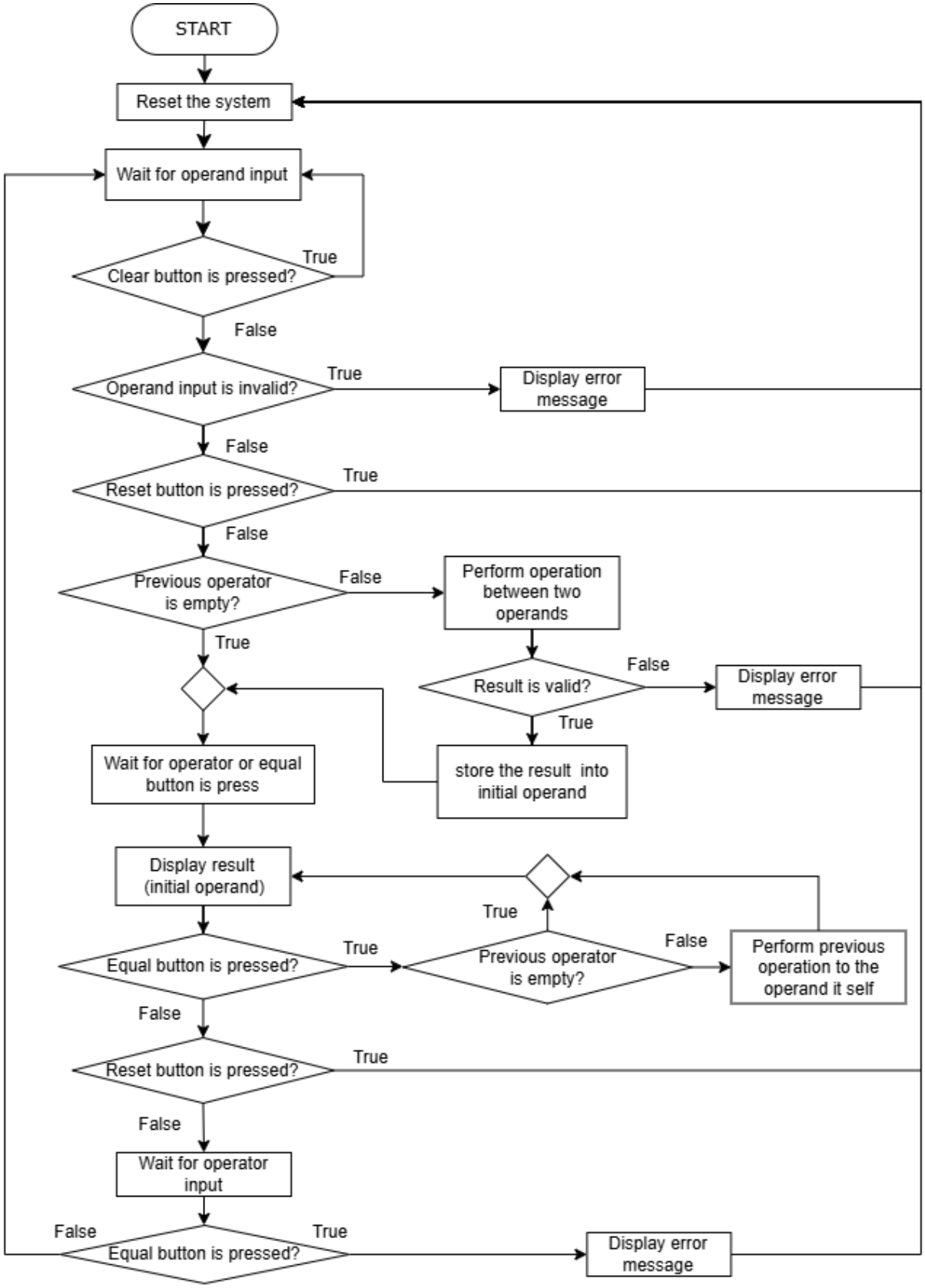

Diagram 1.2.3: Flow chat of operation and behaviour of the 8 bit integer calculator system

1.3 Naming convention

Name	Name used
input	i
Output	o
Block level	b
Sub-block	sb
system	sys
7 segment display decoder	dis
selector	sel
arithmetic and logic unit	alu
enable	en
synchronizer	sync
calculator	calc
register	r
Hex keypad code generator	hex
Operator	op

Table 1.3: Naming Convention of the calculator chip

1.4 Calculator Interface and I/O Pin Description

1.4.1 block interface

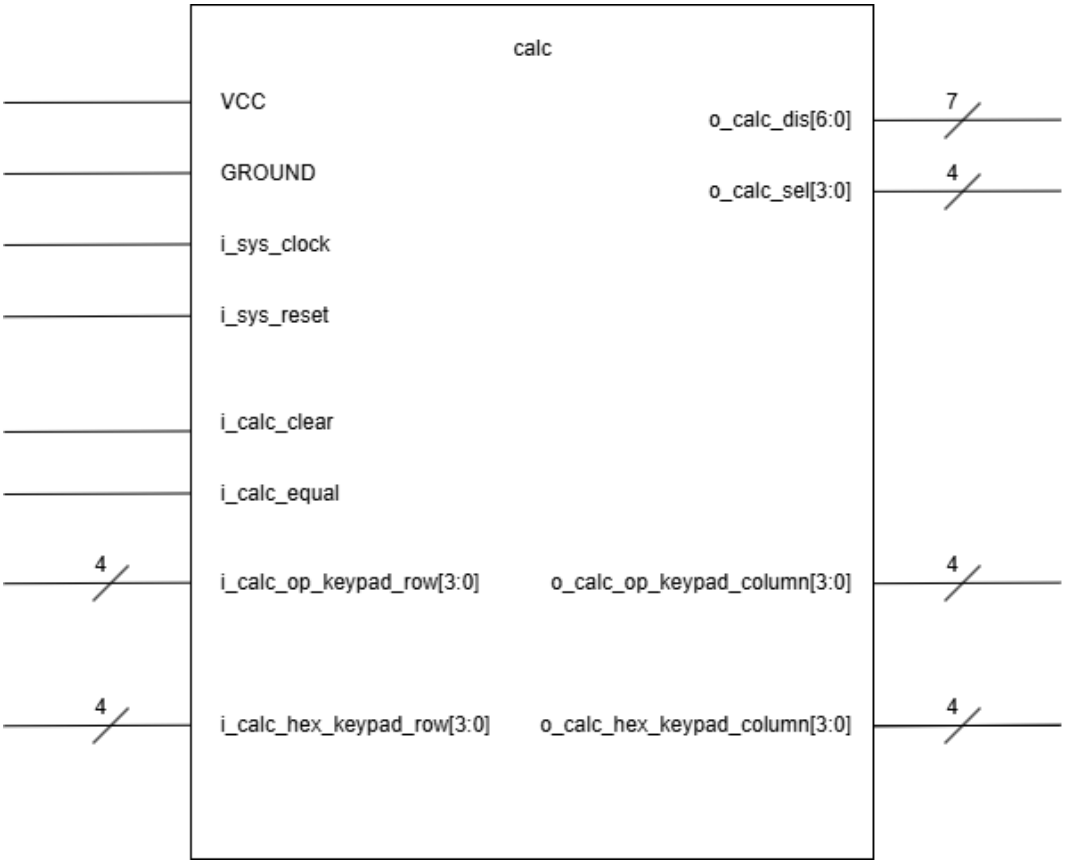

Diagram 1.4.1: block interface of 8-bit integer calculator

1.4.2 I/O Pin Description

Pin name: **Pin class:** **Pin function:**	VCC control to supply power to the calculator	**Source →Destination:**	ON/Reset button→Calculator
Pin name: **Pin class:** **Pin function:**	GROUND control to connect the calculator to the ground	**Source →Destination:**	Calculator→ground
Pin name: **Pin class:** **Pin function:**	i_sys_clock global to provide a system clock to the calculator	**Source →Destination:**	Clock generator → calculator
Pin name: **Pin class:** **Pin function:**	i_sys_reset global to reset the calculator	**Source →Destination:**	ON/Reset button→ calculator
Pin name: **Pin class:** **Pin function:**	i_calc_clear control to clear the number keypad input	**Source →Destination:**	Clear button→ calculator
Pin name: **Pin class:** **Pin function:**	i_calc_equal control to display the calculation result	**Source →Destination:**	Equal button→ Calculator
Pin name: **Pin class:** **Pin function:**	i_calc_op_keypad_row[3:0] data to detect which row of operator keypad is pressed	**Source →Destination:**	Operator keypad → calculator
Pin name: **Pin class:** **Pin function:**	i_calc_hex_keypad_row[3:0] data to detect which row of hexadecimal keypad is pressed	**Source →Destination:**	Hexadecimal keypad → calculator
Pin name: **Pin class:** **Pin function:**	o_calc_op_keypad_column[3:0] data to detect which column of operator keypad is pressed	**Source →Destination:**	Calculator → Operator keypad
Pin name: **Pin class:** **Pin function:**	o_ cal_hex_keypad_column[3:0] data to detect which column of hexadecimal keypad is pressed	**Source →Destination:**	Calculator → hexadecimal keypad
Pin name: **Pin class:** **Pin function:**	o_dis[6:0] data to transmit the data to the 7-segment display	**Source →Destination:**	calculator→ 7-segment display
Pin name: **Pin class:** **Pin function:**	o_sel[3:0] control to select 7 segment display	**Source →Destination:**	calculator→ 7-segment display

Table 1.4.2: pin description of 8-bit integer calculator

1.4.3 Examples of Application with I/O Devices in Block Diagram

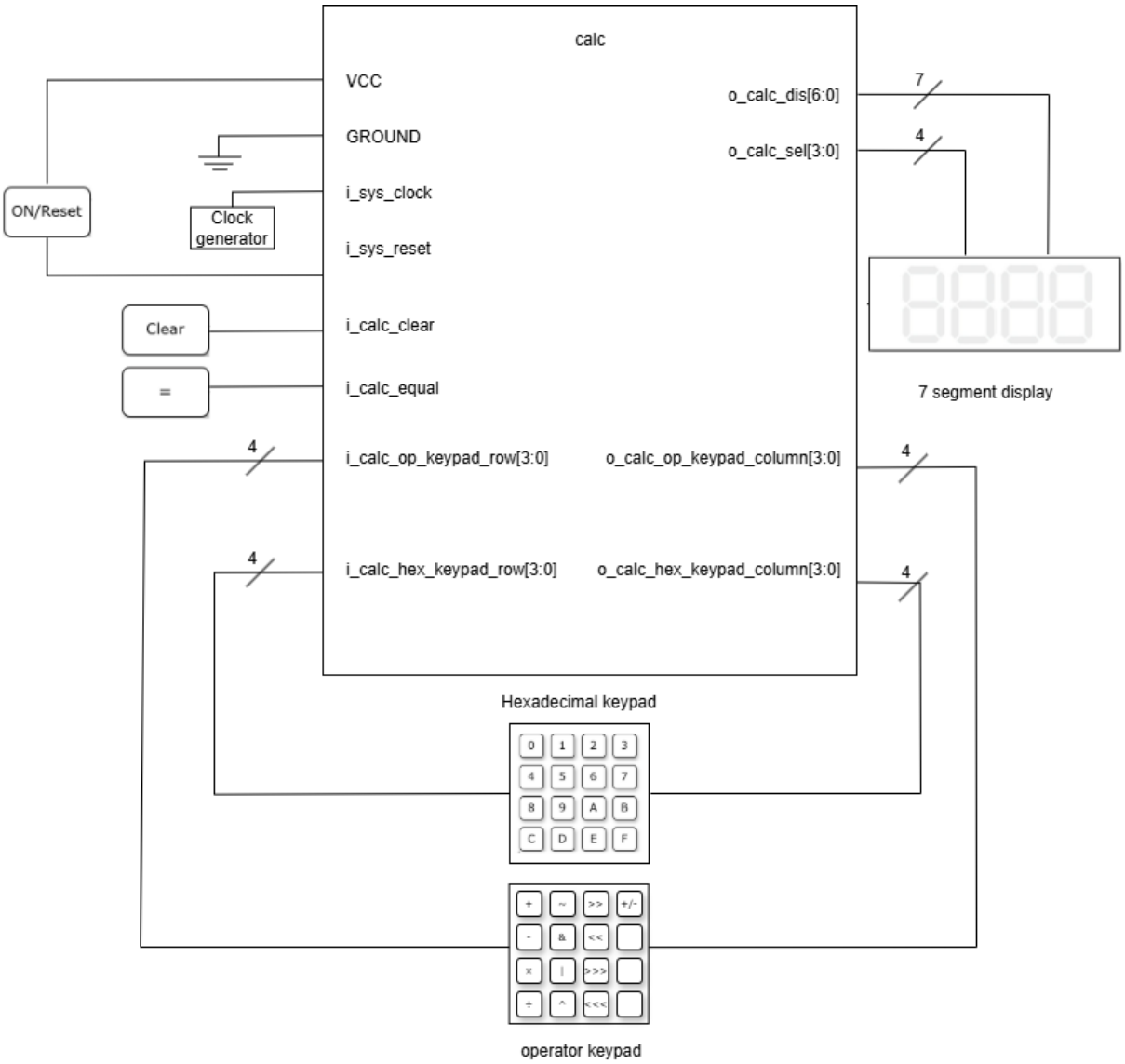

Diagram 1.4.3: Examples of application of 8-bit integer calculator

1.4.4 Timing diagram

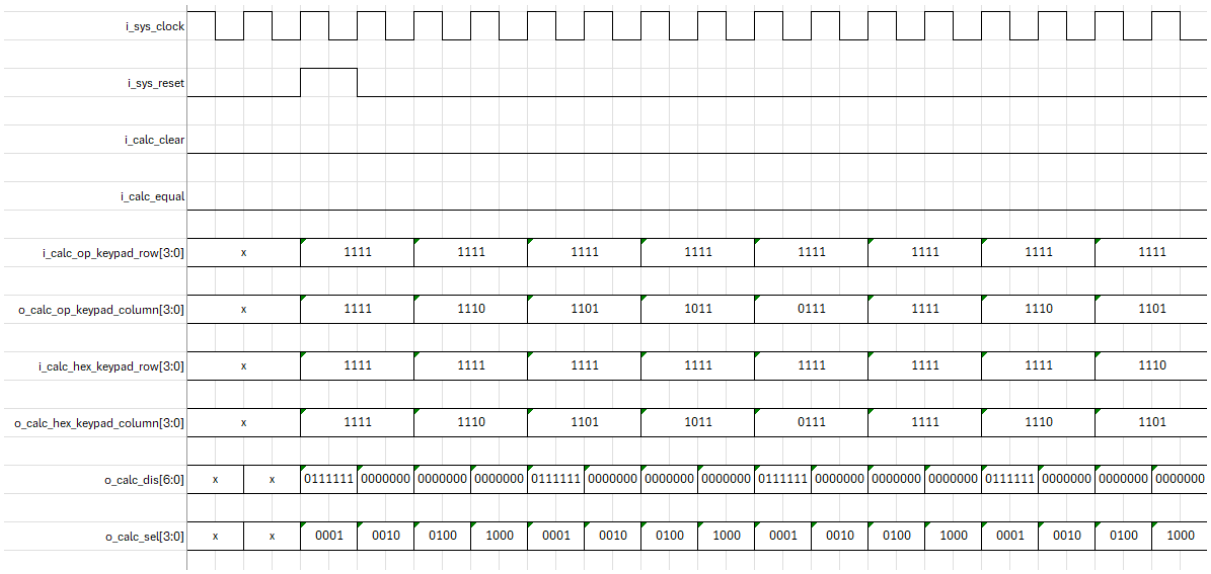

Diagram 1.4.4.1: timing diagram of 8-bit integer calculator

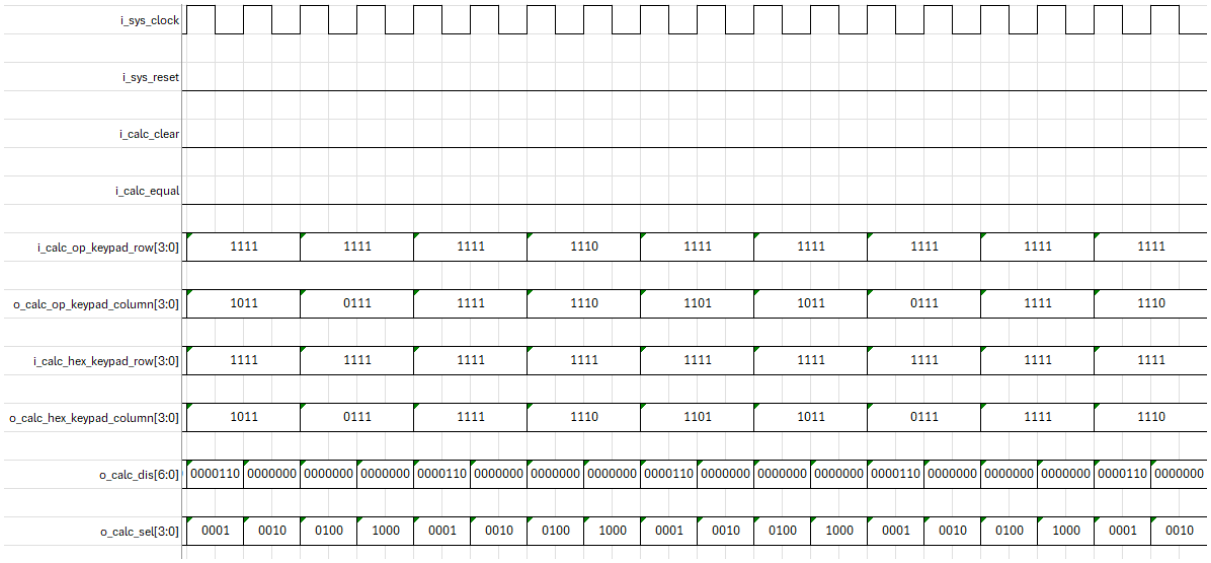

Diagram 1.4.4.2: timing diagram of 8-bit integer calculator

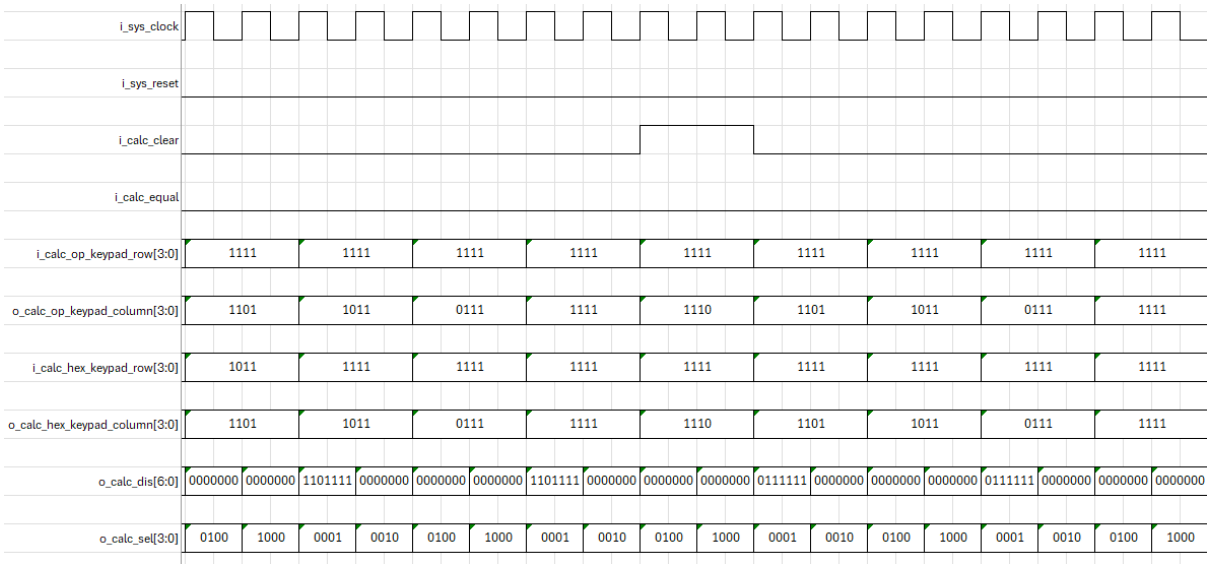

Diagram 1.4.4.3: timing diagram of 8-bit integer calculator

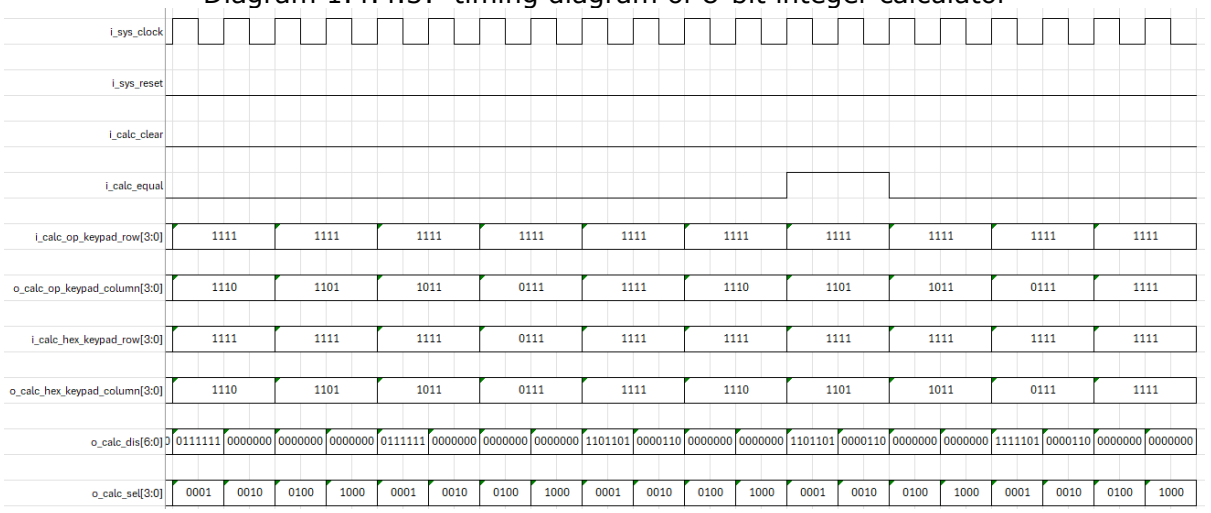

Diagram 1.4.4.4: timing diagram of 8-bit integer calculator

1.5 Internal Operation

1.5.1 Logical/Functional View

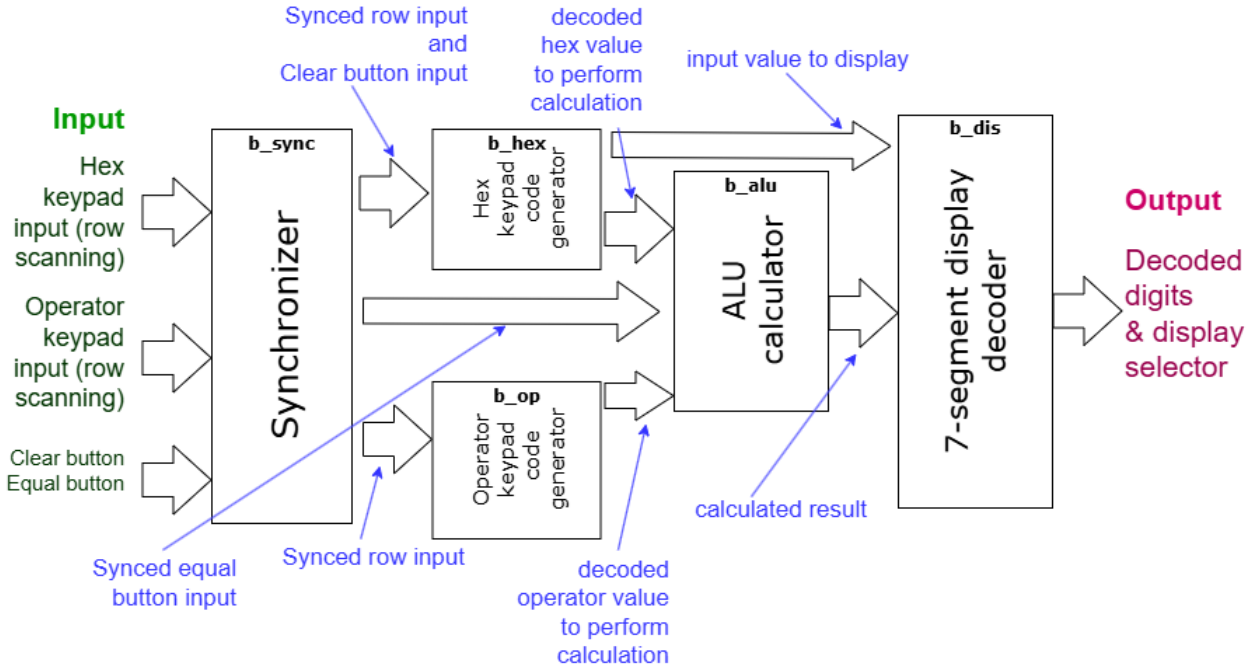

Diagram 1.5.1: Logical/Functional view of the 8-bit integer calculator

1.6 System register

Register	Type	Width	Reset value	Function
r_calc_operand	read/write	8bit	8'b0	Store the valid 8-bit input into operand for further calculation
r_calc_neg_flag	read/write	1bit	1'b0	Store the 1 bit negative flag even after user release the button for calculation purpose
r_calc_operator	read/write	4bit	4'b1111	Store the operator even after user release the button for calculation purpose
r_calc_result	read/write	9bit	9'b0	Store the negative flag and 8-bit integer result

Table 1.6: System registers of the 8-bit integer calculator

2 Micro-Architecture Specification

2.1 Design hierarchy

Chip Partitioning (Top Level) at System Level	Block Partitioning at Micro-Architecture Level
calc	b_sync
	b_hex
	b_op
	b_alu
	b_dis

Table 2.1: Design hierarchy of the 8-bit integer calculator

2.1.1 Major Components Function

Name	**Function**
b_sync	• Synchronize the keypad input to prevent metastability issues
b_op	• Reads the operator keypad to obtain user-entered numeric operands • Encodes operator input
b_hex	• Reads the hexadecimal keypad to obtain user-entered numeric operands • Convert pressed key input into 8-bit binary format • Encodes operand input • Indicate input overflow
b_alu	• Performs the arithmetic, logic and bitwise operations using the operands and operator input •
b_dis	• Converts binary output from the ALU and hex keypad code generator into displayable 7-segment codes

Table 2.1.1: Major component functions of the 8-bit integer calculator

2.2 Block-level functional partitioning

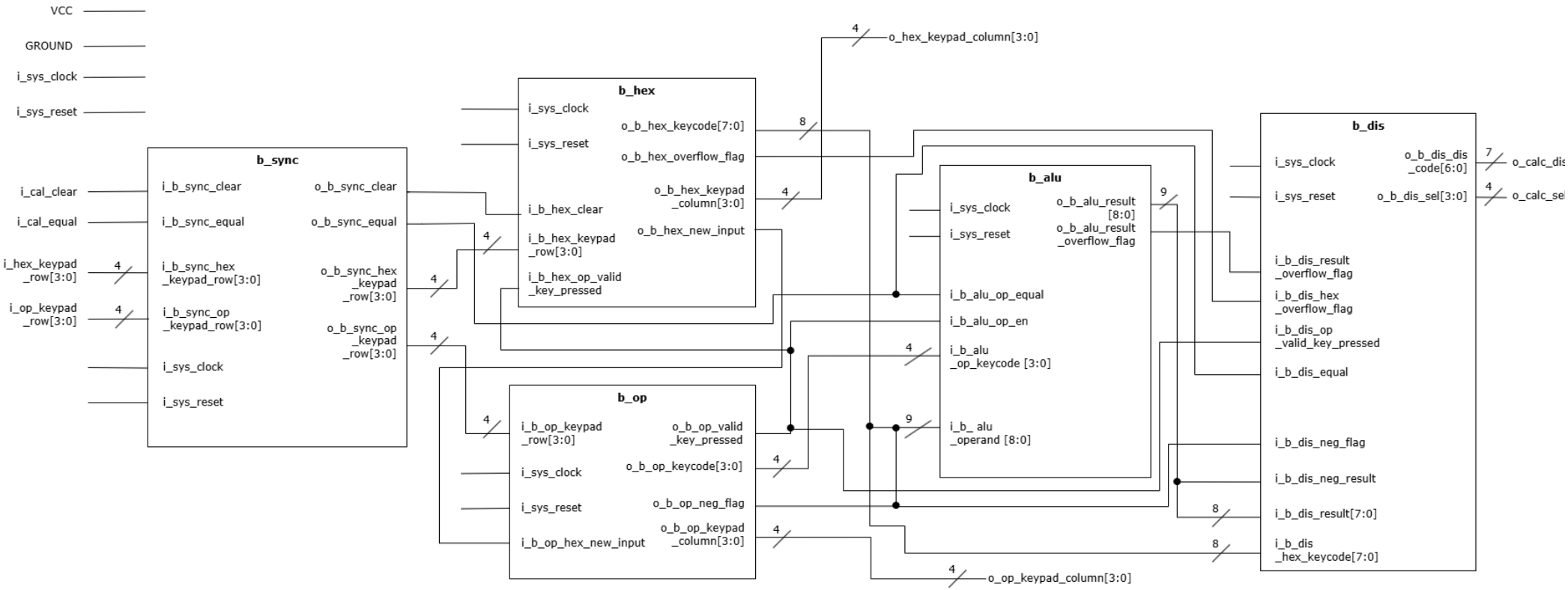

Diagram 2.2: Block-level functional partitioning of 8-bit integer calculator system

2.3 Full Chip Verilog Model

```
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Jing Xuan
//
// Create Date: 20.04.2025 12:23:51
// File Name: calc.sv
// Module Name: cal
// Project Name: 8-bit integer calculator
// Code Type: RTL level
// Description: Modelling of 8-bit integer calculator
//
//////////////////////////////////////////////////////////////////////////////////
module calc(
input logic VCC,
input logic GROUND,
input logic i_sys_clock,
input logic i_sys_reset,
input logic i_calc_clear,
input logic i_calc_equal,
input logic [3:0] i_calc_op_keypad_row,
input logic [3:0] i_calc_hex_keypad_row,
output logic [3:0] o_calc_op_keypad_column,
output logic [3:0] o_calc_hex_keypad_column,
output logic [7:0] o_calc_dis,
output logic [3:0] o_calc_sel);

wire clear, equal, valid, hex_overflow, overflow, hex_new_input;
wire [3:0] hex_row;
wire [3:0] op_row;
logic r_calc_neg_flag;
logic [3:0] r_calc_operator;
logic [7:0] r_calc_operand;
logic [8:0] r_calc_result;

b_sync
sync
(.i_sys_clock(i_sys_clock),
.i_sys_reset(i_sys_reset),
.i_b_sync_clear(i_calc_clear),
.i_b_sync_equal(i_calc_equal),
.i_b_sync_op_keypad_row(i_calc_op_keypad_row),
.i_b_sync_hex_keypad_row(i_calc_hex_keypad_row),
.o_b_sync_clear(clear),
.o_b_sync_equal(equal),
.o_b_sync_hex_keypad_row(hex_row),
.o_b_sync_op_keypad_row(op_row));


b_hex
hex
(.i_sys_clock(i_sys_clock),
.i_sys_reset(i_sys_reset),
.i_b_hex_keypad_row(hex_row),
.i_b_hex_clear(clear),
.i_b_hex_op_valid_key_pressed(valid),
.o_b_hex_keypad_column(o_calc_hex_keypad_column),
```

```
.o_b_hex_keycode(r_calc_operand),
.o_b_hex_overflow_flag(hex_overflow),
.o_b_hex_new_input(hex_new_input));


b_op
op
(.i_sys_clock(i_sys_clock),
.i_sys_reset(i_sys_reset),
.i_b_op_hex_new_input(hex_new_input),
.i_b_op_keypad_row(op_row),
.o_b_op_keycode(r_calc_operator),
.o_b_op_valid_key_pressed(valid),
.o_b_op_neg_flag(r_calc_neg_flag),
.o_b_op_keypad_column(o_calc_op_keypad_column));


b_alu
alu
(.i_sys_clock(i_sys_clock),
.i_sys_reset(i_sys_reset),
.i_b_alu_equal(equal),
.i_b_alu_en(valid),
.i_b_alu_op_keycode(r_calc_operator),
.i_b_alu_operand({r_calc_neg_flag,r_calc_operand}),
.o_b_alu_result(r_calc_result),
.o_b_alu_overflow_flag(overflow));


b_dis
dis
(.i_sys_clock(i_sys_clock),
.i_sys_reset(i_sys_reset),
.i_b_dis_result_overflow_flag(overflow),
.i_b_dis_hex_overflow_flag(hex_overflow),
.i_b_dis_op_valid_key_pressed(valid),
.i_b_dis_equal(equal),
.i_b_dis_neg_flag(r_calc_neg_flag),
.i_b_dis_neg_result(r_calc_result[8]),
.i_b_dis_result(r_calc_result[7:0]),
.i_b_dis_hex_keycode(r_calc_operand),
.o_b_dis_dis_code(o_calc_dis),
.o_b_dis_sel(o_calc_sel));

endmodule
```

3 Architecture Specification: Verification Spec

3.1 Test plan

- The output will set only one 7 segment display to light up around 10ns each time (triggered by clock).
- It will look like all 7-segment display is turned on as the speed is very fast

No.	Test Case	Description of test vector Generation	Expected Output	Status
1	Reset input	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 4. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 5. Set i_sys_reset = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 0)	PASS
2	Clear input	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b0111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 3. Set i_calc_clear = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 0)	PASS
3	Input overflow (input 555)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b0111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1101 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b0111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1101 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b0111	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h50 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h79 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS

		6. Set i_calc_hex_keypad_row[3:0] = 4'b1101 for 2 clock cycle		
4	"Add" operation (2+3)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 4. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 6. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h6D ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 5)	PASS
5	"Add" with negative number (-2+3)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 4. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 8. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h06 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display 1)	PASS
6	Addition overflow (FF+F)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h50	PASS

		2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 4. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 8. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h79 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	
7	"Minus" operation (7-3)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b1101 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 4. Set i_calc_op_keypad_row[3:0] = 4'b1101 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 6. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h66 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 4)	PASS
8	"Minus" with negative number (-7-8)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b1101 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 4. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h6D ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h06 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00	PASS

		5. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 6. Set i_calc_op_keypad_row[3:0] = 4'b1101 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 8. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display -15)	
9	Minus result overflow (-FE-2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 8. Set i_calc_op_keypad_row[3:0] = 4'b1101 for 2 clock cycle 9. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 10. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 11. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h79 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h50 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS
10	"Multiply" operation (AxB)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h06	PASS

		4. Set i_calc_op_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 6. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 7. Set i_calc_equal = 1 for 2 clock cycle	iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h06 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 110)	
11	"Multiply" with negative number (-AxB)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 4. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 6. Set i_calc_op_keypad_row[3:0] = 4'b1011 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 8. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h06 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h06 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display -110)	PASS
12	Multiplication overflow (20xE)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 4. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 6. Set i_calc_op_keypad_row[3:0] = 4'b1011 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h79 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h50 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS

		7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle		
13	"Divide" operation (F/2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 4. Set i_calc_op_keypad_row[3:0] = 4'b0111 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 6. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h07 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 7)	PASS
14	"Divide" with negative number (F/-2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 4. Set i_calc_op_keypad_row[3:0] = 4'b0111 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 6. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 8. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h07 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display -7)	PASS

15	"Divide" by zero (invalid)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 2. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 3. Wait for o_calc_op_keypad_column[3:0] = 4'b1011 4. Set i_calc_op_keypad_row[3:0] = 4'b0111 for 2 clock cycle 5. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 6. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h79 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h50 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS
16	"NOT" operation (~A5)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 2. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b0111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1101 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b0111 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h6F iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 90)	PASS
17	"AND" operation (FA & A0)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h7D iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h06 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00	PASS

		5. Wait for o_calc_op_keypad_column[3:0] = 4'b0111 6. Set i_calc_op_keypad_row[3:0] = 4'b1101 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 10. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 11. Set i_calc_equal = 1 for 2 clock cycle	(display 160)	
18	"OR" operation (FA | A0)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b0111 6. Set i_calc_op_keypad_row[3:0] = 4'b1011 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 10. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h6D iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h5B iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 250)	PASS

		11. Set i_calc_equal = 1 for 2 clock cycle		
19	"XOR" operation (FA ^ A0)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b0111 6. Set i_calc_op_keypad_row[3:0] = 4'b0111 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Wait for o_calc_hex_keypad_column[3:0] = 4'b1011 10. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 11. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h67 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 90)	PASS
20	"Bitwise right shift" operation (FA >> 2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1111 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h5B ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h7D iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 62)	PASS

		8. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle		
21	"Bitwise left shift" operation (FA << 9)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1111 6. Set i_calc_op_keypad_row[3:0] = 4'b1101 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b0111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 0)	PASS
22	"Arithmetic right shift" operation (FA >> >2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1111 6. Set i_calc_op_keypad_row[3:0] = 4'b1011 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h5B ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h7D iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 62)	PASS

		7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle		
23	"Arithmetic right shift" of negative number (-FA >>> 2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_op_keypad_column[3:0] = 4'b1111 8. Set i_calc_op_keypad_row[3:0] = 4'b1011 for 2 clock cycle 9. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 10. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 11. Set i_calc_equal = 1 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h4F ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h7D iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display -63)	PASS
24	"Arithmetic left shift" operation (FA <<<2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h66 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h5B iii) o_b_dis_sel = 4'b0100	PASS

		4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1111 6. Set i_calc_op_keypad_row[3:0] = 4'b0111 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 9. Set i_calc_equal = 1 for 2 clock cycle	o_b_dis_dis_code = 7'h00 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display -24)	
25	Invalid shifting (FA >> -2)	1. Wait for o_calc_hex_keypad_column[3:0] = 4'b1110 2. Set i_calc_hex_keypad_row[3:0] = 4'b0111 for 2 clock cycle 3. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 4. Set i_calc_hex_keypad_row[3:0] = 4'b1011 for 2 clock cycle 5. Wait for o_calc_op_keypad_column[3:0] = 4'b1111 6. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle 7. Wait for o_calc_hex_keypad_column[3:0] = 4'b1111 8. Set i_calc_hex_keypad_row[3:0] = 4'b1110 for 2 clock cycle 9. Wait for o_calc_op_keypad_column[3:0] = 4'b1110 10. Set i_calc_op_keypad_row[3:0] = 4'b1110 for 2 clock cycle	• Loop output between different value continuously i) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h79 ii) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 iii) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h50 iv) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS

		11. Set i_calc_equal = 1 for 2 clock cycle		

3.2 Testbench and Simulation result

3.2.1 Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Jing Xuan
//
// Create Date: 20.04.2025 16:01:24
// File Name: tb_calc.sv
// Module Name: tb_calc
// Project Name: 8-bit integer calculator
// Code Type: Behavioural
// Description: Testbench for 8-bit integer calculator
//
//////////////////////////////////////////////////////////////////////////////////
`define PERIOD_CLK 10

module tb_calc();

logic tb_VCC, tb_GROUND, tb_sys_clock, tb_sys_reset, tb_calc_clear, tb_calc_equal;
logic [3:0] tb_calc_op_keypad_row;
logic [3:0] tb_calc_hex_keypad_row;
wire  [3:0] tb_calc_op_keypad_column;
wire  [3:0] tb_calc_hex_keypad_column;
wire  [6:0] tb_calc_dis;
wire  [3:0] tb_calc_sel;


calc
dut_calc
(.VCC(tb_VCC),
.GROUND(tb_GROUND),
.i_sys_clock(tb_sys_clock),
.i_sys_reset(tb_sys_reset),
.i_calc_clear(tb_calc_clear),
.i_calc_equal(tb_calc_equal),
.i_calc_op_keypad_row(tb_calc_op_keypad_row),
.i_calc_hex_keypad_row(tb_calc_hex_keypad_row),
.o_calc_op_keypad_column(tb_calc_op_keypad_column),
.o_calc_hex_keypad_column(tb_calc_hex_keypad_column),
.o_calc_dis(tb_calc_dis),
.o_calc_sel(tb_calc_sel));


initial begin
tb_sys_clock = 1'b1;
forever #(`PERIOD_CLK/2) tb_sys_clock = ~tb_sys_clock;
end

//initialization
initial begin
tb_sys_reset = 1'b1;
#(`PERIOD_CLK*4)
tb_VCC = 1'b1;
tb_GROUND = 1'b0;
tb_sys_reset = 1'b0;
```

```
tb_calc_clear = 1'b0;
tb_calc_equal = 1'b0;
tb_calc_hex_keypad_row = 4'b1111;
tb_calc_op_keypad_row = 4'b1111;


//------------------------------------------------------------
// Test Case 1: Reset input
//------------------------------------------------------------
wait (tb_calc_hex_keypad_column == 4'b1111)  //represent 1011 since there are 3
clock cycle delay for row
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)  // represent 1110
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2)  //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //set reset high for 2 clock cycle and back to low
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 2: Clear input
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait (tb_calc_hex_keypad_column == 4'b0111)  // represent 1101
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_clear = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_clear = 1'b0;


//------------------------------------------------------------
// Test Case 3: Input overflow  (input 555 -> display Err)
//------------------------------------------------------------
#(`PERIOD_CLK*8)  //delay after reset before start next test case
wait (tb_calc_hex_keypad_column == 4'b0111)  // represent 1101
tb_calc_hex_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b0111)  // represent 1101
tb_calc_hex_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
```

```
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b0111)  // represent 1101
tb_calc_hex_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 4: Add operation (2+3)
//------------------------------------------------------------
#(`PERIOD_CLK*8)  //delay after reset before start next test case
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 5: Add with negative number (-2+3)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;
```

```
#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 6: Addition overflow (FF+F)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
```

```
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 7: Minus operation (7-3)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 8: Minus with negative number (-7-8)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
```

```
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 9: Minus result overflow (-FE-2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)   //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
```

```
tb_calc_op_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 10: Multiply operation (AxB)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
```

```
// Test Case 11: Multiply with negative number (-AxB)
//------------------------------------------------------------
#(`PERIOD_CLK*4)   //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 12: Multiplication overflow (20xE)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
```

```
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 13: Divide operation (F/2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;
```

```
//------------------------------------------------------------
// Test Case 14: Divide with negative number (F/-2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 15: Divide by zero (0/0)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1011)
tb_calc_op_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;
```

```
#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 16: Not operation (~A5)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b0111)
tb_calc_hex_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b0111)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*12)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 17: AND operation (FA & A0)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;
```

```
#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b0111)
tb_calc_op_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 18: OR operation (FA | A0)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b0111)
tb_calc_op_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
```

```
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*6)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 19: XOR operation (FA ^ A0)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b0111)
tb_calc_op_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
```

```
wait (tb_calc_hex_keypad_column == 4'b1011)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*6)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 20: Bitwise right shift operation (FA>>2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1111)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;
```

```
//------------------------------------------------------------
// Test Case 21: Bitwise left shift operation (FA<<9)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1111)
tb_calc_op_keypad_row = 4'b1101;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b0111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 22: Arithmetic right shift operation (FA>>>2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
```

```
wait (tb_calc_op_keypad_column == 4'b1111)
tb_calc_op_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 23: Arithmetic right shift of negative number (-FA>>>2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*2)
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1111)
tb_calc_op_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;
```

```
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 24: Arithmetic left shift operation (FA<<<2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1111)
tb_calc_op_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;


//------------------------------------------------------------
// Test Case 25: Invalid shifting (FA>>-2)
//------------------------------------------------------------
#(`PERIOD_CLK*4)  //delay after reset before start next test case
```

```
wait(tb_calc_sel == 4'b0001)
wait (tb_calc_hex_keypad_column == 4'b1110)
tb_calc_hex_keypad_row = 4'b0111;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1011;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_op_keypad_column == 4'b1111)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
wait (tb_calc_hex_keypad_column == 4'b1111)
tb_calc_hex_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_hex_keypad_row = 4'b1111;

#(`PERIOD_CLK*5)
#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait (tb_calc_op_keypad_column == 4'b1110)
tb_calc_op_keypad_row = 4'b1110;
#(`PERIOD_CLK*2) //set row for 2 cycle
tb_calc_op_keypad_row = 4'b1111;

#(`PERIOD_CLK*4)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_calc_equal = 1'b1;
#(`PERIOD_CLK*2)
tb_calc_equal = 1'b0;
#(`PERIOD_CLK*8)  //delay to see the complete waveform of display
wait(tb_calc_sel == 4'b0001)
tb_sys_reset = 1'b1; //reset for the next test case
#(`PERIOD_CLK*2)
tb_sys_reset = 1'b0;

end
endmodule
```

3.2.2 Simulation result

Reference
tb_calc_dis = 3F; display "0" on 7-segment display
tb_calc_dis = 06; display "1" on 7-segment display
tb_calc_dis = 5B; display "2" on 7-segment display
tb_calc_dis = 4F; display "3" on 7-segment display
tb_calc_dis = 66; display "4" on 7-segment display
tb_calc_dis = 6D; display "5" on 7-segment display
tb_calc_dis = 7D; display "6" on 7-segment display
tb_calc_dis = 07; display "7" on 7-segment display
tb_calc_dis = 7F; display "8" on 7-segment display
tb_calc_dis = 6F; display "9" on 7-segment display
tb_calc_dis = 40; display "-" (negative) on 7-segment display
tb_calc_dis = 50; display "r" on 7-segment display
tb_calc_dis = 79; display "E" on 7-segment display
tb_calc_dis = 00; display " " (blank) on 7-segment display

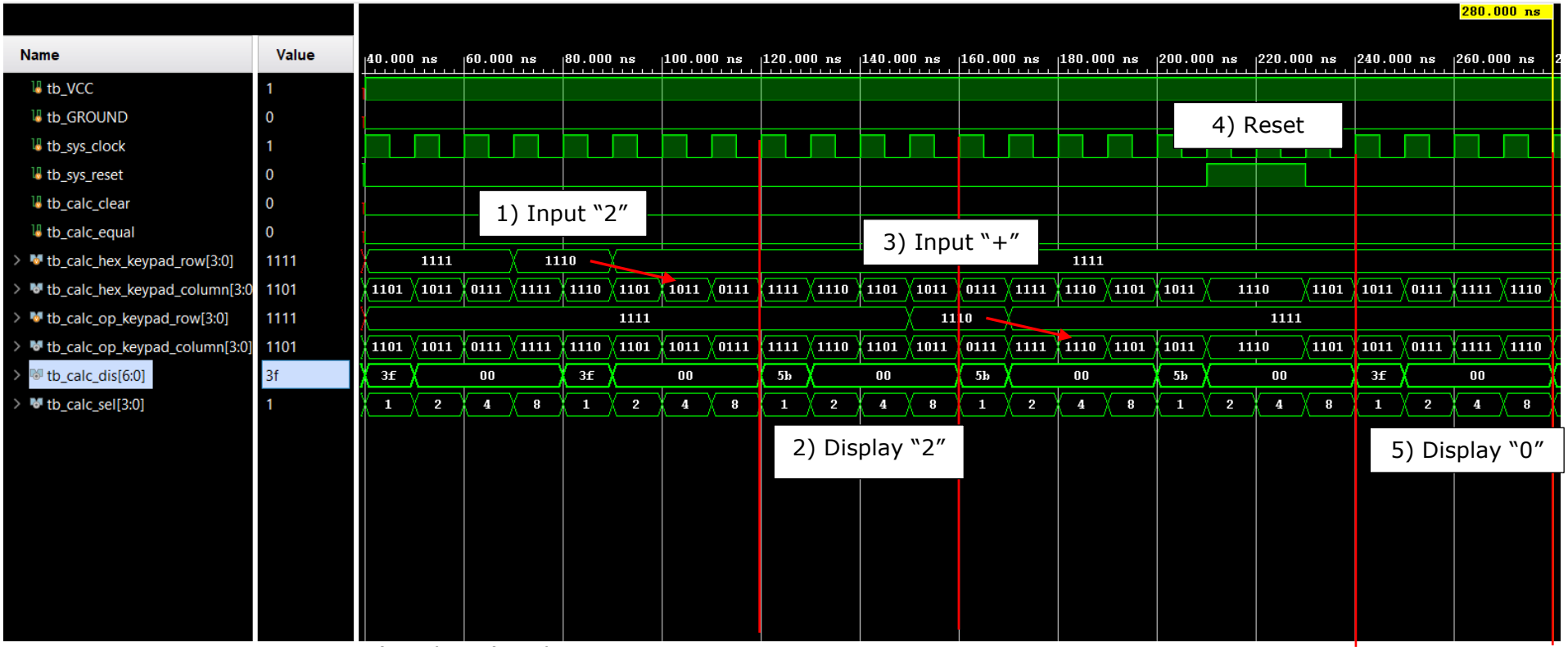

Figure 3.2.2.1 Test case 1: Input (number 2) and reset

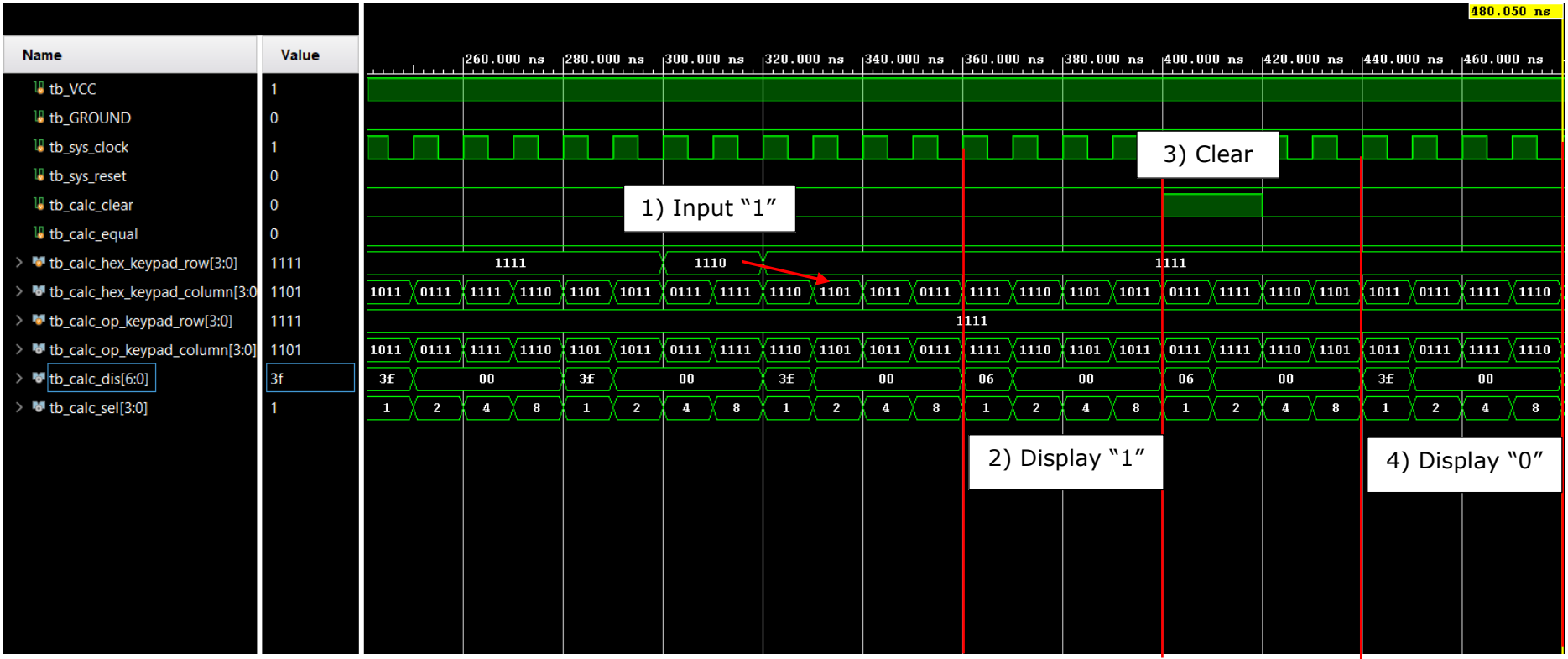

Figure 3.2.2.2 Test case 2: Input (number 1) and clear

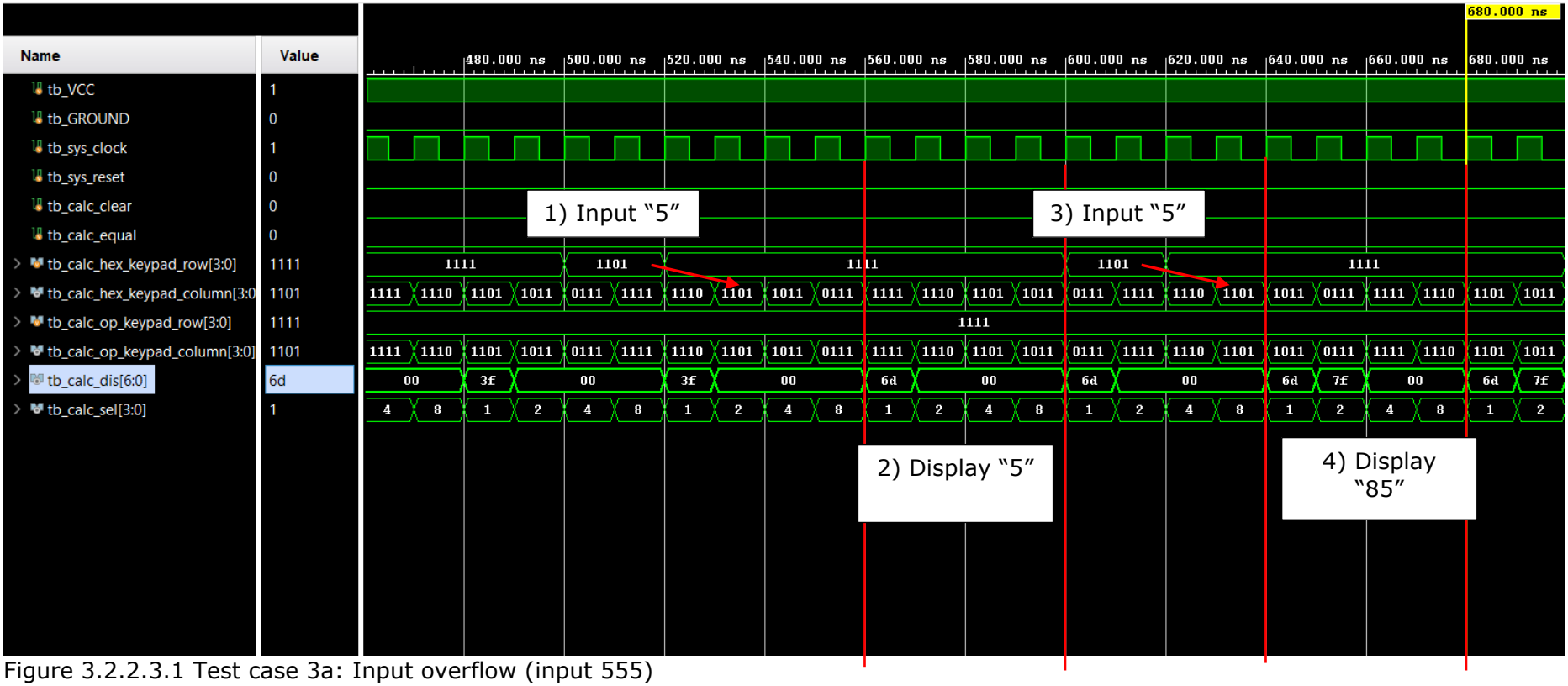

Figure 3.2.2.3.1 Test case 3a: Input overflow (input 555)

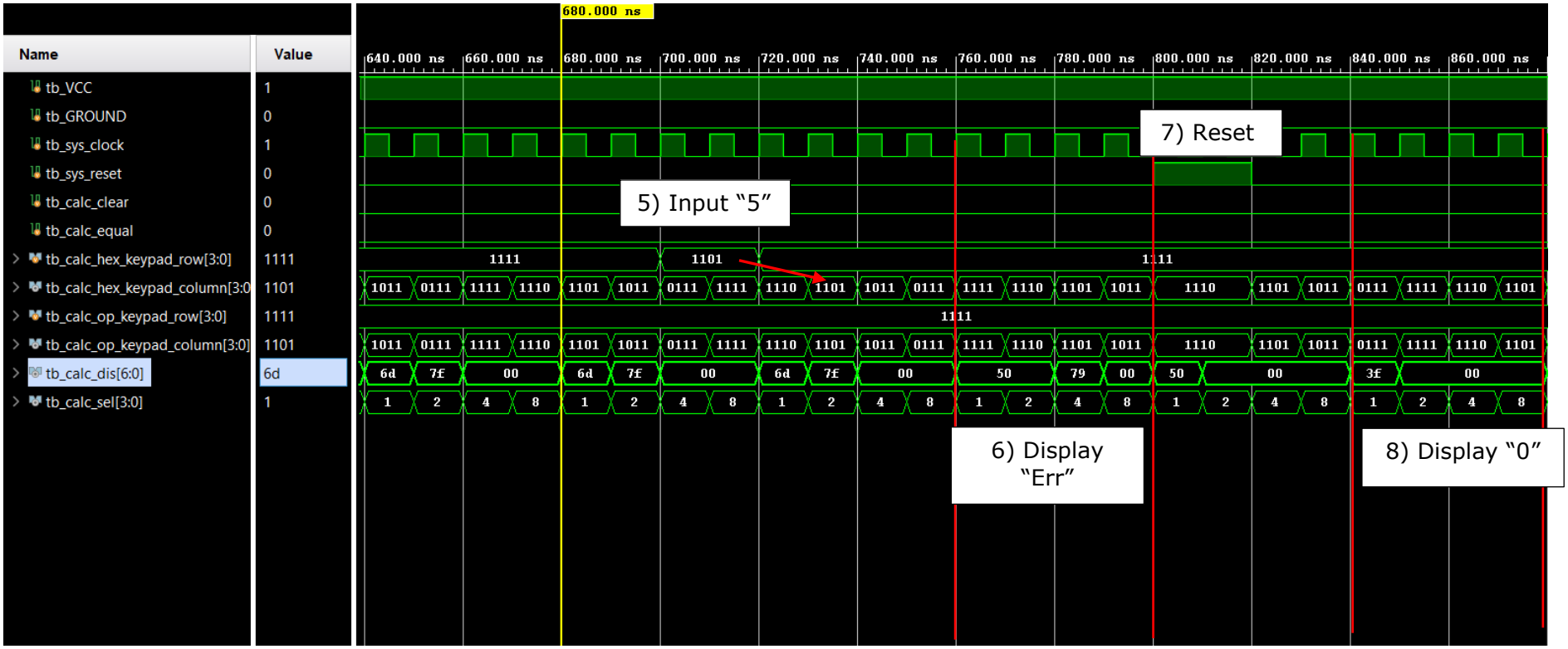

Figure 3.2.2.3.2 Test case 3b: Input overflow (input 555)

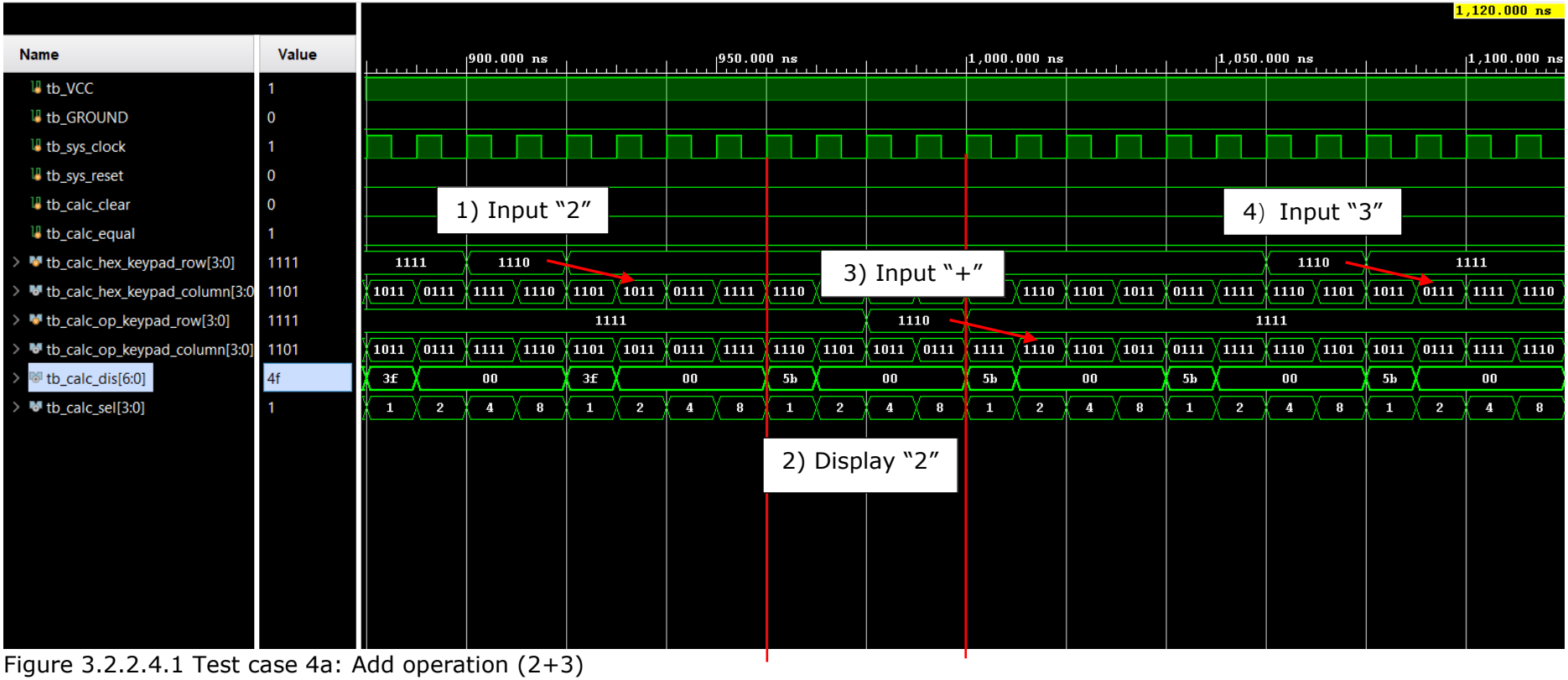

Figure 3.2.2.4.1 Test case 4a: Add operation (2+3)

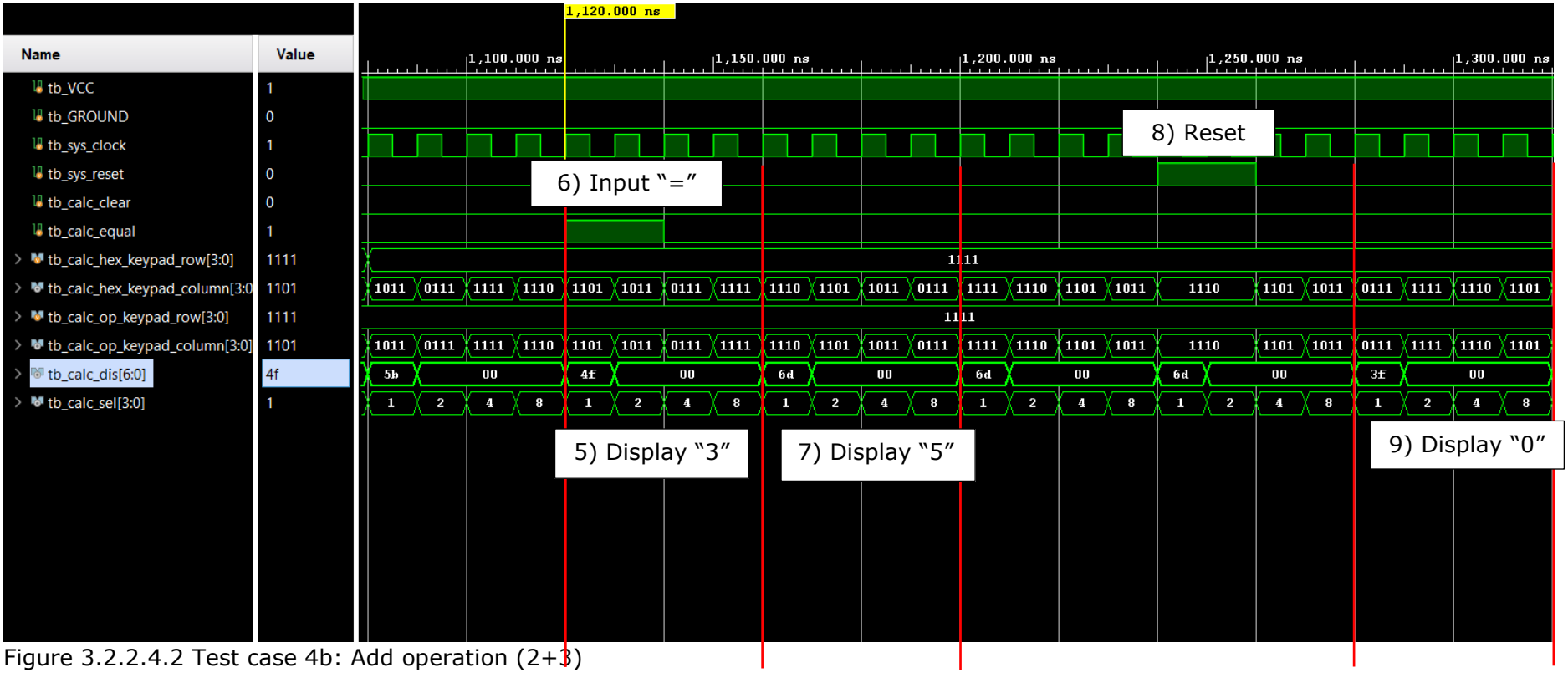

Figure 3.2.2.4.2 Test case 4b: Add operation (2+3)

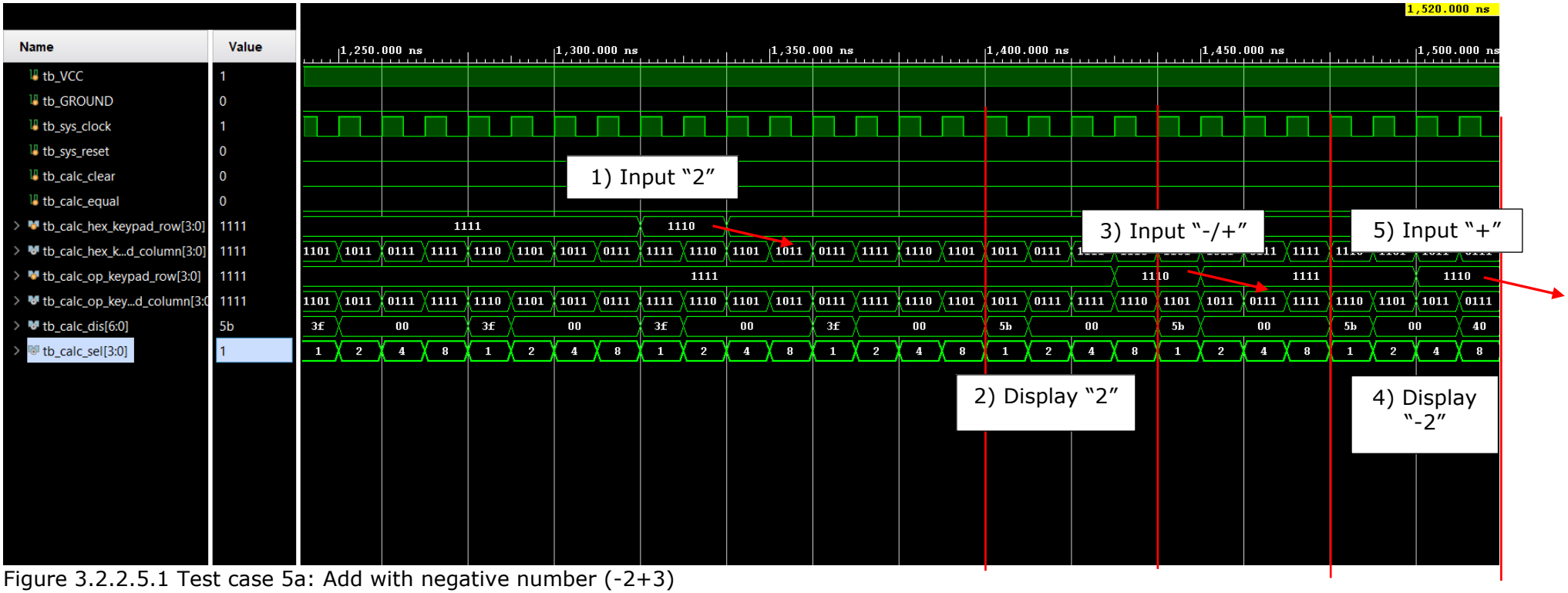

Figure 3.2.2.5.1 Test case 5a: Add with negative number (-2+3)

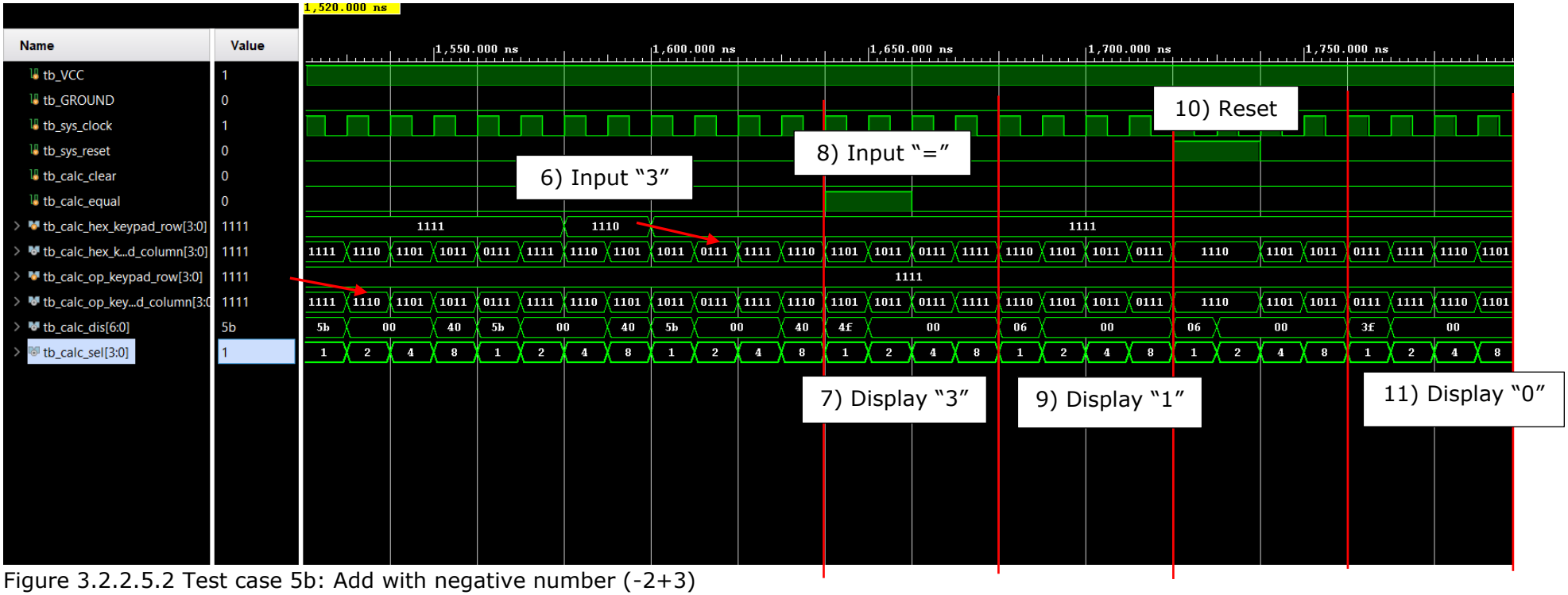

Figure 3.2.2.5.2 Test case 5b: Add with negative number (-2+3)

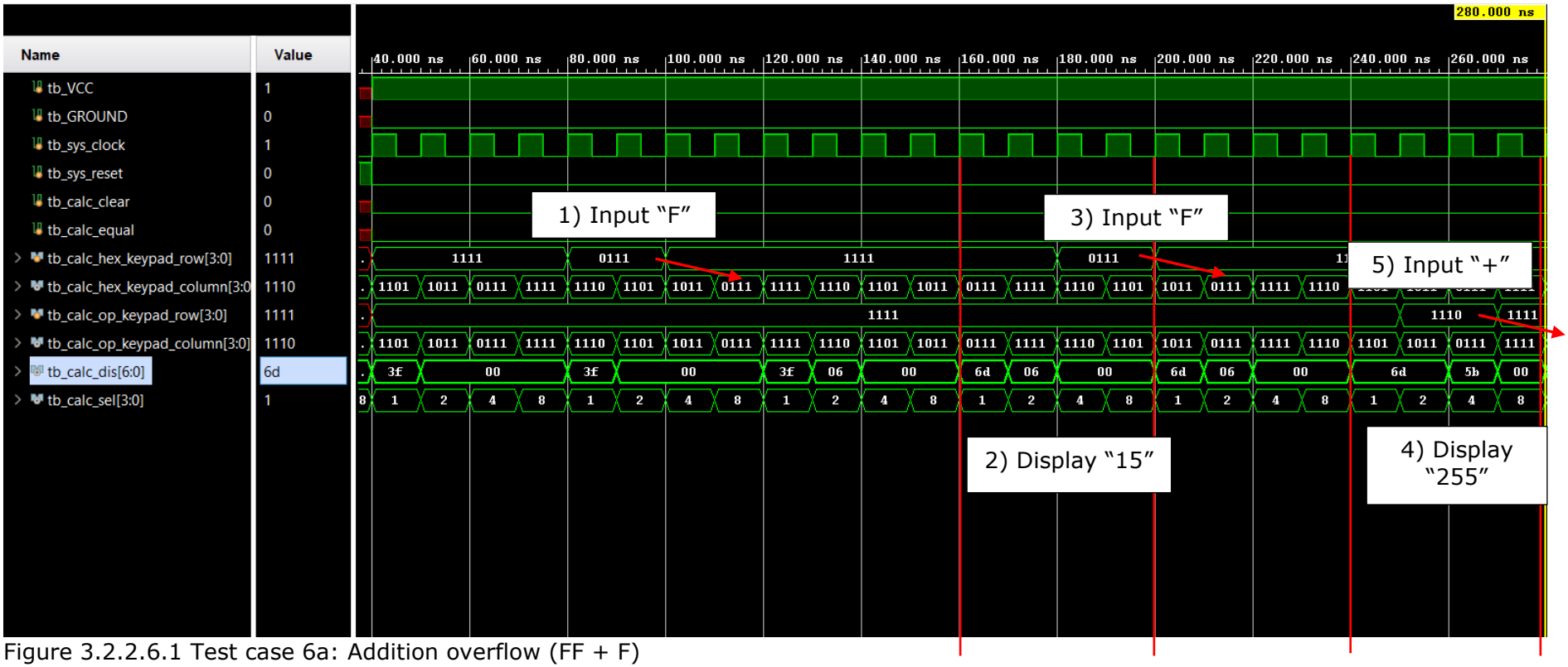

Figure 3.2.2.6.1 Test case 6a: Addition overflow (FF + F)

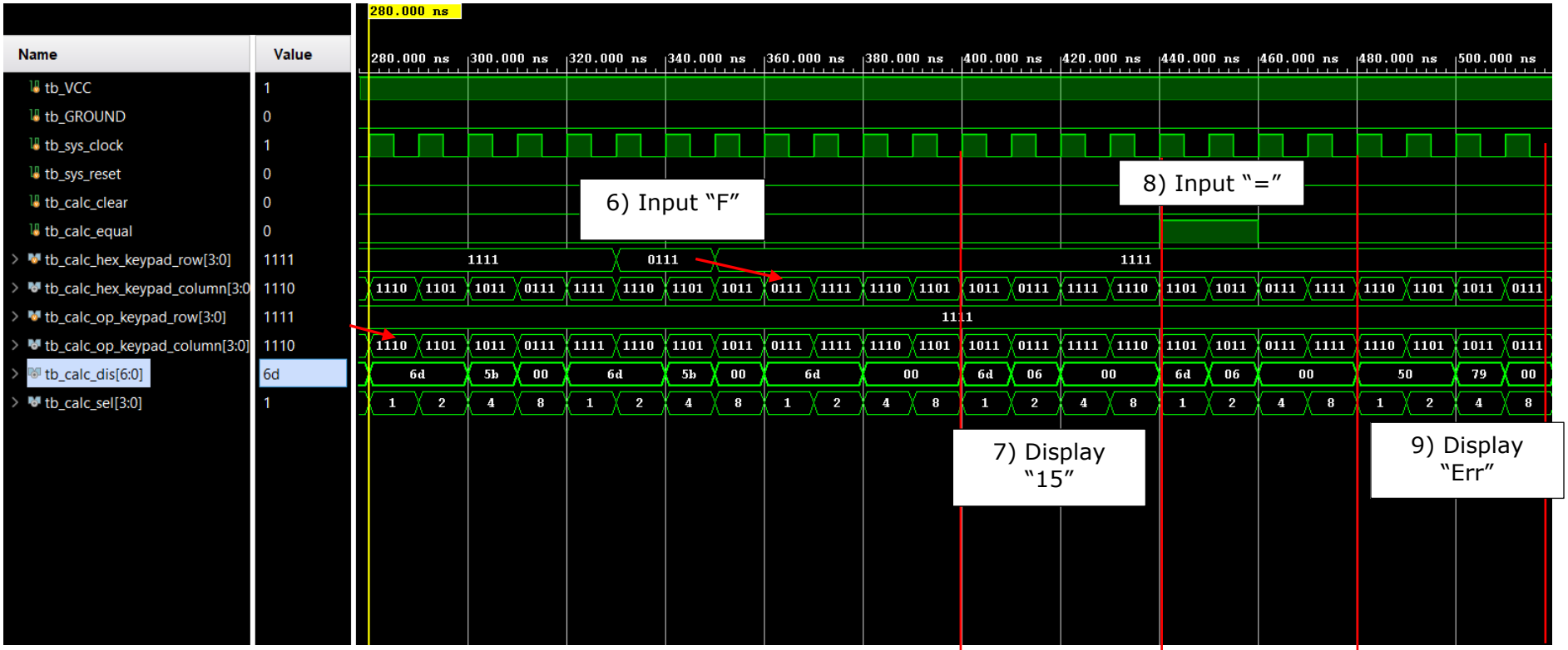

Figure 3.2.2.6.2 Test case 6b: Addition overflow (FF + F)

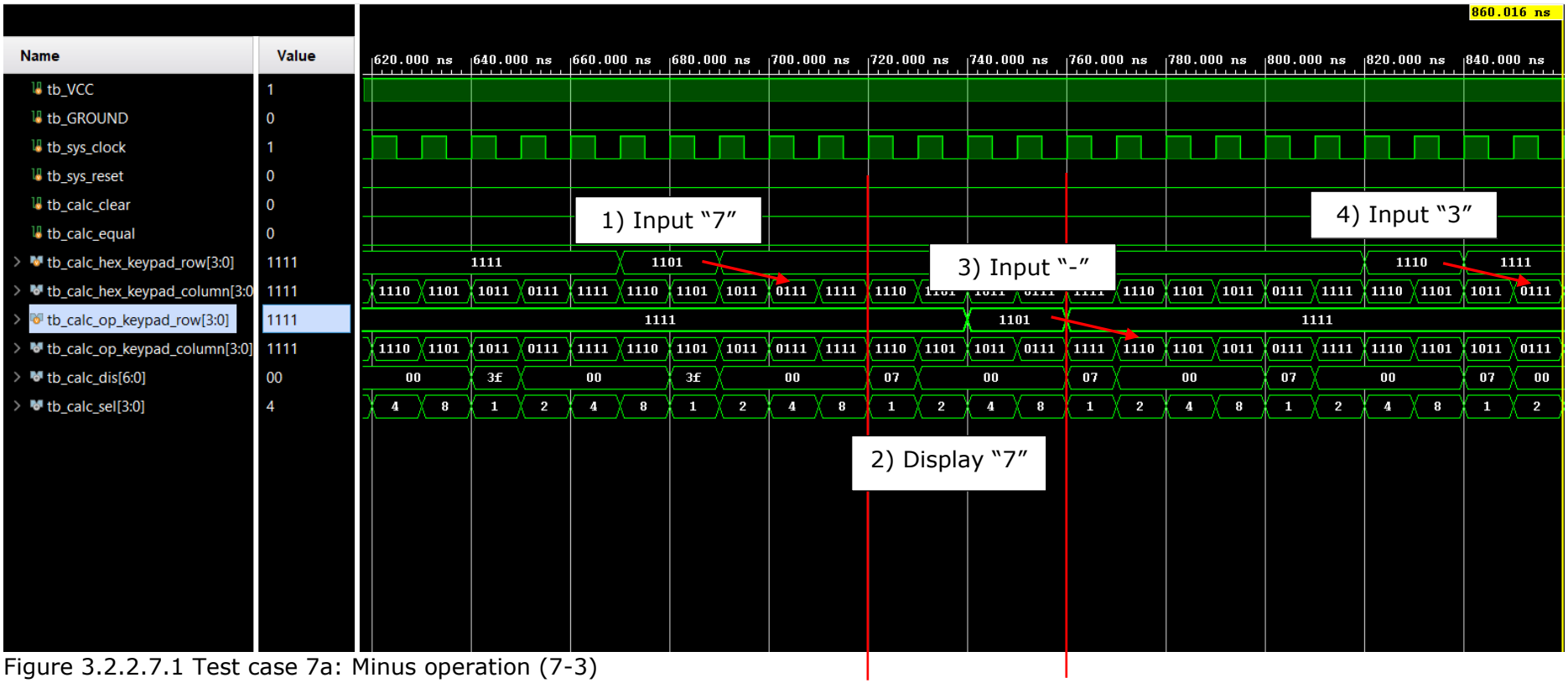

Figure 3.2.2.7.1 Test case 7a: Minus operation (7-3)

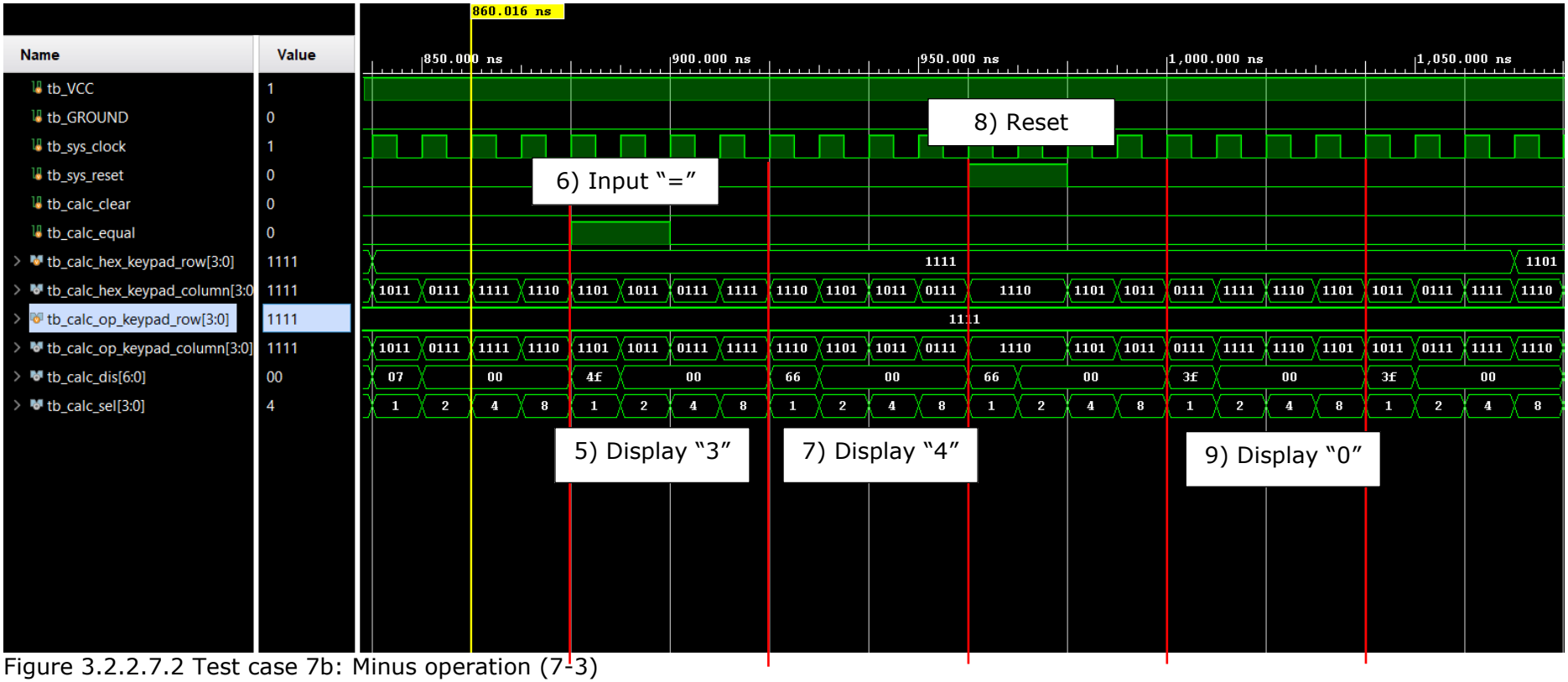

Figure 3.2.2.7.2 Test case 7b: Minus operation (7-3)

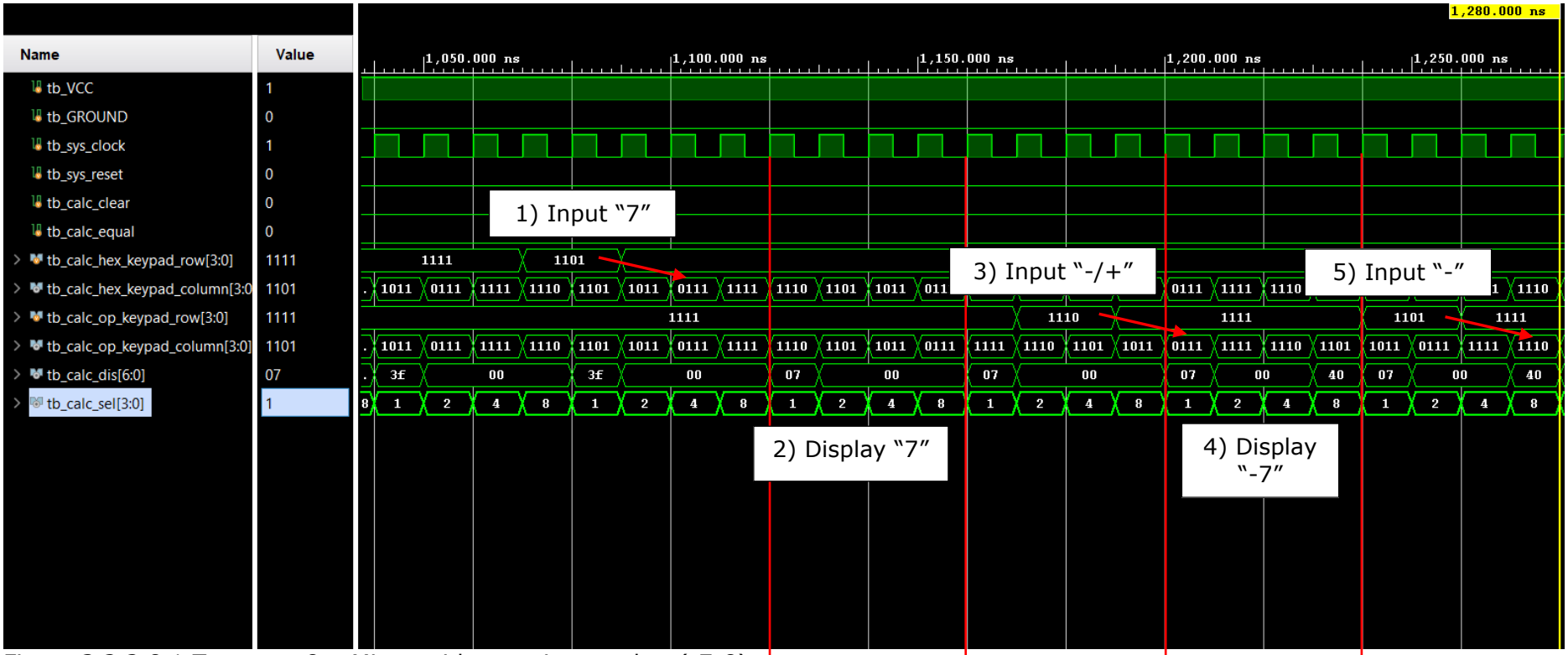

Figure 3.2.2.8.1 Test case 8a: Minus with negative number (-7-8)

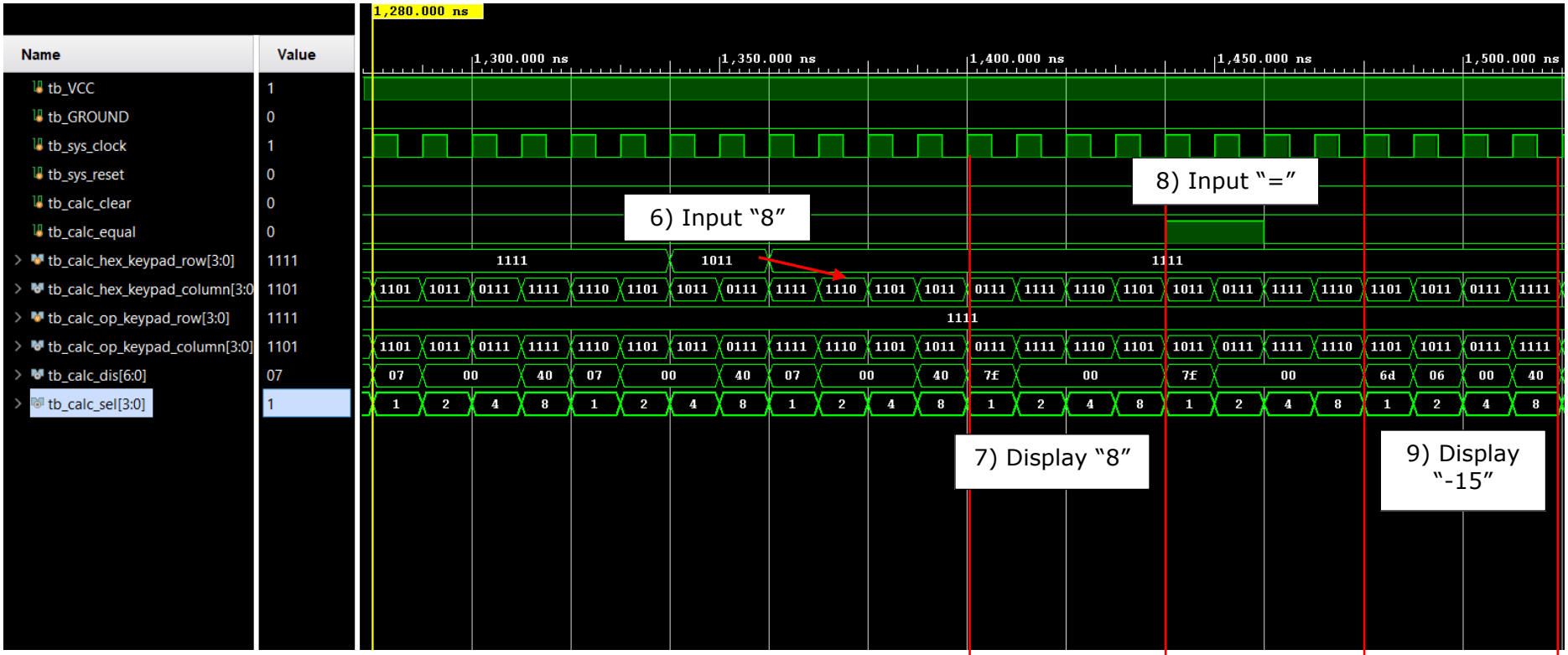

Figure 3.2.2.8.2 Test case 8b: Minus with negative number (-7-8)

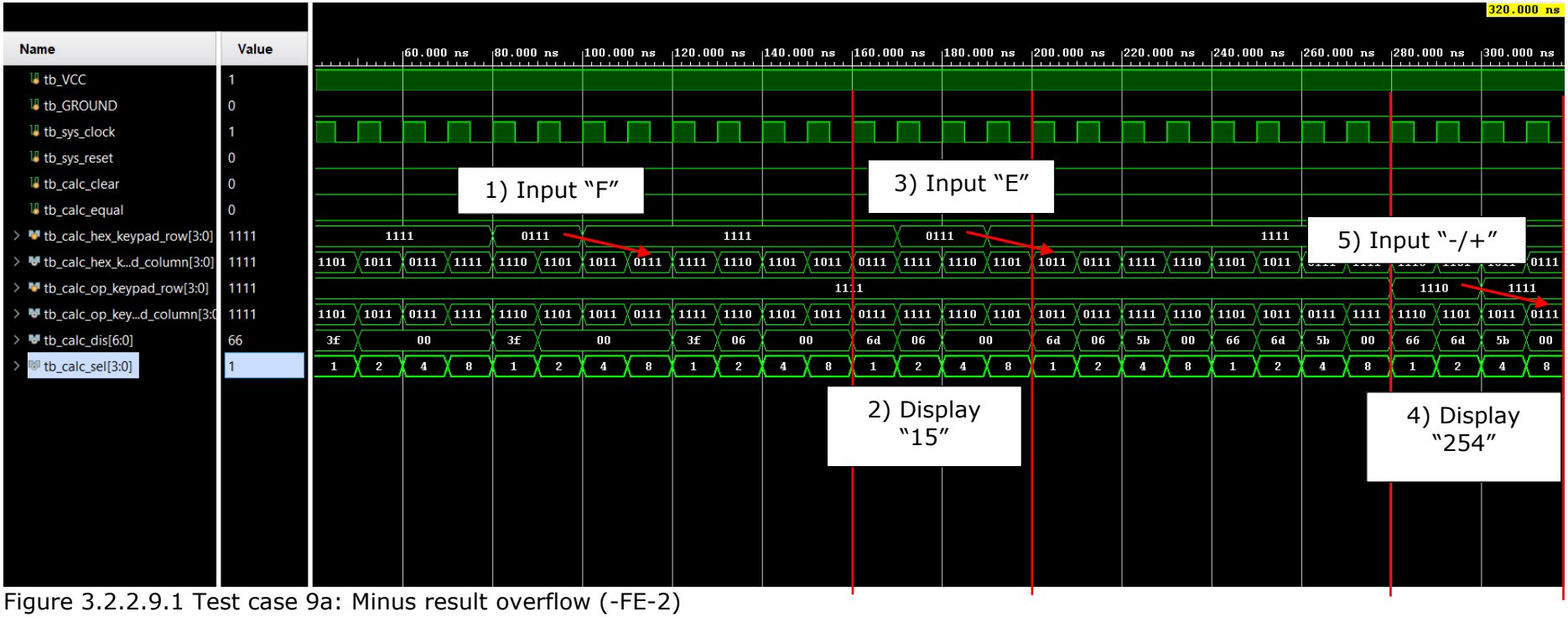

Figure 3.2.2.9.1 Test case 9a: Minus result overflow (-FE-2)

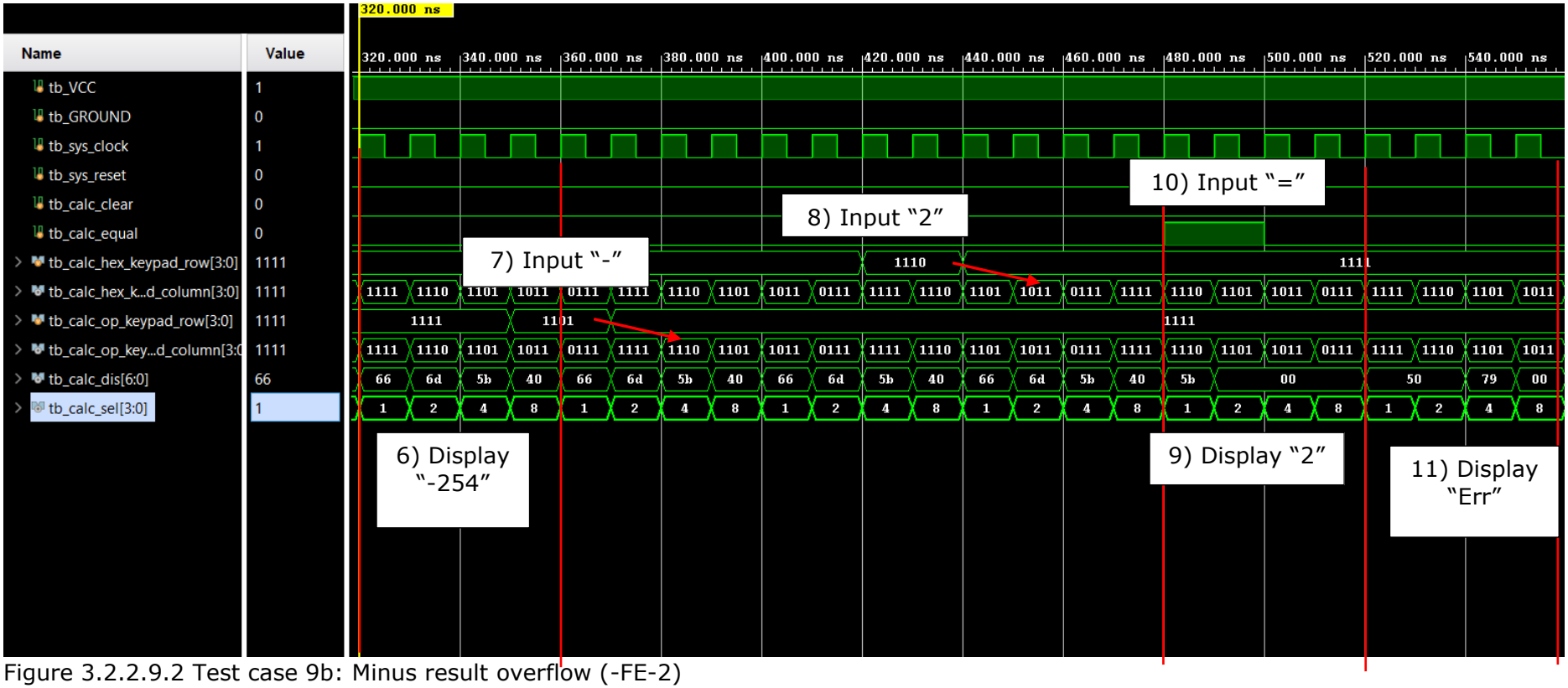

Figure 3.2.2.9.2 Test case 9b: Minus result overflow (-FE-2)

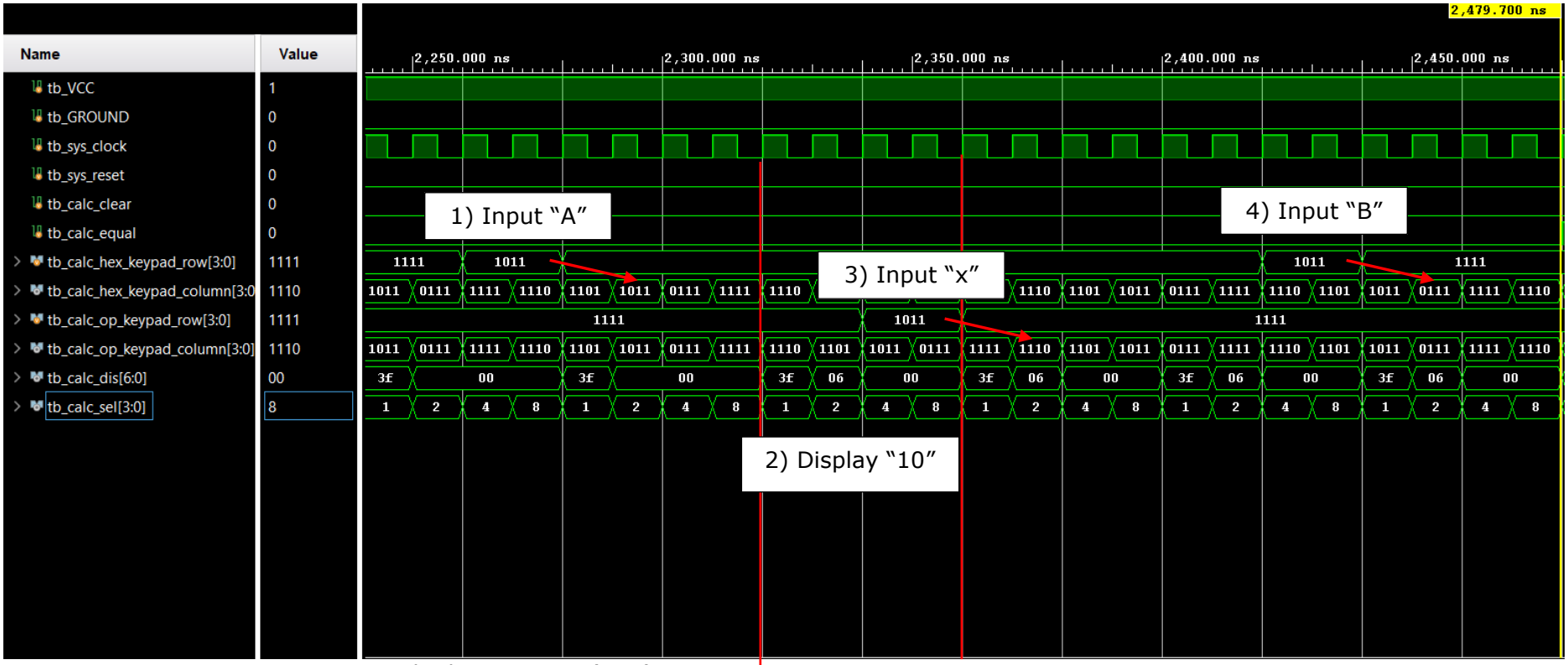

Figure 3.2.2.10.1 Test case 10a: Multiply operation (AxB)

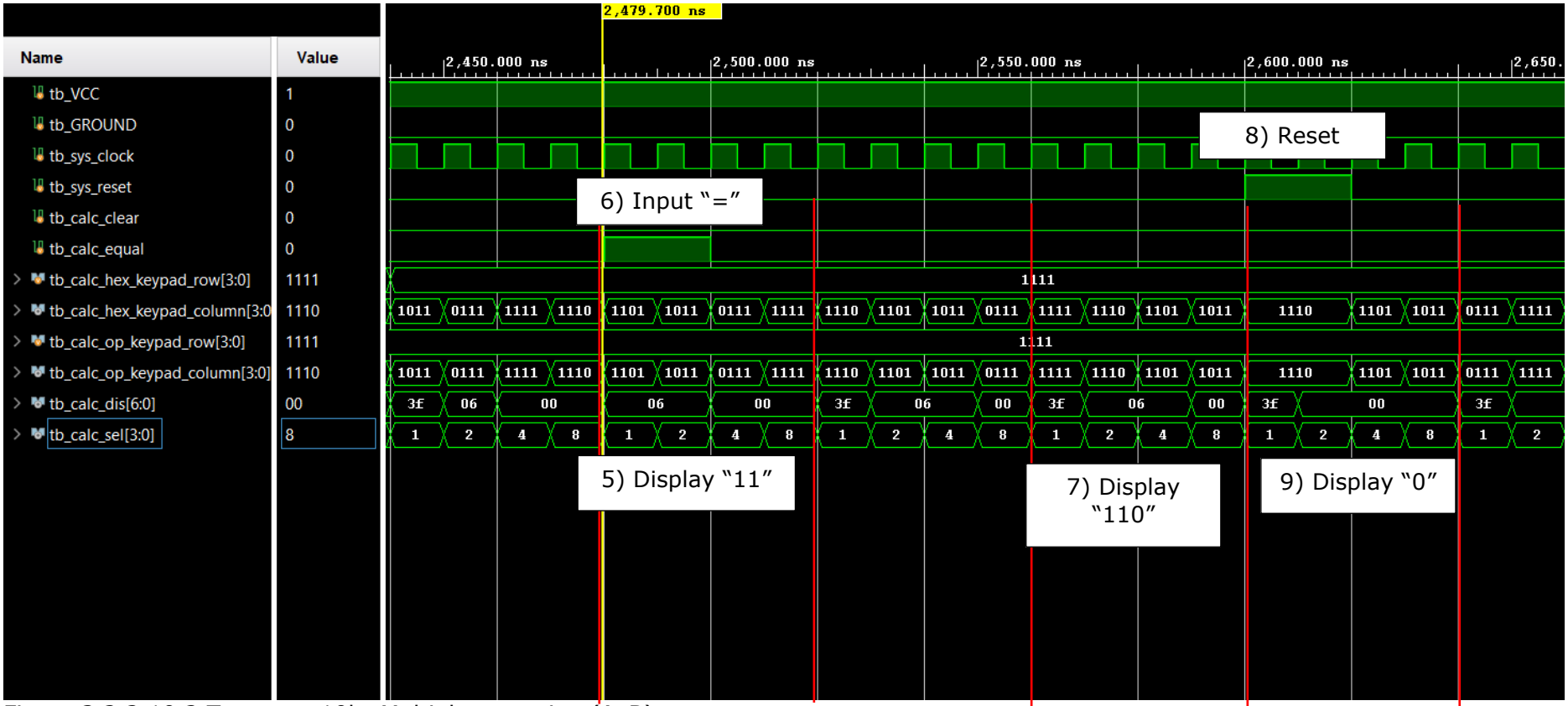

Figure 3.2.2.10.2 Test case 10b: Multiply operation (AxB)

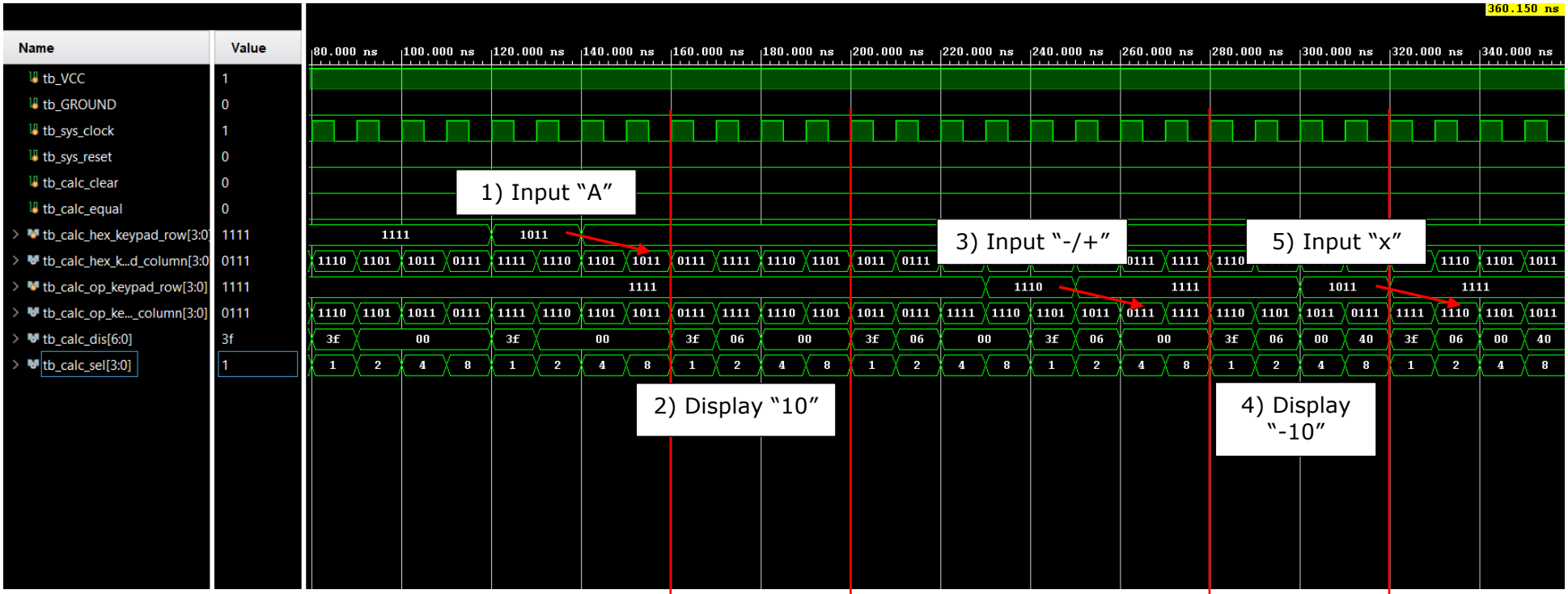

Figure 3.2.2.11.1 Test case 11a: Multiply with negative number (-AxB)

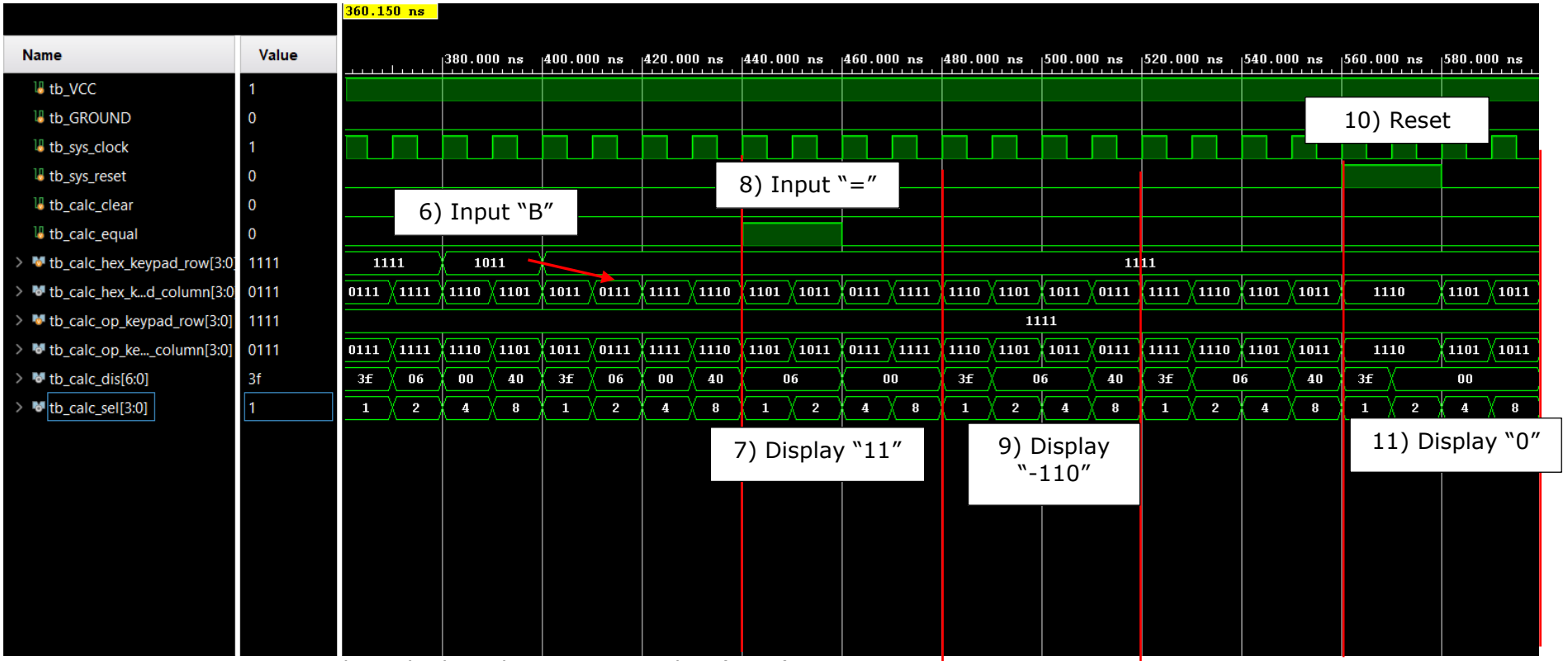

Figure 3.2.2.11.2 Test case 11b: Multiply with negative number (-AxB)

Figure 3.2.2.12.1 Test case 12a: Multiplication overflow (20xE)

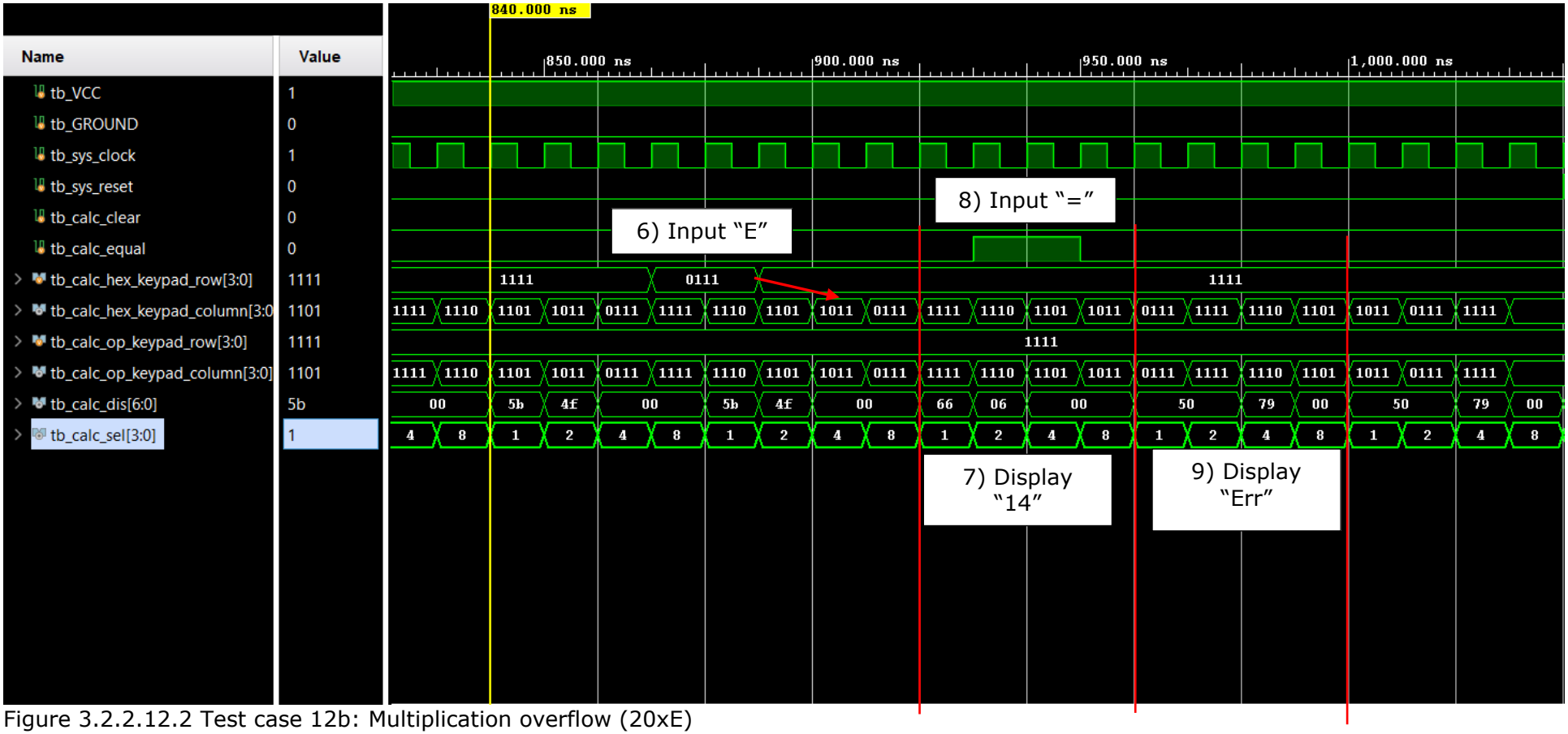

Figure 3.2.2.12.2 Test case 12b: Multiplication overflow (20xE)

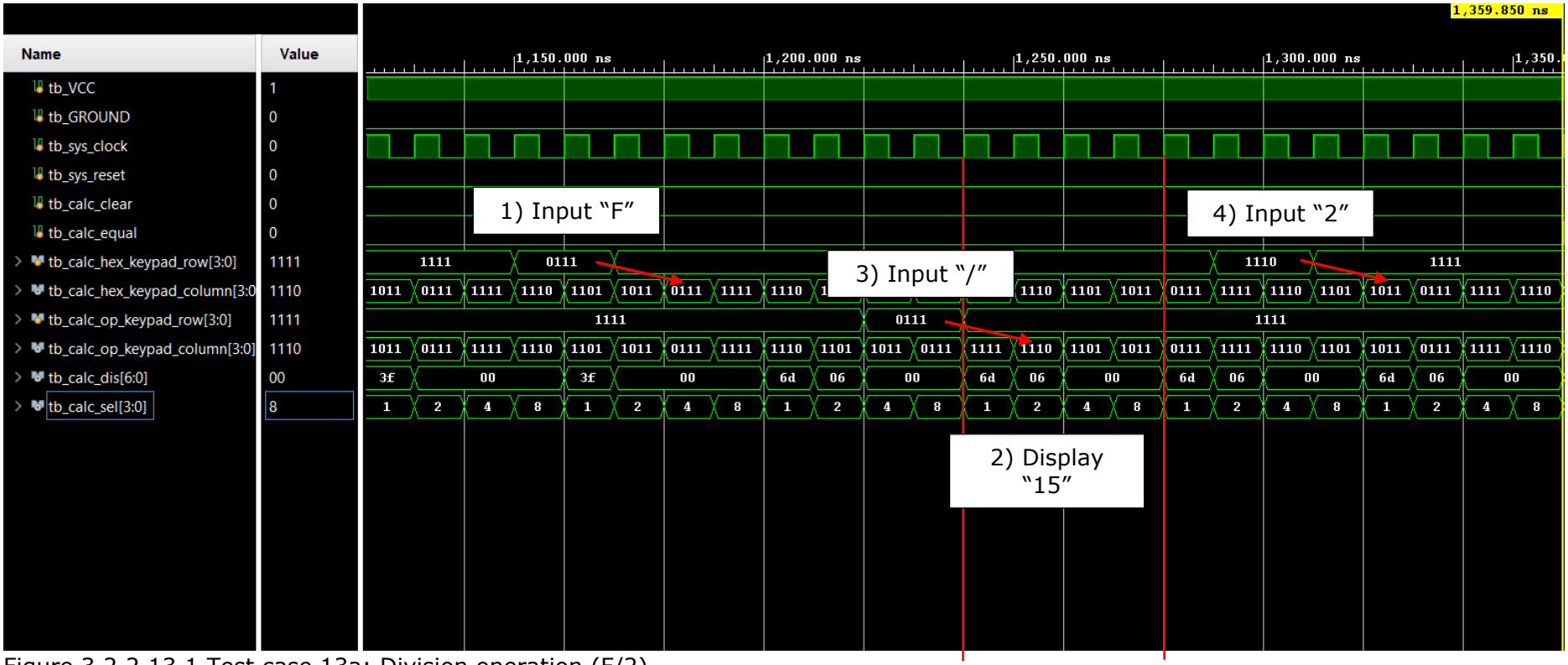

Figure 3.2.2.13.1 Test case 13a: Division operation (F/2)

Figure 3.2.2.13.2 Test case 13b: Division operation (F/2)

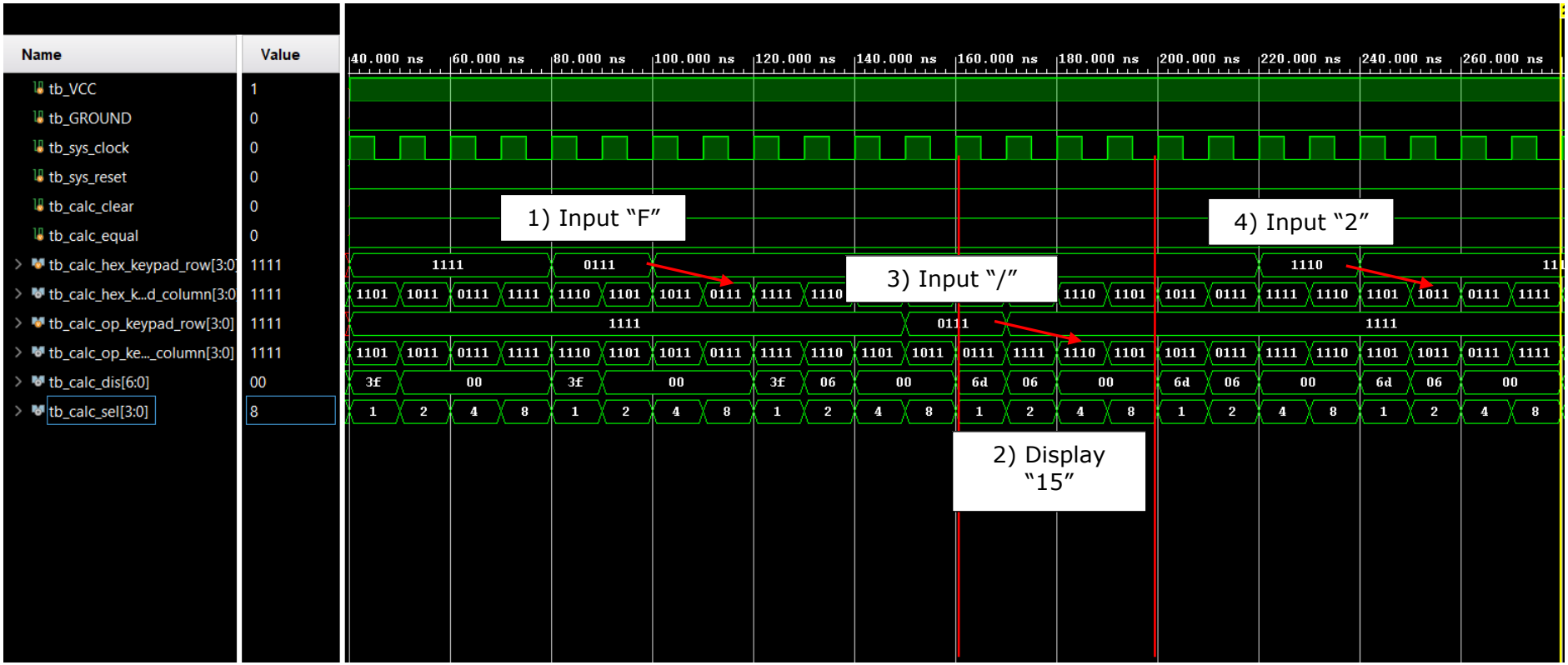

Figure 3.2.2.14.1 Test case 14a: Division with negative number (F/-2)

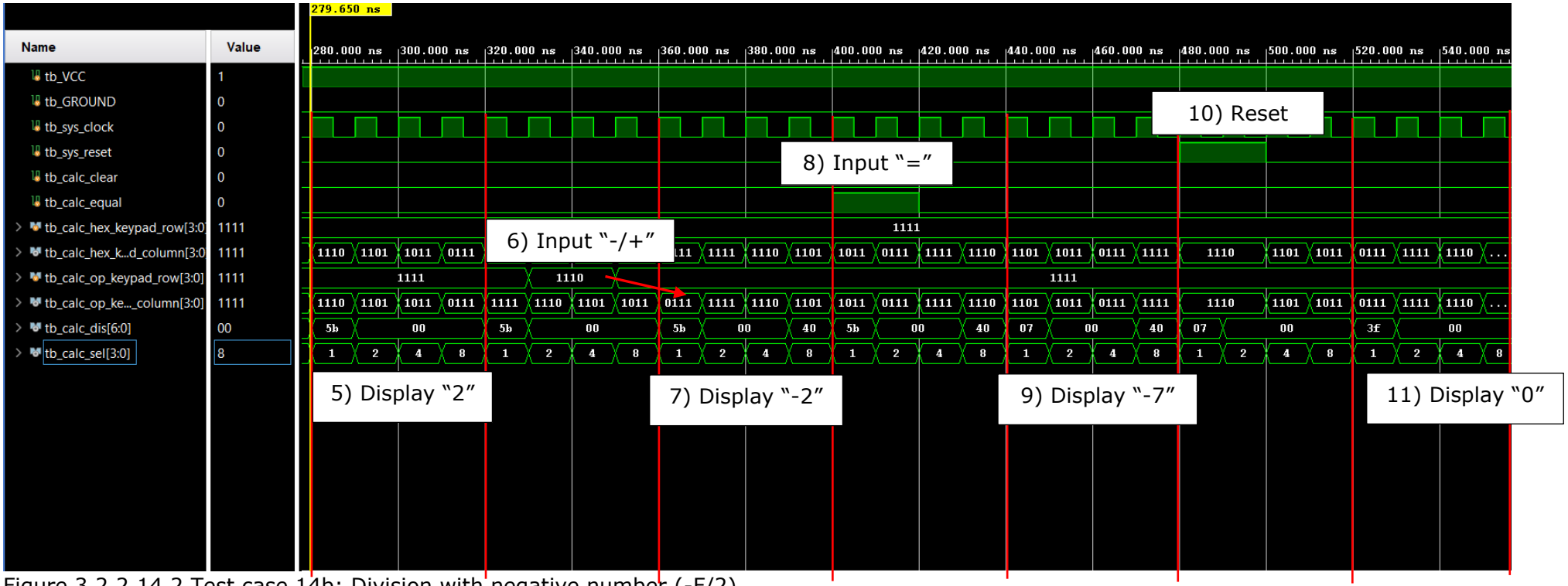

Figure 3.2.2.14.2 Test case 14b: Division with negative number (-F/2)

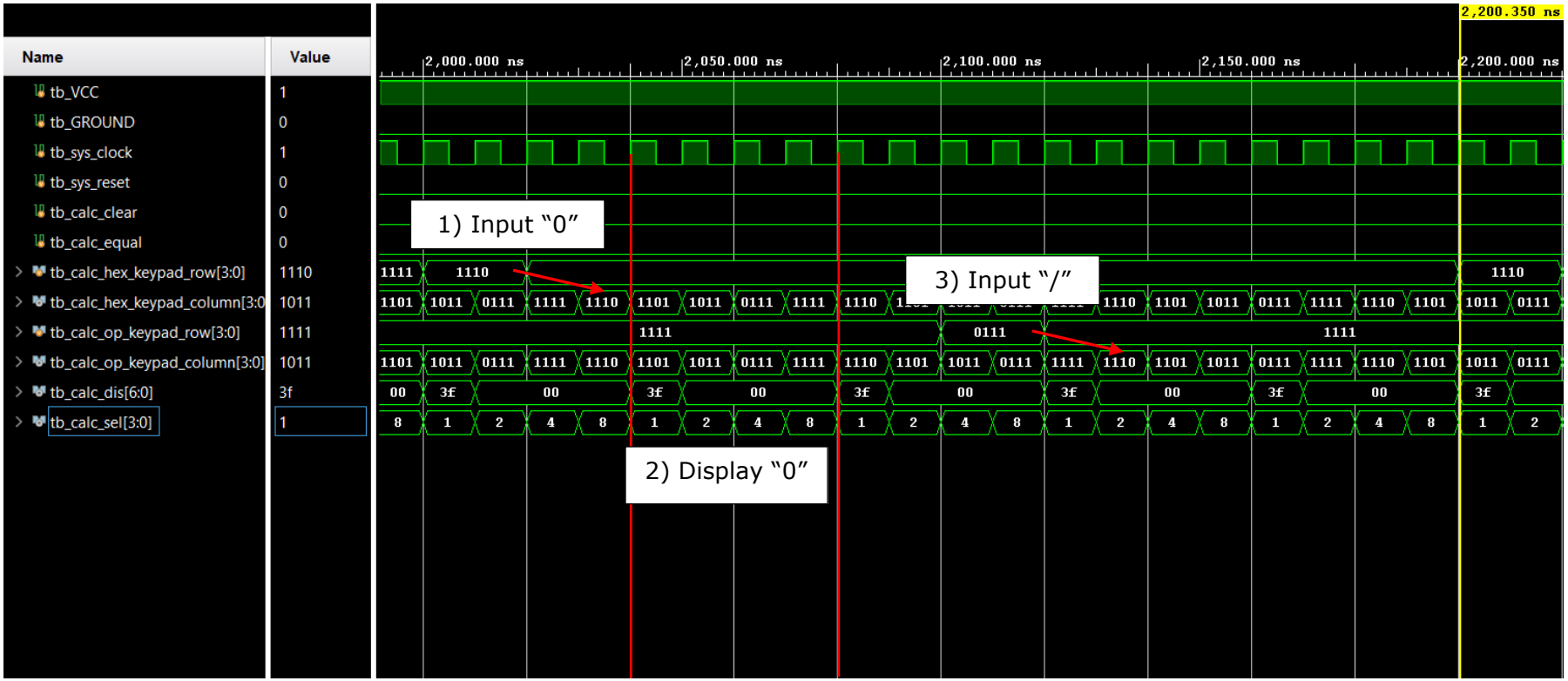

Figure 3.2.2.15.1 Test case 15a: Divide by zero (0/0)

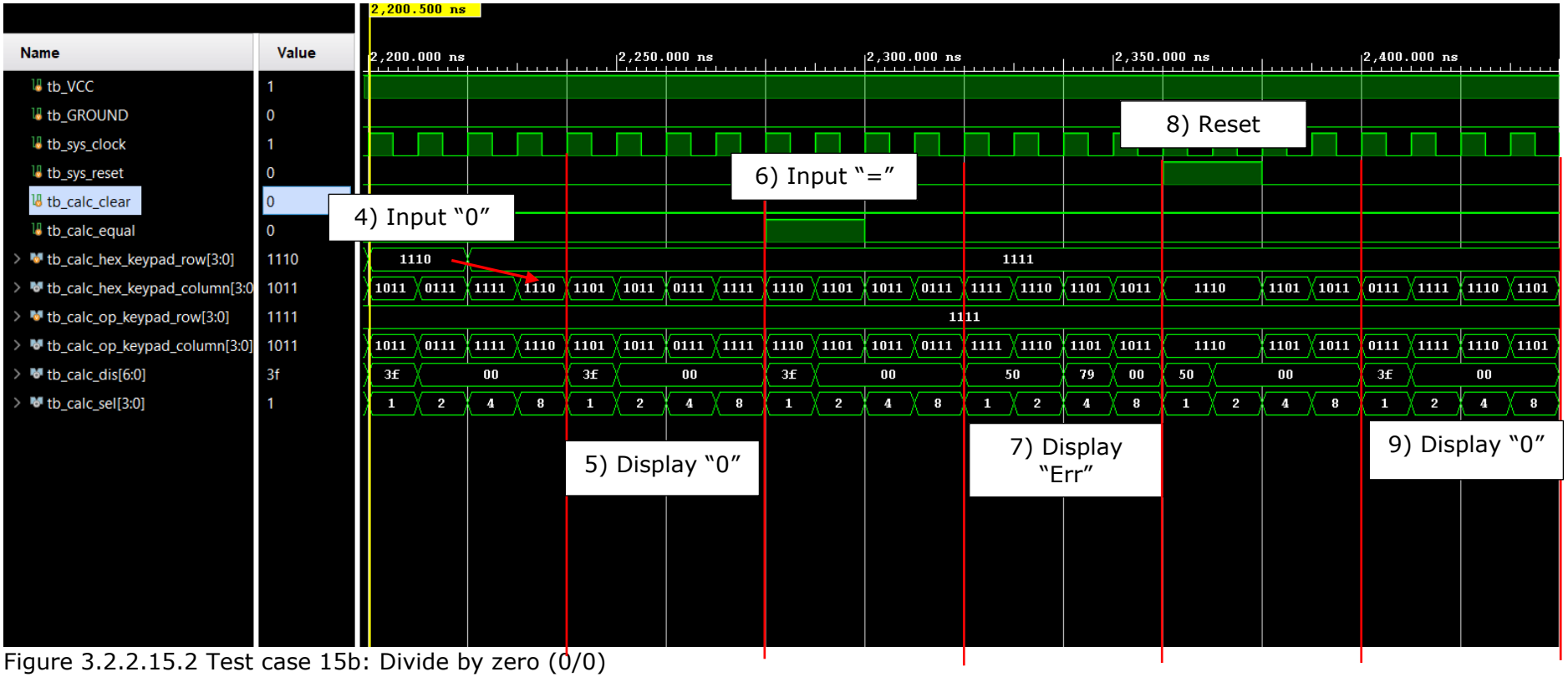

Figure 3.2.2.15.2 Test case 15b: Divide by zero (0/0)

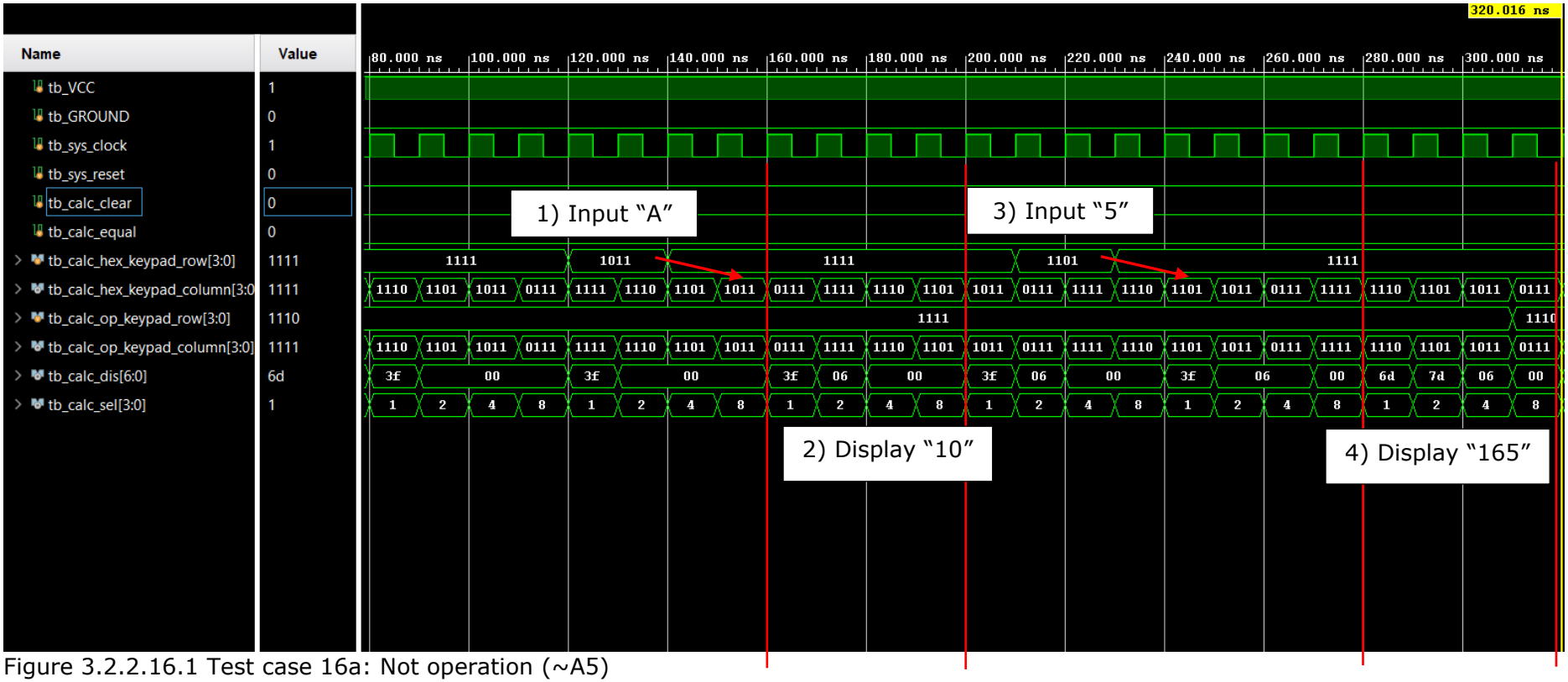

Figure 3.2.2.16.1 Test case 16a: Not operation (~A5)

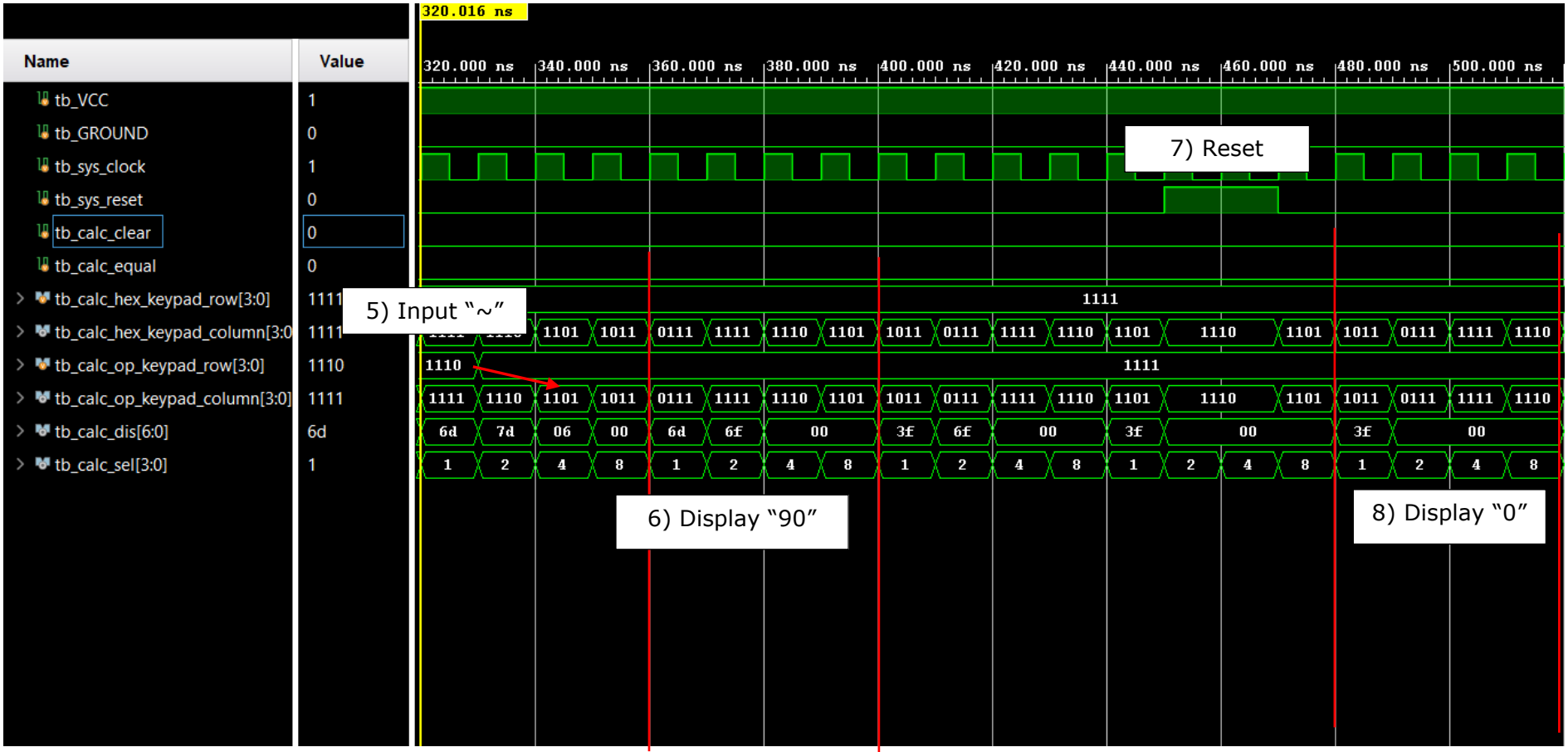

Figure 3.2.2.16.2 Test case 16b: Not operation (~A5)

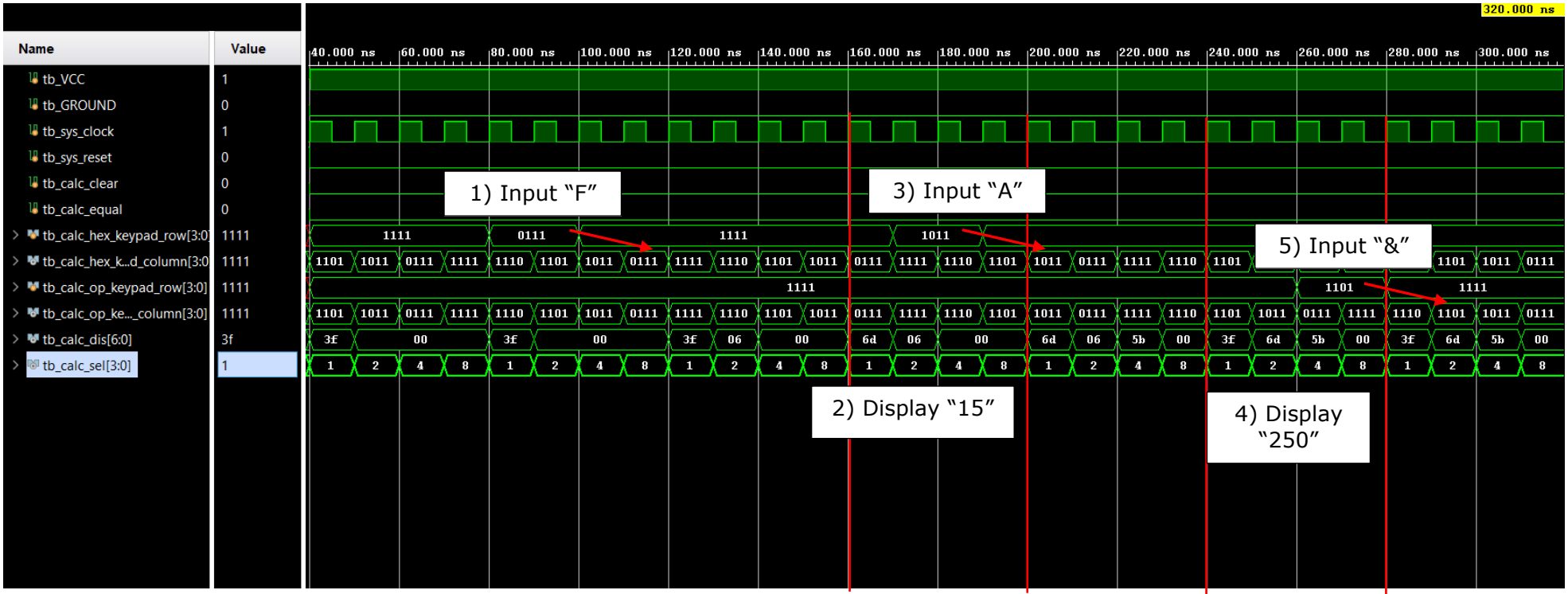

Figure 3.2.2.17.1 Test case 17a: AND operation (FA & A0)

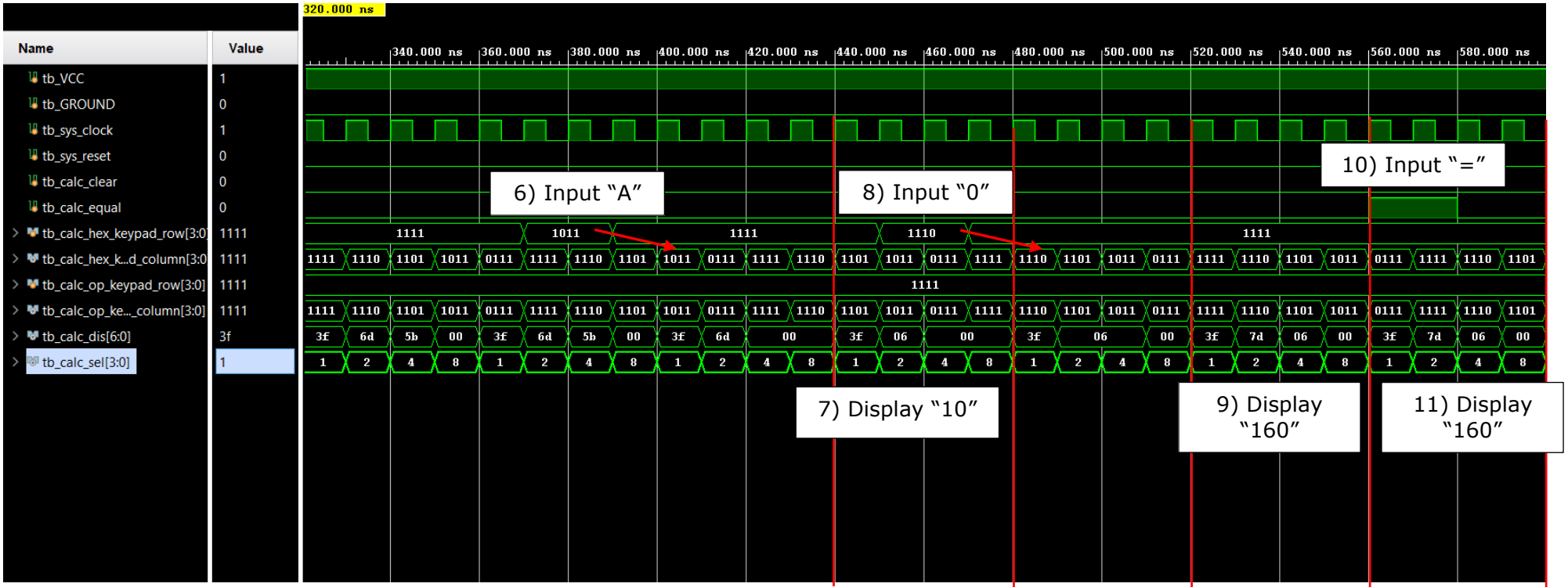

Figure 3.2.2.17.2 Test case 17b: AND operation (FA & A0)

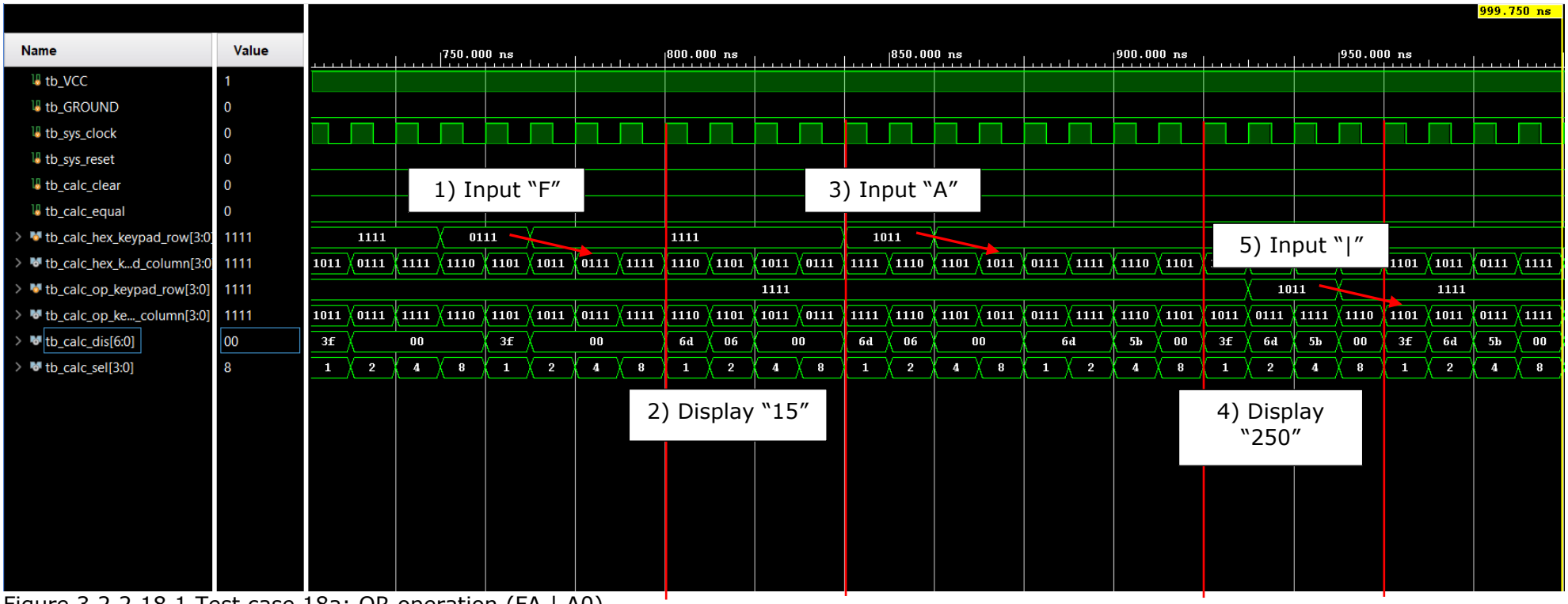

Figure 3.2.2.18.1 Test case 18a: OR operation (FA | A0)

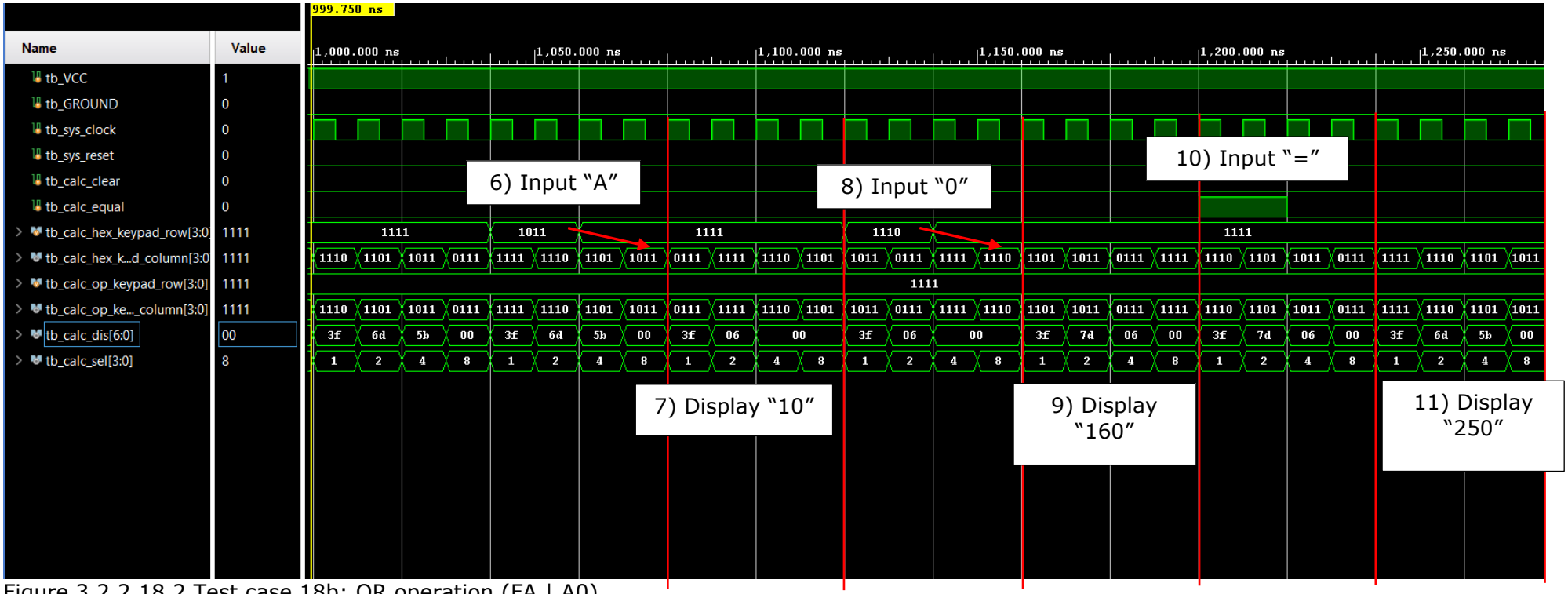

Figure 3.2.2.18.2 Test case 18b: OR operation (FA | A0)

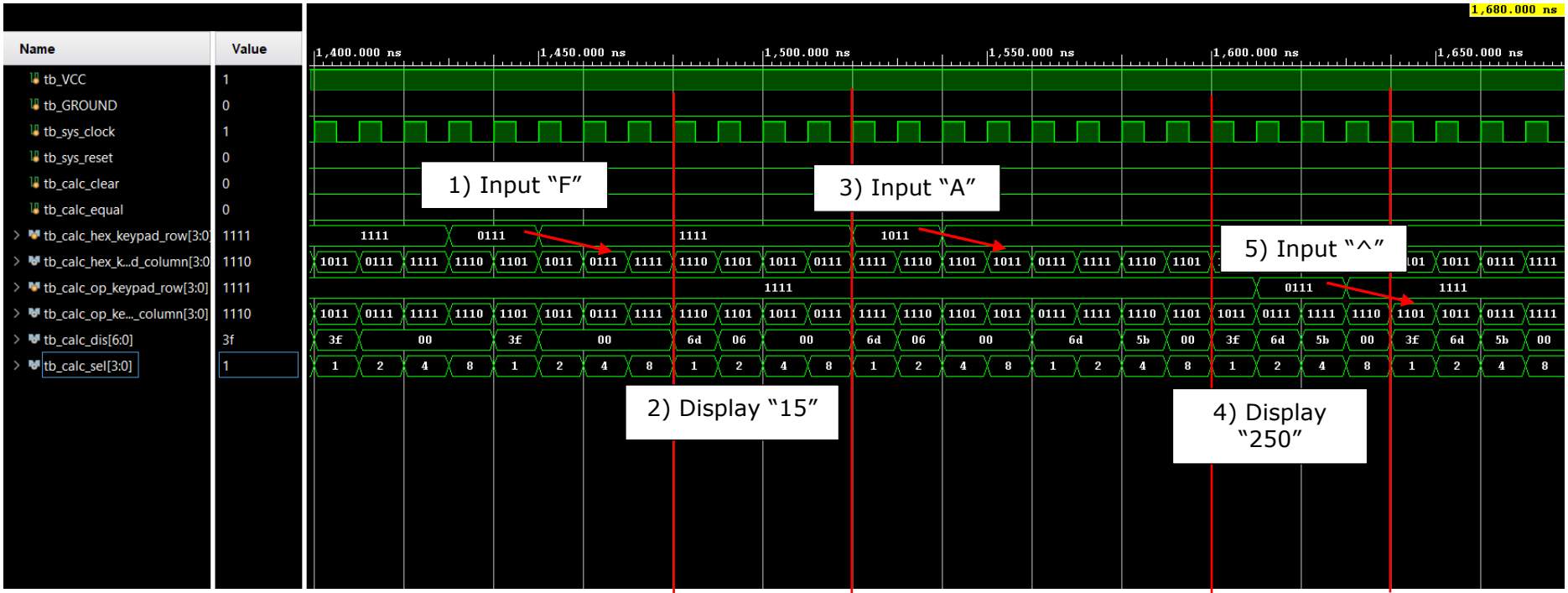

Figure 3.2.2.19.1 Test case 19a: XOR operation (FA ^ A0)

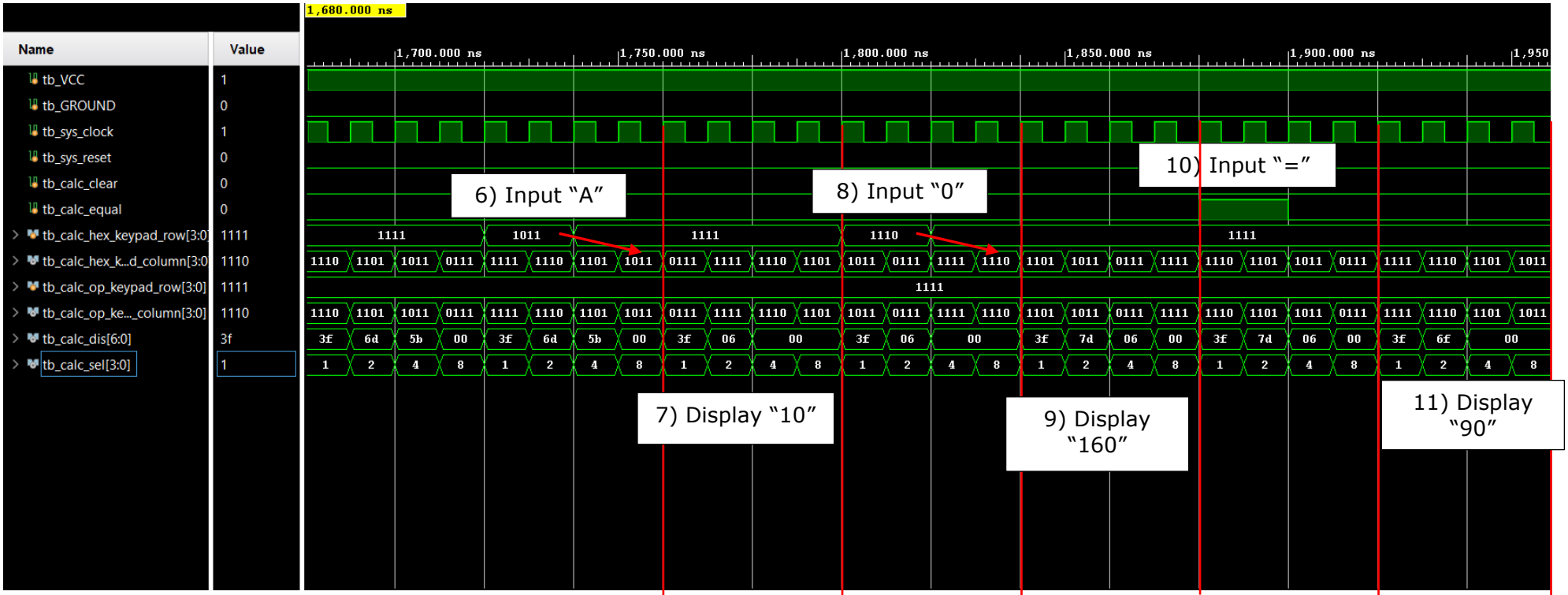

Figure 3.2.2.19.2 Test case 19b: XOR operation (FA ^ A0)

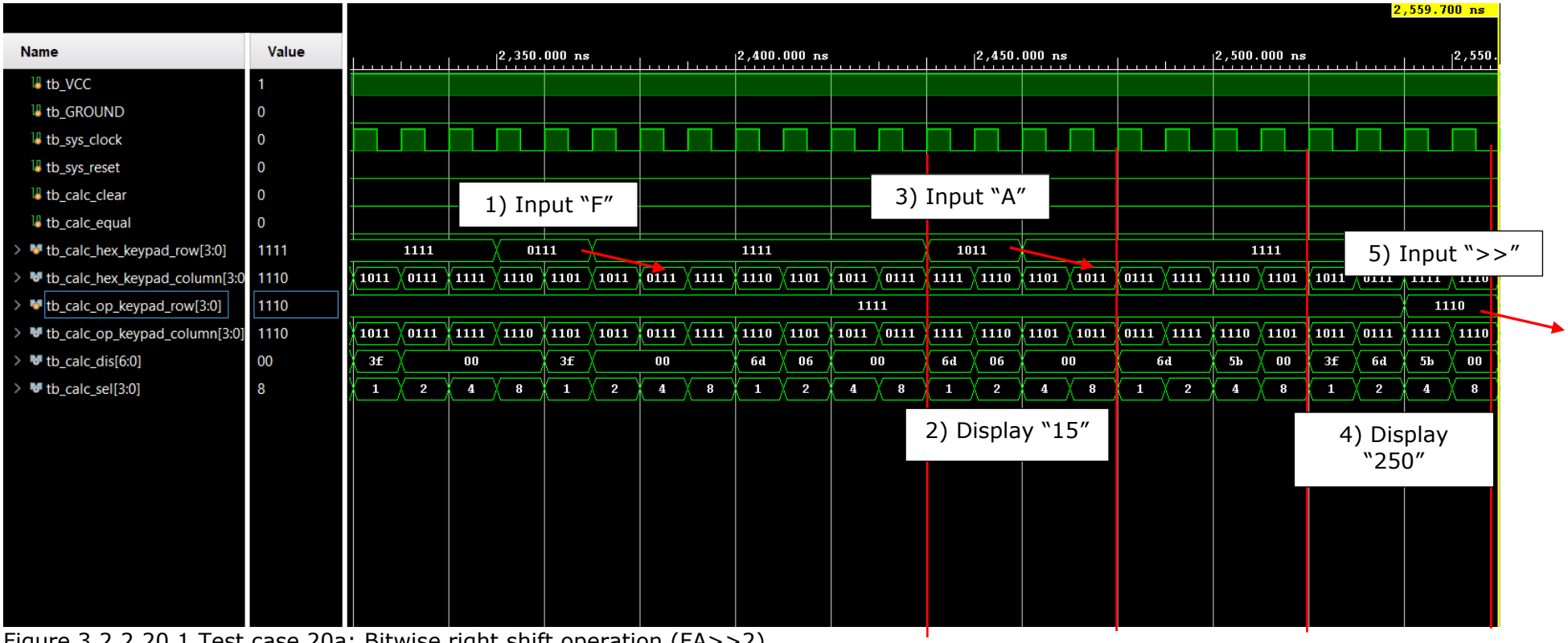

Figure 3.2.2.20.1 Test case 20a: Bitwise right shift operation (FA>>2)

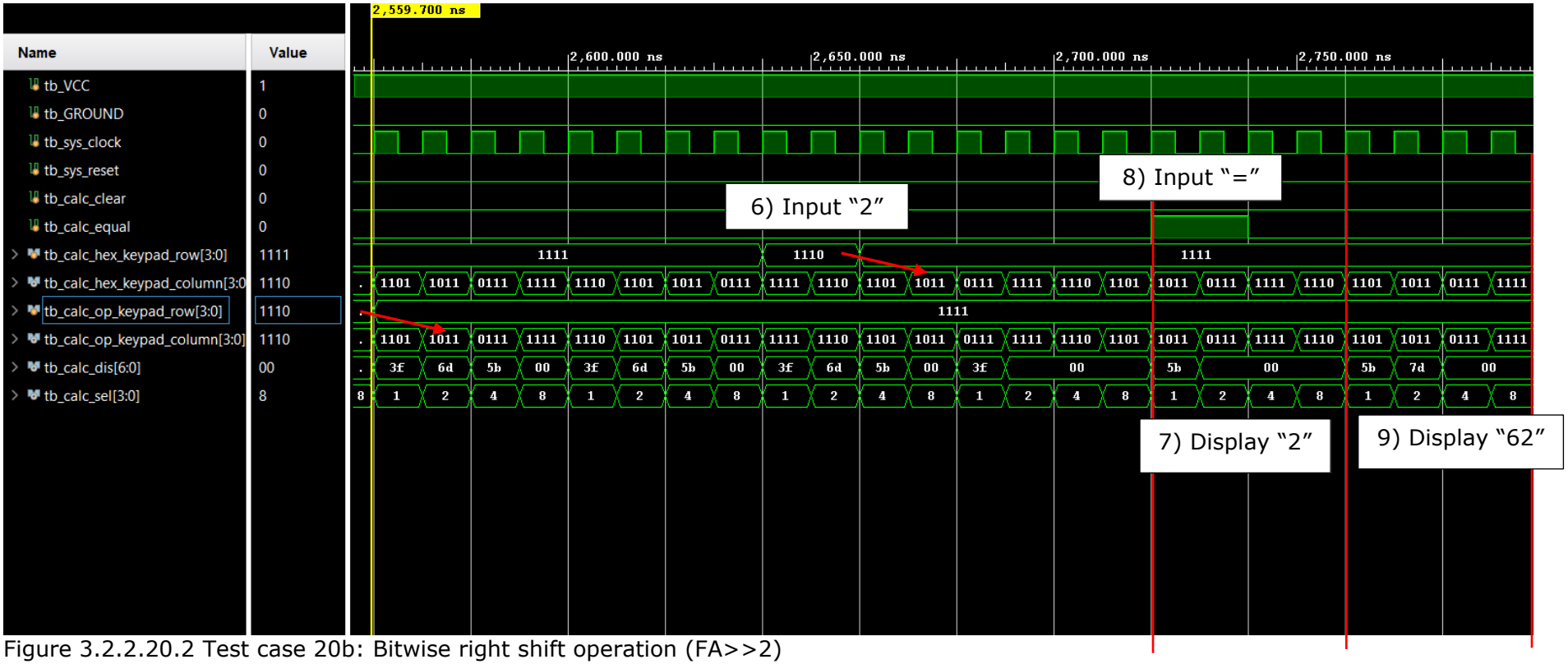

Figure 3.2.2.20.2 Test case 20b: Bitwise right shift operation (FA>>2)

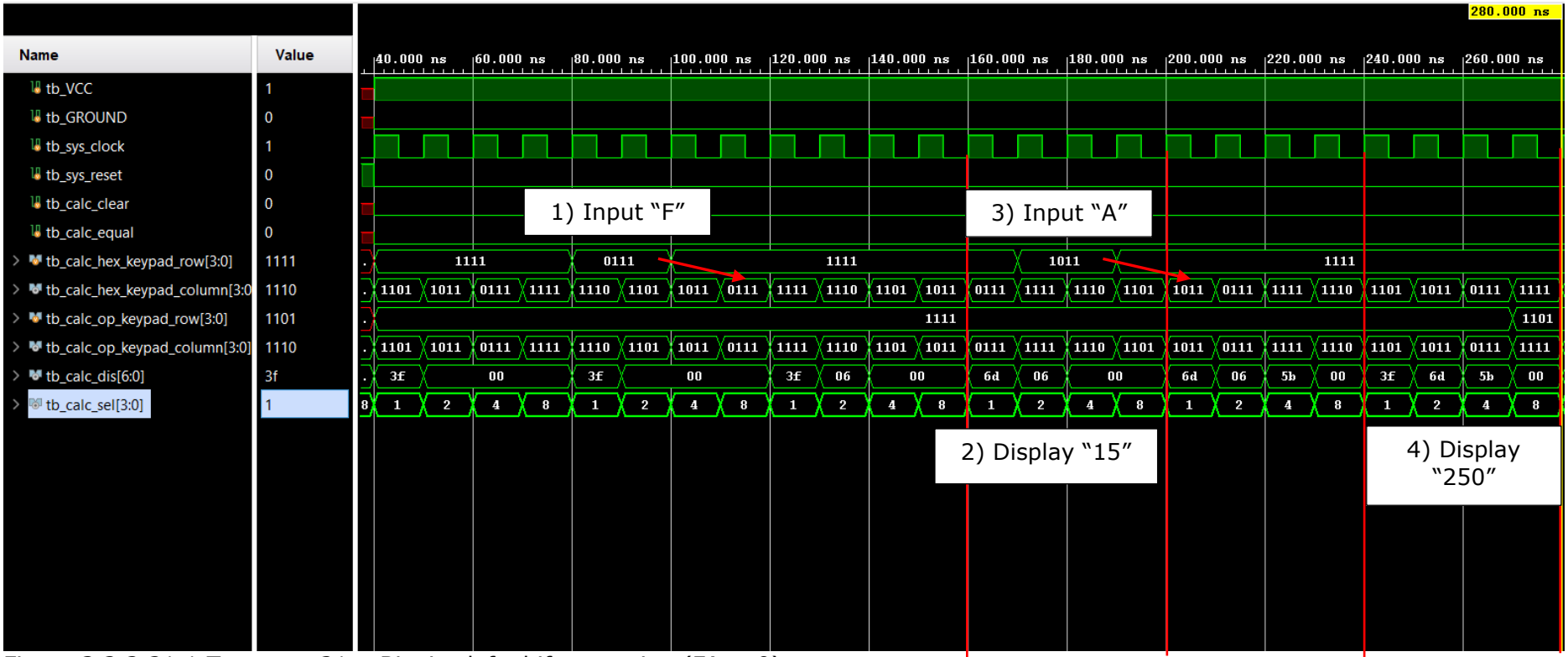

Figure 3.2.2.21.1 Test case 21a: Bitwise left shift operation (FA<<9)

Figure 3.2.2.21.2 Test case 21b: Bitwise left shift operation (FA<<9)

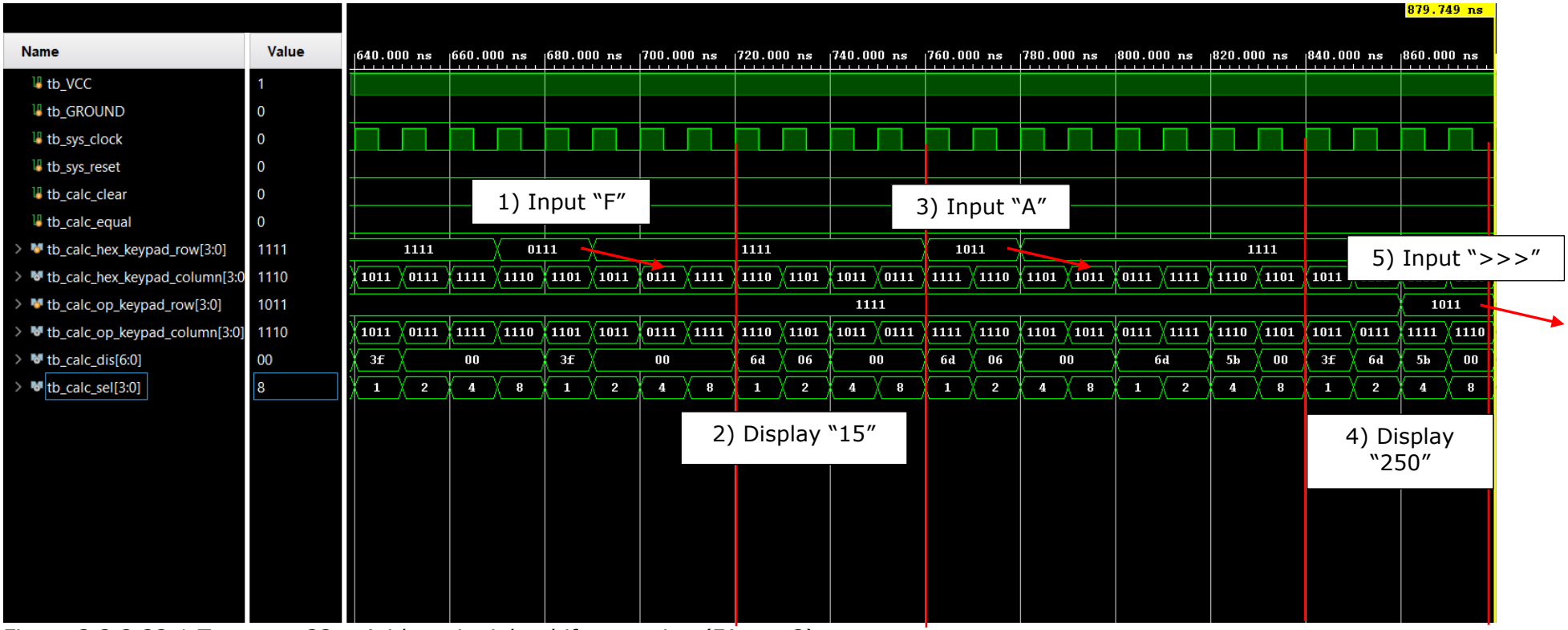

Figure 3.2.2.22.1 Test case 22a: Arithmetic right shift operation (FA>>>2)

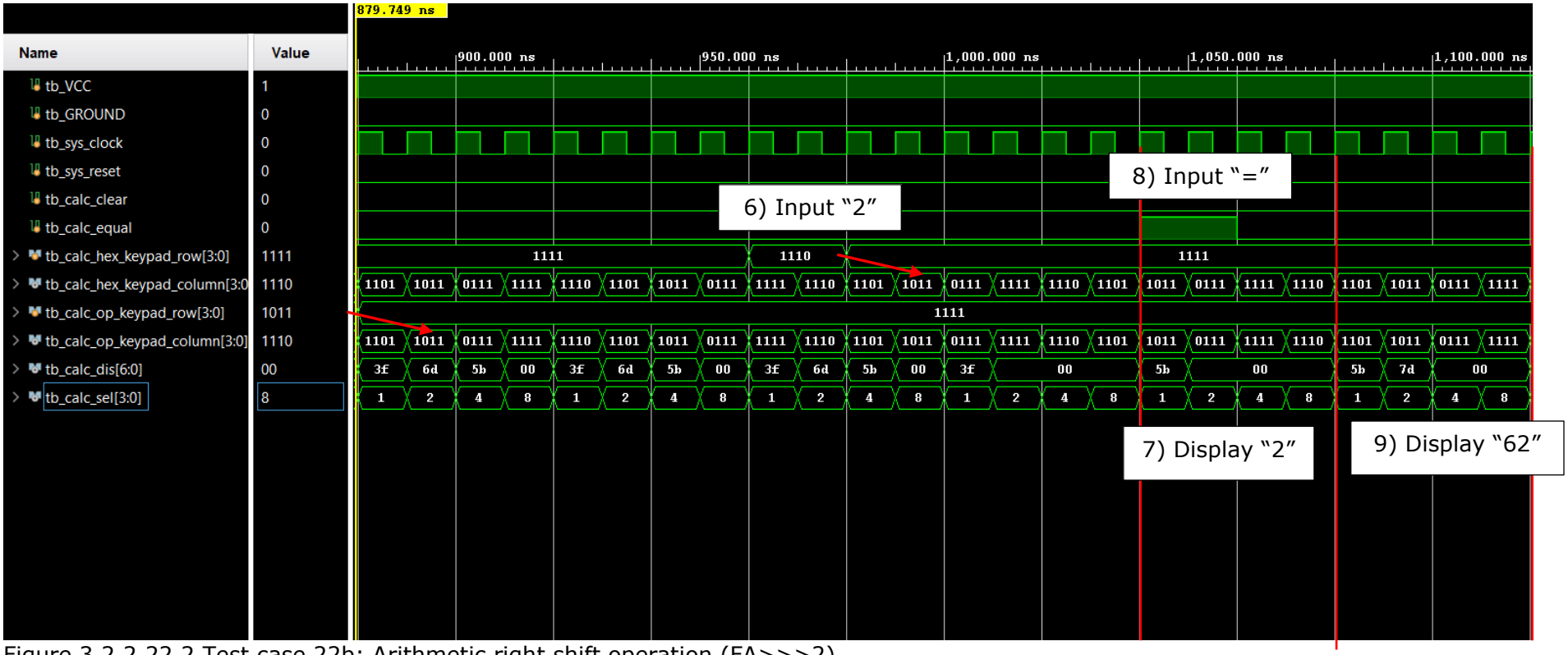

Figure 3.2.2.22.2 Test case 22b: Arithmetic right shift operation (FA>>>2)

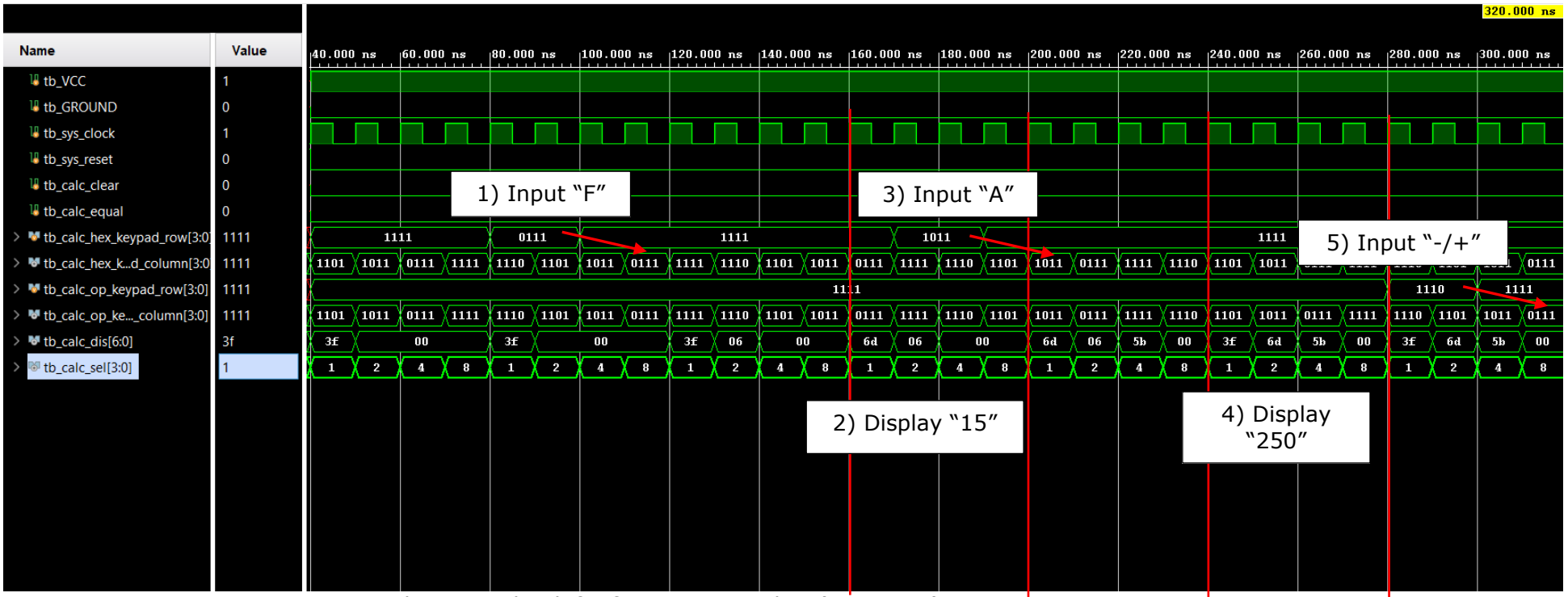

Figure 3.2.2.23.1 Test case 23a: Arithmetic right shift of negative number (-FA>>>2)

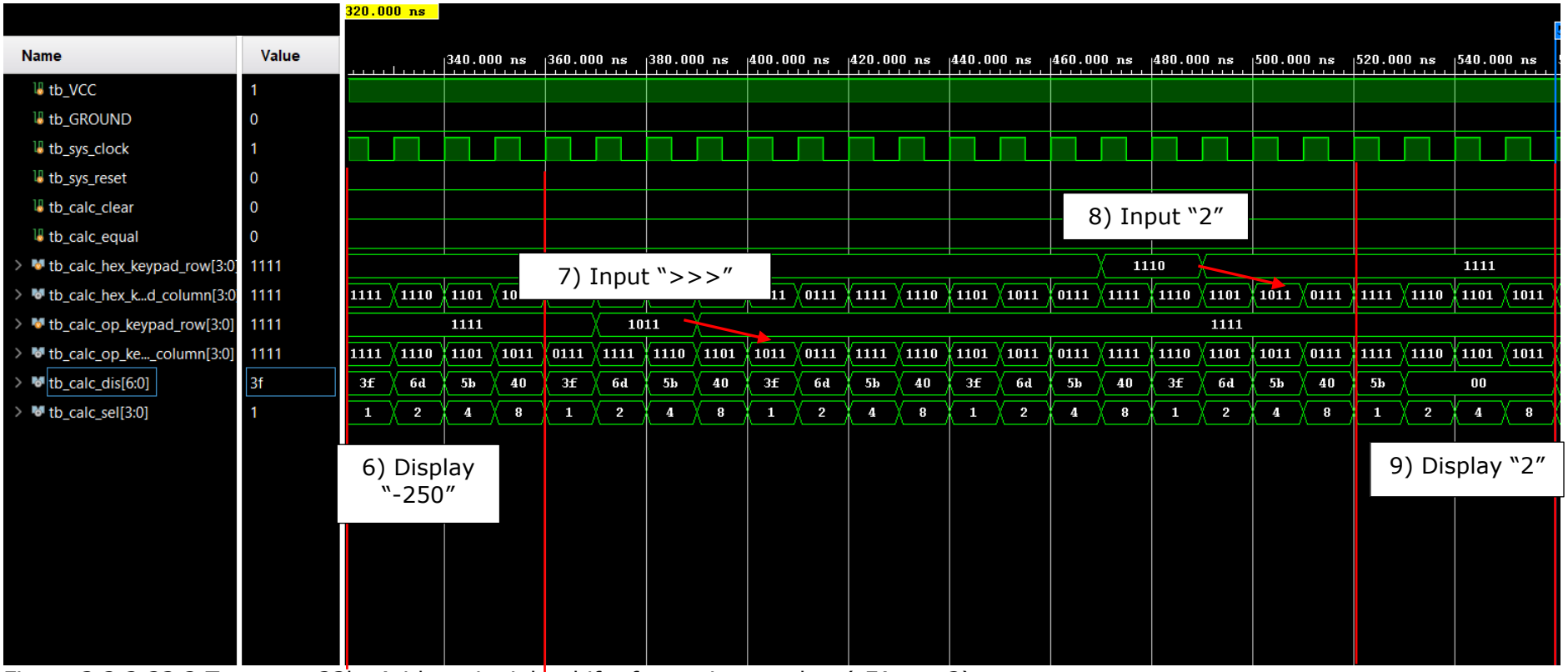

Figure 3.2.2.23.2 Test case 23b: Arithmetic right shift of negative number (-FA>>>2)

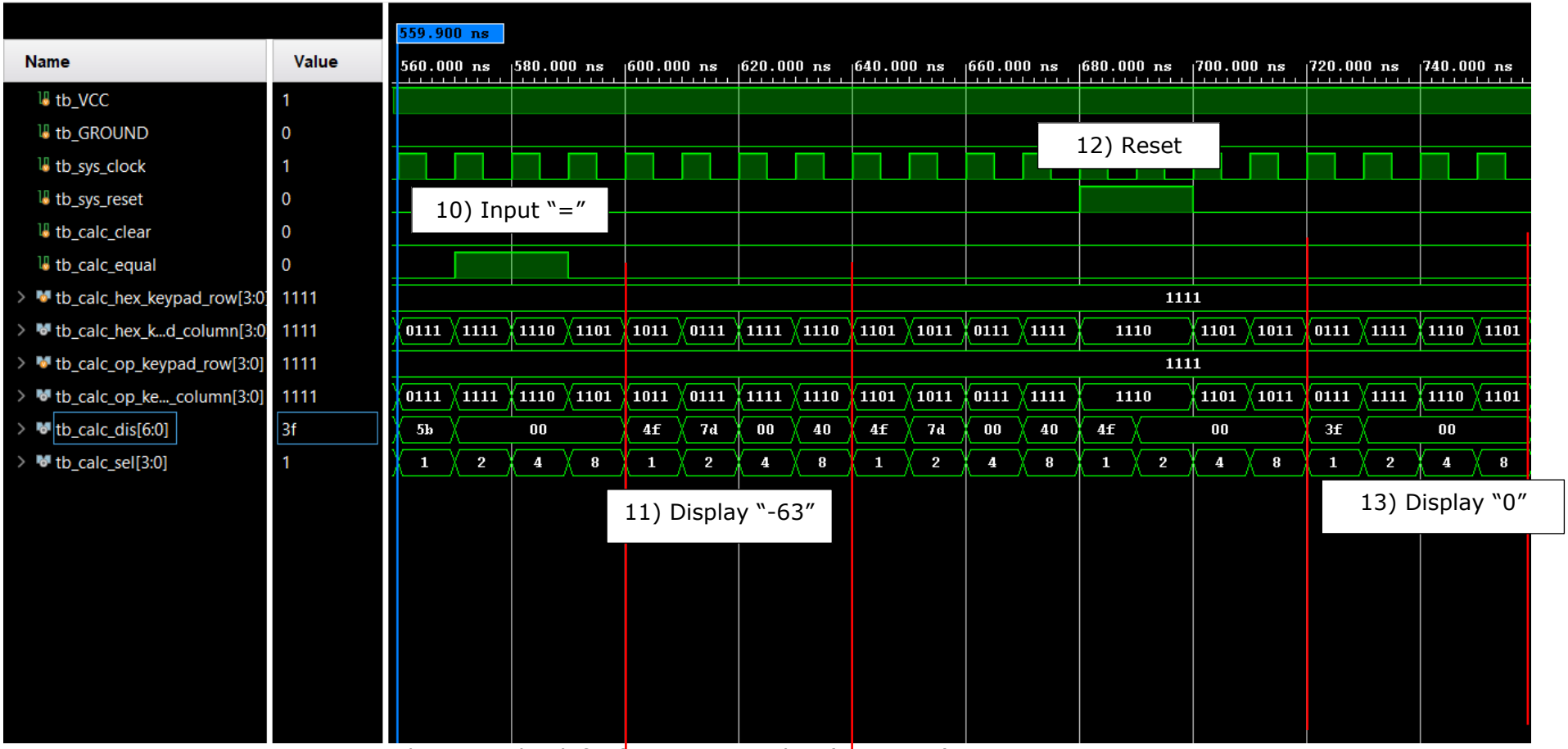

Figure 3.2.2.23.3 Test case 23c: Arithmetic right shift of negative number (-FA>>>2)

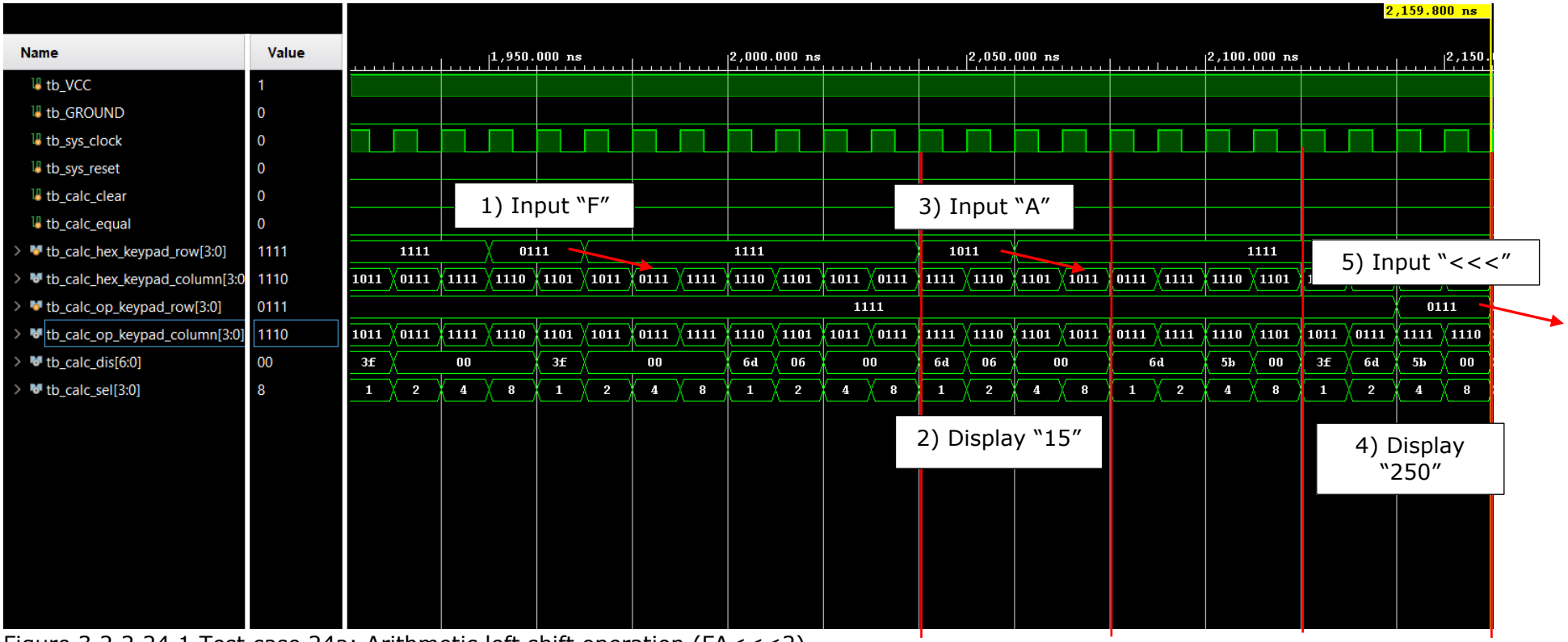

Figure 3.2.2.24.1 Test case 24a: Arithmetic left shift operation (FA<<<2)

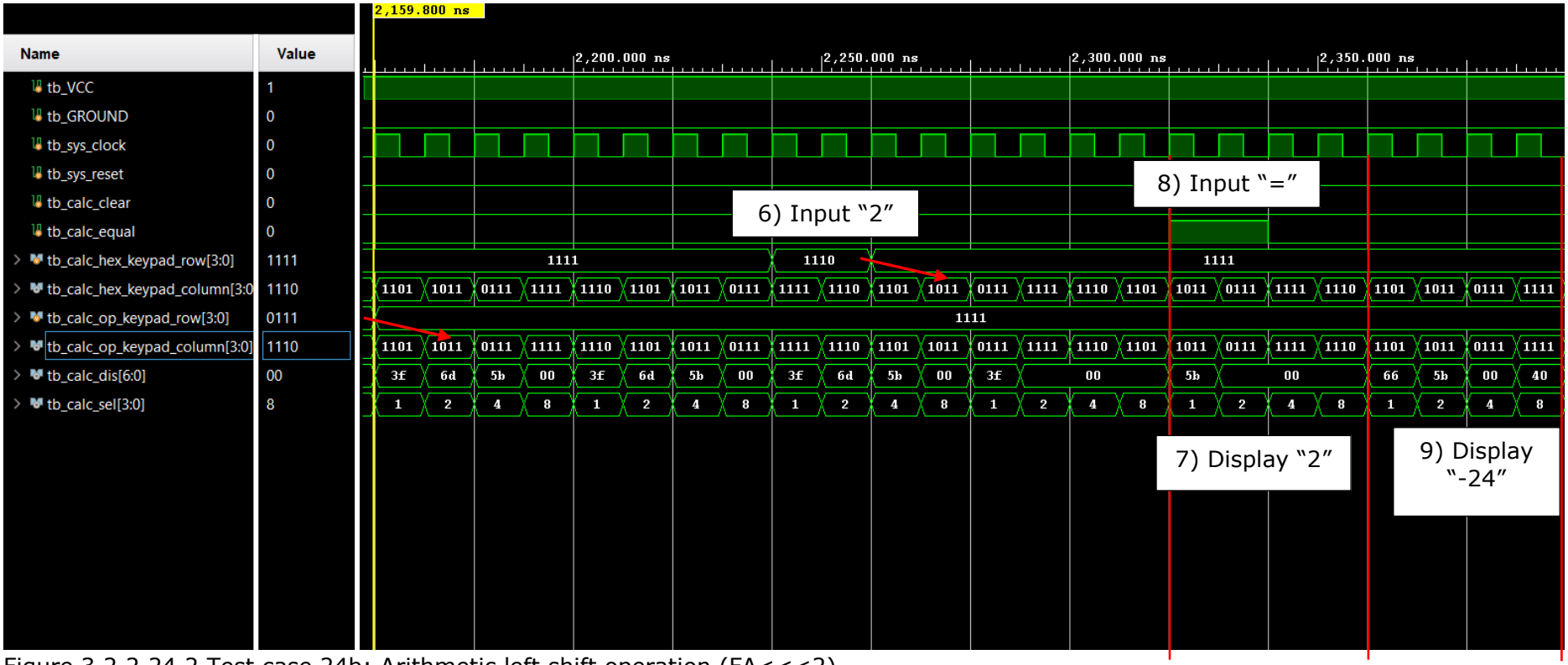

Figure 3.2.2.24.2 Test case 24b: Arithmetic left shift operation (FA<<<2)

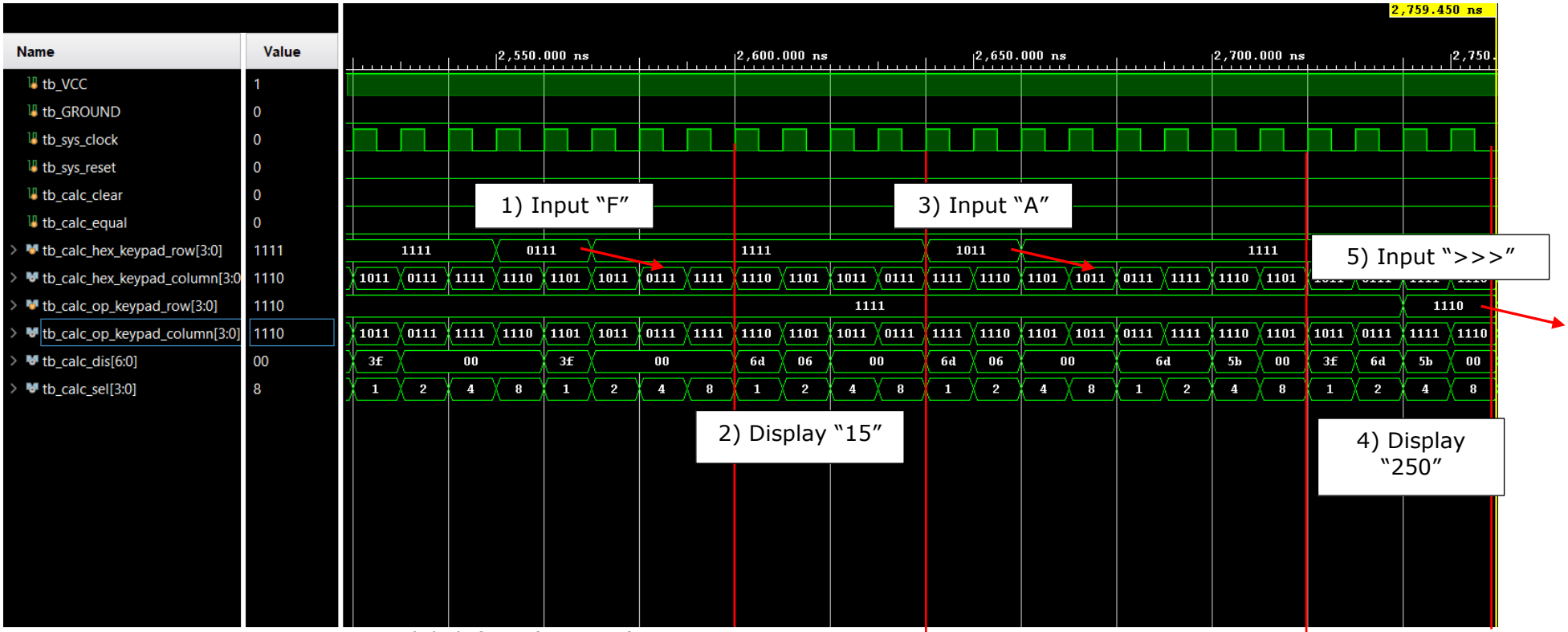

Figure 3.2.2.25.1 Test case 25a: Invalid shifting (FA>>-2)

x

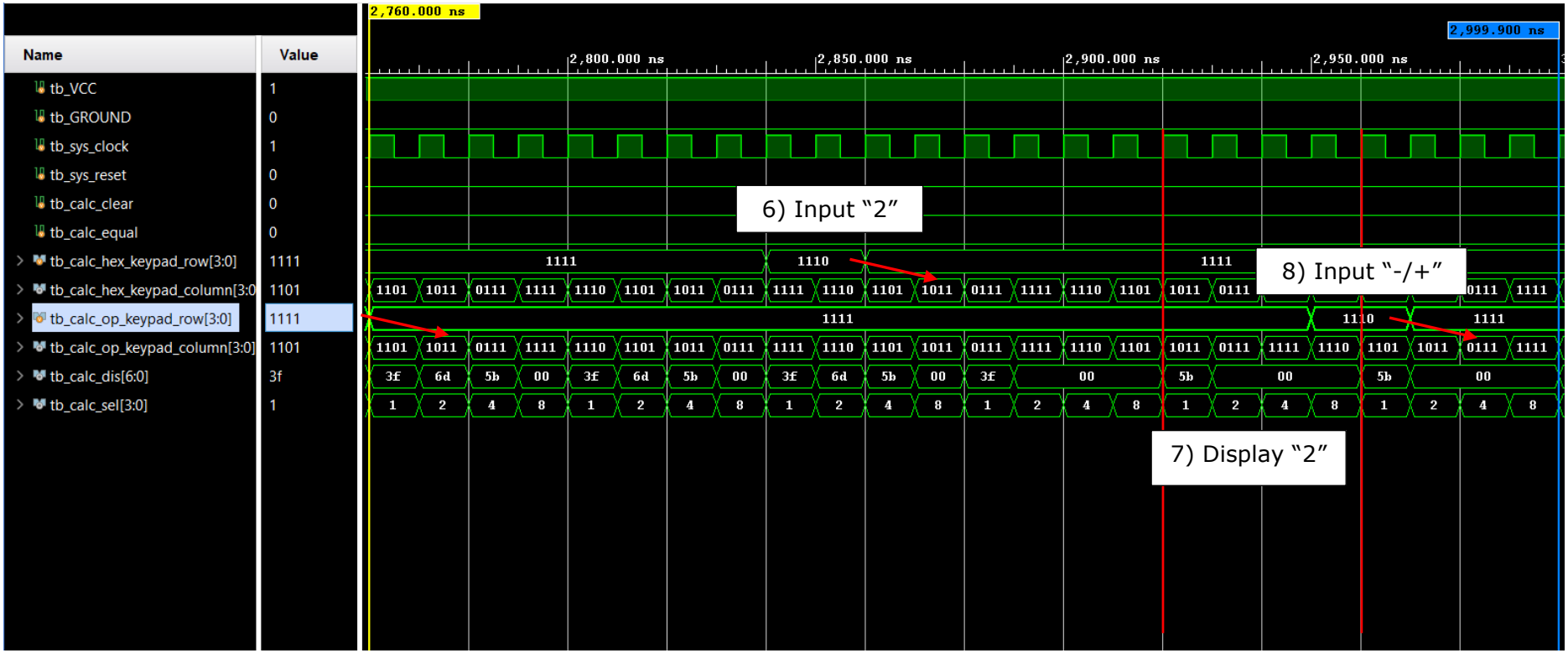

Figure 3.2.2.25.2 Test case 25b: Invalid shifting (FA>>-2)

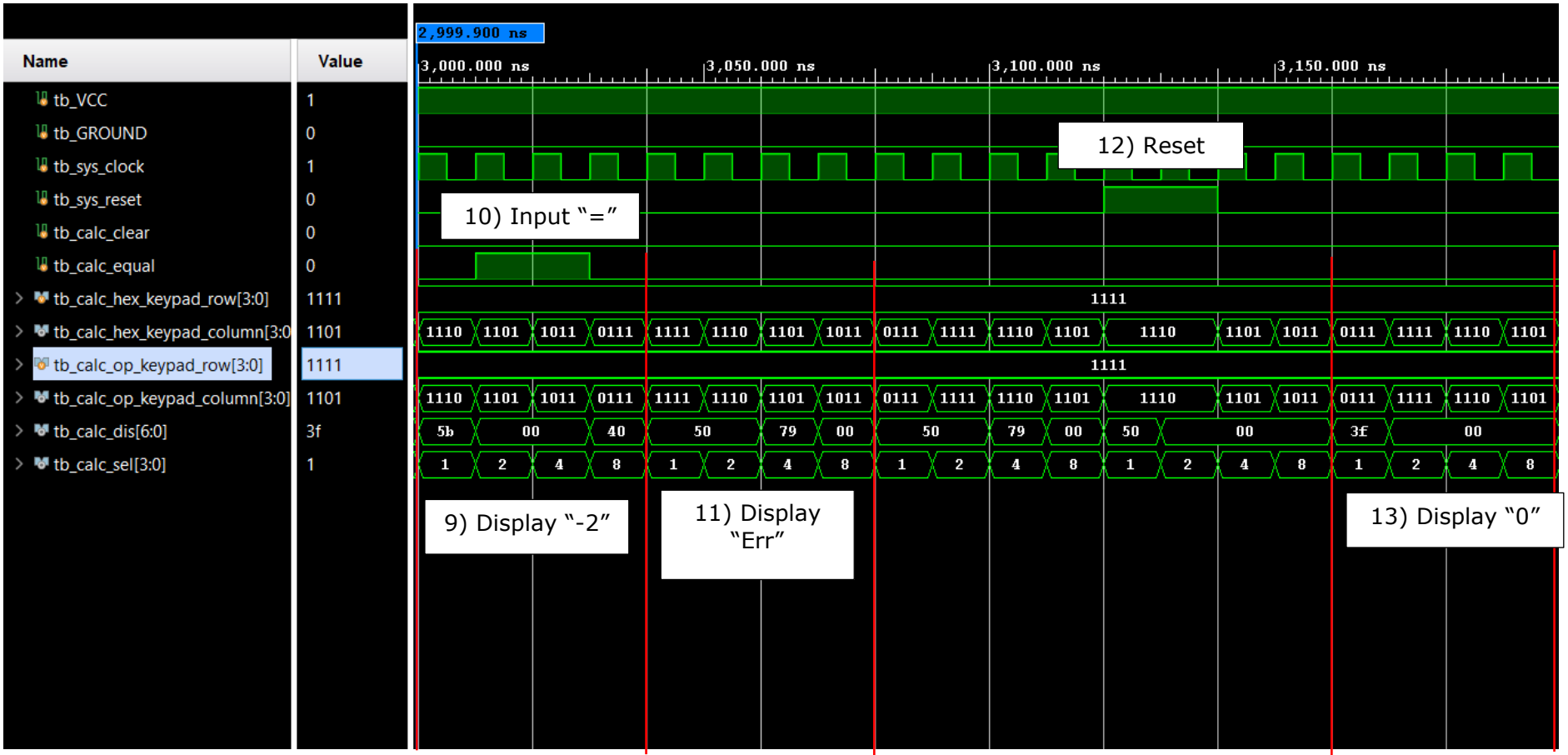

Figure 3.2.2.25.3 Test case 25c: Invalid shifting (FA>>-2)

4 Micro-Architecture Specification (Block level)

4.1 Synchronizer

4.1.1 Functionality/features

- Synchronizes asynchronous input signals such as keypad rows and control signals to the system clock domain to prevent metastability and ensure reliable operation in a synchronous system.

4.1.2 Block interface and I/O pin description

4.1.2.1 Synchronizer block interface

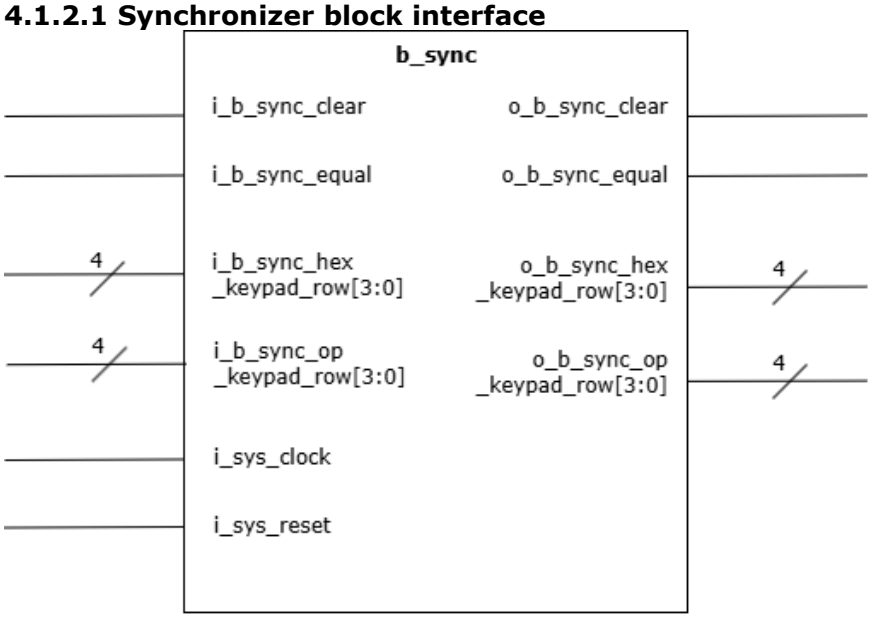

Diagram 4.1.2.1: Block interface of synchronizer

4.1.2.2 I/O pin description

Pin name: **Pin class:** **Pin function:**	i_b_sync_clear control to clear the number keypad input	**Source →Destination:**	Clear button→ Synchronizer
Pin name: **Pin class:** **Pin function:**	i_b_sync_equal control to display the output result	**Source →Destination:**	Equal button → Synchronizer
Pin name: **Pin class:** **Pin function:**	i_b_sync_hex_keypad_row[3:0] data to detect which row of hexadecimal keypad is pressed	**Source →Destination:**	Hexadecimal keypad → Synchronizer
Pin name: **Pin class:** **Pin function:**	i_b_sync_op_keypad_row[3:0] data to detect which row of operator keypad is pressed	**Source →Destination:**	Operator keypad → Synchronizer
Pin name: **Pin class:** **Pin function:**	i_sys_clock control to provide a system clock to the calculator	**Source →Destination:**	Clock generator → synchronizer
Pin name: **Pin class:** **Pin function:**	i_sys_reset control to reset the calculator	**Source →Destination:**	ON/Reset button→ synchronizer

Pin name: **Pin class:** **Pin function:**	o_b_sync_op_keypad_row[3:0] data to pass synced operator keypad row input to other blocks	**Source →Destination:**	Synchronizer → Operator keypad
Pin name: **Pin class:** **Pin function:**	o_b_sync_hex_keypad_row[3:0] data to pass synced hex keypad row input to other blocks	**Source →Destination:**	Synchronizer → Hexadecimal keypad
Pin name: **Pin class:** **Pin function:**	o_b_sync_clear control to pass synced clear button signal to other blocks	**Source →Destination:**	Synchronizer → Hex keypad code generator
Pin name: **Pin class:** **Pin function:**	o_b_sync_equal control to pass synced equal button signal to other blocks	**Source →Destination:**	Synchronizer → ALU calculator & 7-segment display decoder

Table 4.1.2.2: I/O pin description of synchronizer block

4.1.3 Internal Operation: Function Table

Clock signal i_sys_clock	Input(Qn)	Output	Synchronizer function
↑ 0 → 1 (rising edge)	Qn	Previous output	Latches synced input delayed 2 clock cycle without metastability issue
↑ 0 → 1 (next rising edge)	Qn	Qn	
↓ 1 → 0 (falling edge)	Qn	Previous output	No change to output

Table 4.1.3: Function table of synchronizer block

4.1.4 Timing Requirement

Signal	Requirement
All inputs	**More than 2 clock cycle** -Inputs must be stable for more than 2 clock cycles before being sampled
Synchronizer latency	**2 clock cycles** -to ensure metastability protection

Table 4.1.4: Timing requirements of synchronizer block

4.1.5 Schematic Diagram

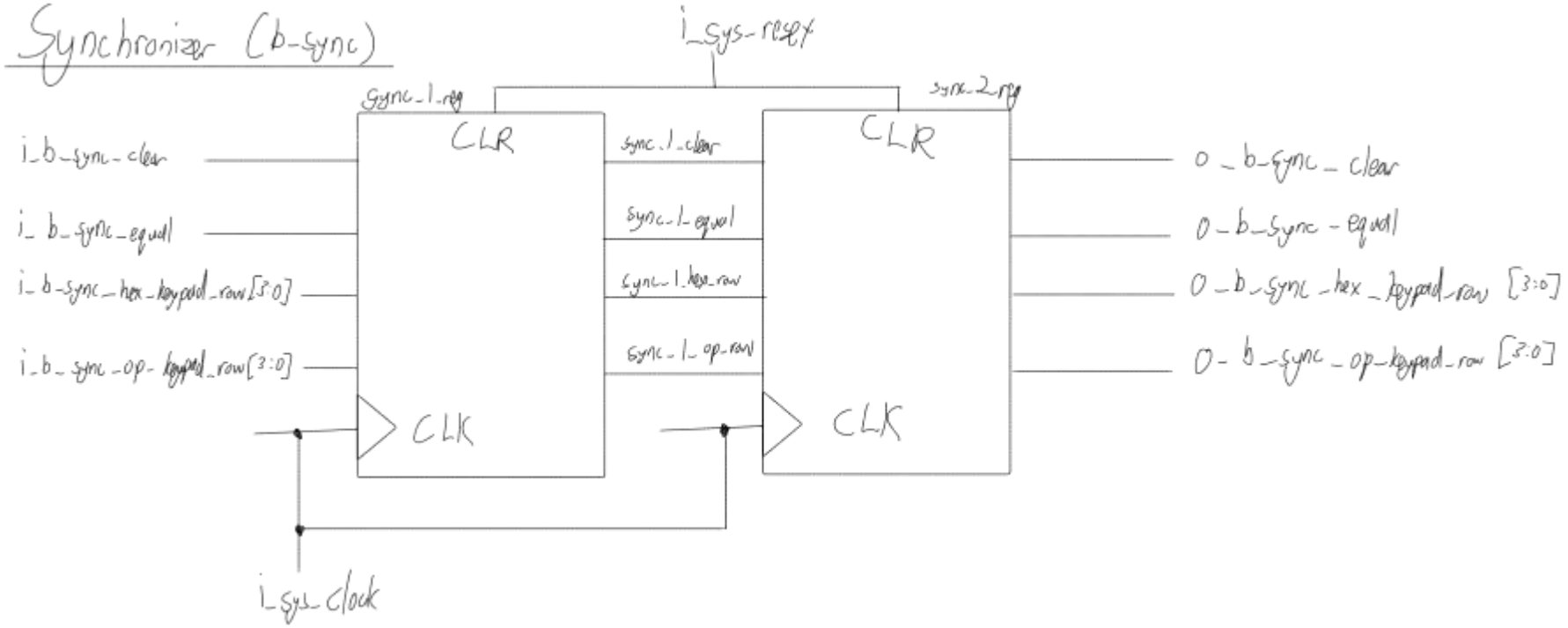

Figure 4.1.5.2 Micro architecture of synchronizer

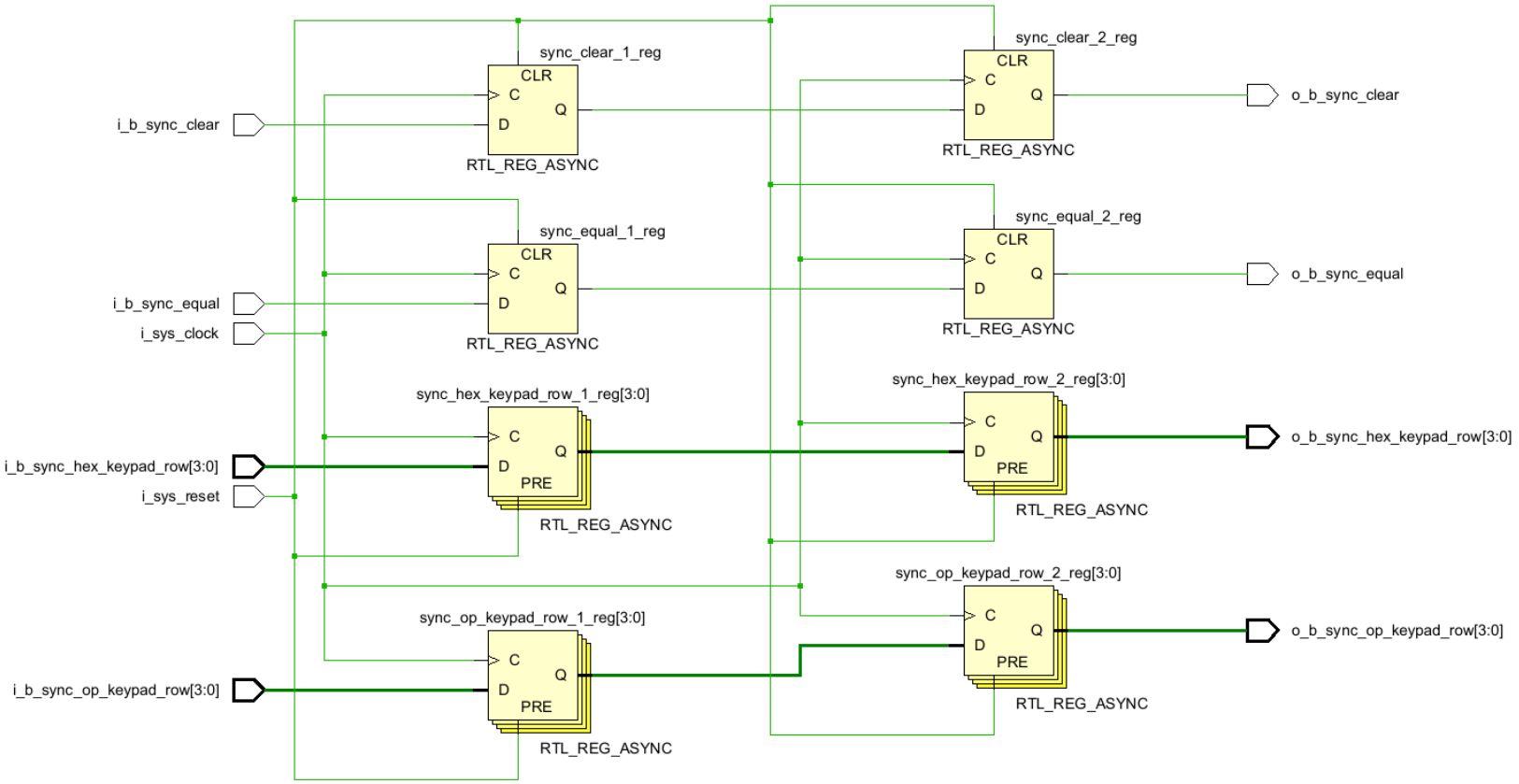

Figure 4.1.5.2 Schematic Diagram of synchronizer

4.1.6 System Verilog Model

```
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Yu Heng
//
// Create Date: 21.04.2025 20:24:32
// File Name: b_sync.sv
// Module Name: b_sync
// Project Name: 8-bit integer calculator
// Code Type: RTL level
// Description: Modelling of synchronizer
//
//////////////////////////////////////////////////////////////////////////////////

module b_sync (
   input  logic       i_b_sync_clear,
   input  logic       i_b_sync_equal,
   input  logic [3:0]  i_b_sync_hex_keypad_row,
   input  logic [3:0]  i_b_sync_op_keypad_row,
   input  logic       i_sys_clock,
   input  logic       i_sys_reset,
   output logic        o_b_sync_clear,
   output logic        o_b_sync_equal,
   output logic [3:0]  o_b_sync_hex_keypad_row,
   output logic [3:0]  o_b_sync_op_keypad_row
);

   // First stage registers
   logic       sync_clear_1;
   logic       sync_equal_1;
   logic [3:0]  sync_hex_keypad_row_1;
   logic [3:0]  sync_op_keypad_row_1;

   // Second stage registers
   logic       sync_clear_2;
   logic       sync_equal_2;
   logic [3:0]  sync_hex_keypad_row_2;
   logic [3:0]  sync_op_keypad_row_2;

   // Double flip flop synchronizer
   always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin
      if (i_sys_reset) begin
         // Reset all registers
         sync_clear_1 <= 1'b0;
         sync_equal_1 <= 1'b0;
         sync_hex_keypad_row_1 <= 4'b1111;
         sync_op_keypad_row_1 <= 4'b1111;

         sync_clear_2 <= 1'b0;
         sync_equal_2 <= 1'b0;
         sync_hex_keypad_row_2 <= 4'b1111;
         sync_op_keypad_row_2 <= 4'b1111;
      end

      else begin
         // First stage flip flop (capture inputs to stage 1 registers)
         sync_clear_1 <= i_b_sync_clear;
```

```
            sync_equal_1 <= i_b_sync_equal;
            sync_hex_keypad_row_1 <= i_b_sync_hex_keypad_row;
            sync_op_keypad_row_1 <= i_b_sync_op_keypad_row;

            // Second stage flip flop(transfer from stage 1 to stage 2 registers)
            sync_clear_2 <= sync_clear_1;
            sync_equal_2 <= sync_equal_1;
            sync_hex_keypad_row_2 <= sync_hex_keypad_row_1;
            sync_op_keypad_row_2 <= sync_op_keypad_row_1;
        end
    end

    // Connect synchronized registers to outputs
    assign o_b_sync_clear = sync_clear_2;
    assign o_b_sync_equal = sync_equal_2;
    assign o_b_sync_hex_keypad_row = sync_hex_keypad_row_2;
    assign o_b_sync_op_keypad_row = sync_op_keypad_row_2;
endmodule
```

4.1.7 Test Plan

No.	Test Case	Description of test vector Generation	Expected Output	Status
1	Asynchronous input change of clear button	1. Set i_b_sync_clear to 1 for 5 clock cycle	o_b_sync_clear = 1 after one clock cycle (from 2nd clock cycle from the input until 6th clock cycle)	Pass
2	Asynchronous input change of equal button	1. Set i_b_sync_equal to 1 for 5 clock cycle	o_b_sync_equal = 1 after one clock cycle (from 2nd clock cycle from the input until 6th clock cycle)	Pass
3	Asynchronous input change of hex keypad row signal	1. Set i_b_sync_hex_keypad_row to 1011 for 5 clock cycle	o_b_sync_hex_keypad_row = 1011 after one clock cycle (from 2nd clock cycle from the input until 6th clock cycle)	Pass
4	Asynchronous input change of operator keypad row signal	1. Set i_b_sync_op_keypad_row to 1110 for 5 clock cycle	o_b_sync_hex_keypad_row = 1110 after one clock cycle (from 2nd clock cycle from the input until 6th clock cycle)	Pass
5	Reset button pressed	1. Set i_b_sync_clear to 1 2.Set i_b_sync_equal to 3.Set i_b_sync_hex_keypad_row to 1011 4.Set i_b_sync_op_keypad_row to 1110 5.Set i_sys_reset to 1 for 5 clock cycle	o_b_sync_op_keypad_row = 4'b1111 o_b_sync_hex_keypad_row = 4'b1111 o_b_sync_clear = 0 o_b_sync_equal = 0 when the reset is high (after reset go to low, if the input is still key pressed value, it will be indicated as new input)	pass

Table 4.1.5: Test plan for synchronizer block

4.1.8 Testbench and Simulation result

4.1.8.1 Testbench

```
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Yu Heng
//
// Create Date: 21.04.2025 19:16:19
// File Name: tb_b_sync.sv
// Module Name: tb_b_sync
// Project Name: 8-bit integer calculator
// Code Type: Behavioural
// Description: Testbench for synchronizer
//
//////////////////////////////////////////////////////////////////////////////////

module tb_b_sync;

  logic       tb_i_sys_clock;
  logic       tb_i_sys_reset;
  logic       tb_i_b_sync_clear;
  logic       tb_i_b_sync_equal;
  logic [3:0]  tb_i_b_sync_hex_keypad_row;
  logic [3:0]  tb_i_b_sync_op_keypad_row;
  logic       tb_o_b_sync_clear;
  logic       tb_o_b_sync_equal;
  logic [3:0]  tb_o_b_sync_hex_keypad_row;
  logic [3:0]  tb_o_b_sync_op_keypad_row;

  b_sync b_sync_dut (
    .i_b_sync_clear(tb_i_b_sync_clear),
    .i_b_sync_equal(tb_i_b_sync_equal),
    .i_b_sync_hex_keypad_row(tb_i_b_sync_hex_keypad_row),
    .i_b_sync_op_keypad_row(tb_i_b_sync_op_keypad_row),
    .i_sys_clock(tb_i_sys_clock),
    .i_sys_reset(tb_i_sys_reset),
    .o_b_sync_clear(tb_o_b_sync_clear),
    .o_b_sync_equal(tb_o_b_sync_equal),
    .o_b_sync_hex_keypad_row(tb_o_b_sync_hex_keypad_row),
    .o_b_sync_op_keypad_row(tb_o_b_sync_op_keypad_row)
  );

    // Clock generation
    always #5 tb_i_sys_clock = ~tb_i_sys_clock; // 10ns period, 50MHz

    // Reset
    initial begin
        tb_i_sys_clock = 1;
        tb_i_sys_reset = 1;
        #10;
        tb_i_sys_reset = 0;
    end

    initial begin
        // Initialize all inputs
        tb_i_b_sync_clear = 0;
        tb_i_b_sync_equal = 0;
        tb_i_b_sync_hex_keypad_row = 4'b1111;
        tb_i_b_sync_op_keypad_row = 4'b1111;
```

```
        //------------------------------------------------------------------
        // Test Case 1: Asynchronous input change of clear button
        //------------------------------------------------------------------
        #20;
        tb_i_b_sync_clear = 1;
        #50; //5 clock cycles
        tb_i_b_sync_clear = 0;
        #20;

        //------------------------------------------------------------------
        // Test Case 2: Asynchronous input change of equal button
        //------------------------------------------------------------------
        #20;
        tb_i_b_sync_equal = 1;
        #50;// 5 clock cycles
        tb_i_b_sync_equal = 0;
        #20;

        //------------------------------------------------------------------
        // Test Case 3: Asynchronous input change of hex keypad row signal
        //------------------------------------------------------------------
        #20;
        tb_i_b_sync_hex_keypad_row = 4'b1011;
        #50;//5 clock cycles
        tb_i_b_sync_hex_keypad_row = 4'b1111; // Return to default
        #20;

        //------------------------------------------------------------------
        // Test Case 4: Asynchronous input change of operator keypad row signal
        //------------------------------------------------------------------
        #20;
        tb_i_b_sync_op_keypad_row = 4'b1110;
        #50;//5 clock cycles
        tb_i_b_sync_op_keypad_row = 4'b1111;
        #20;

        //------------------------------------------------------------------
        // Test Case 5: Reset button pressed
        //------------------------------------------------------------------
        #20;
        tb_i_b_sync_clear = 1;
        tb_i_b_sync_equal = 1;
        tb_i_b_sync_hex_keypad_row = 4'b1011;
        tb_i_b_sync_op_keypad_row = 4'b1110;
        #20
        tb_i_sys_reset = 1;
        #50; // simulate pressing reset button
        tb_i_sys_reset = 0;

        #150;
        $stop;
    end


endmodule
```

4.1.8.2 Simulation Result

test case 1: Asynchronous input change of clear button

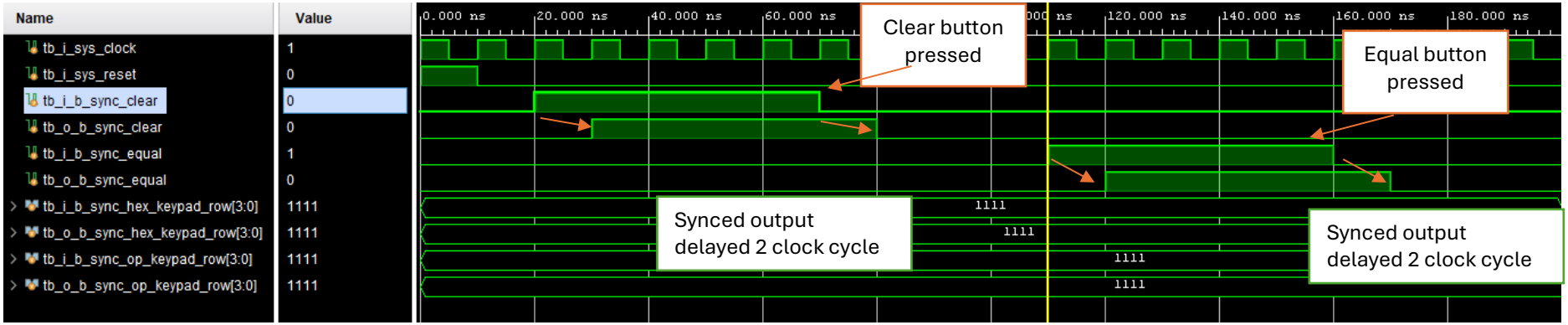

Figure 4.1.8.2.1 Test case 1,2

test case 2: Asynchronous input change of equal button

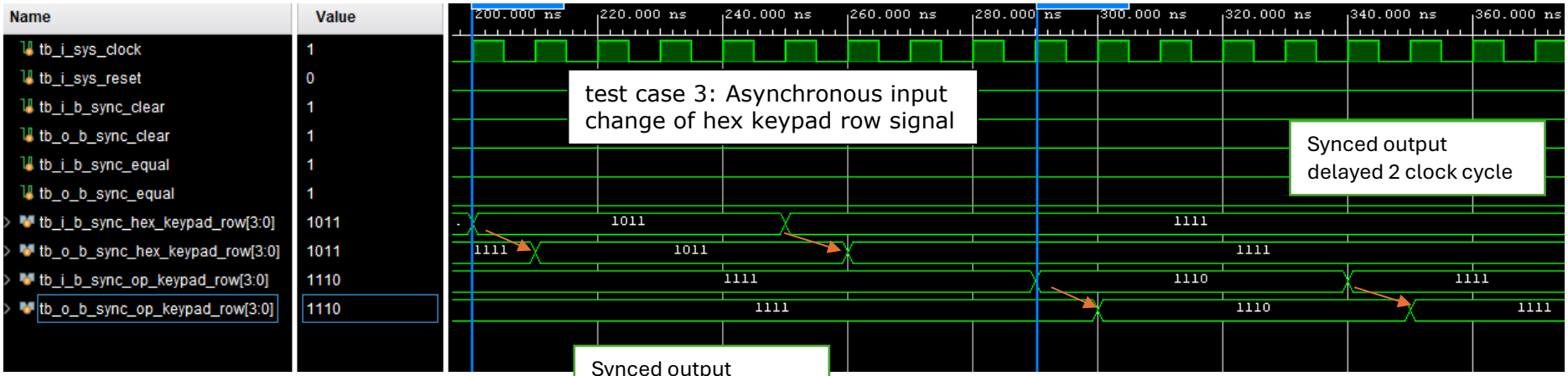

Figure 4.1.8.2.2 Test case

test case 4: Asynchronous input change of operator keypad row signal

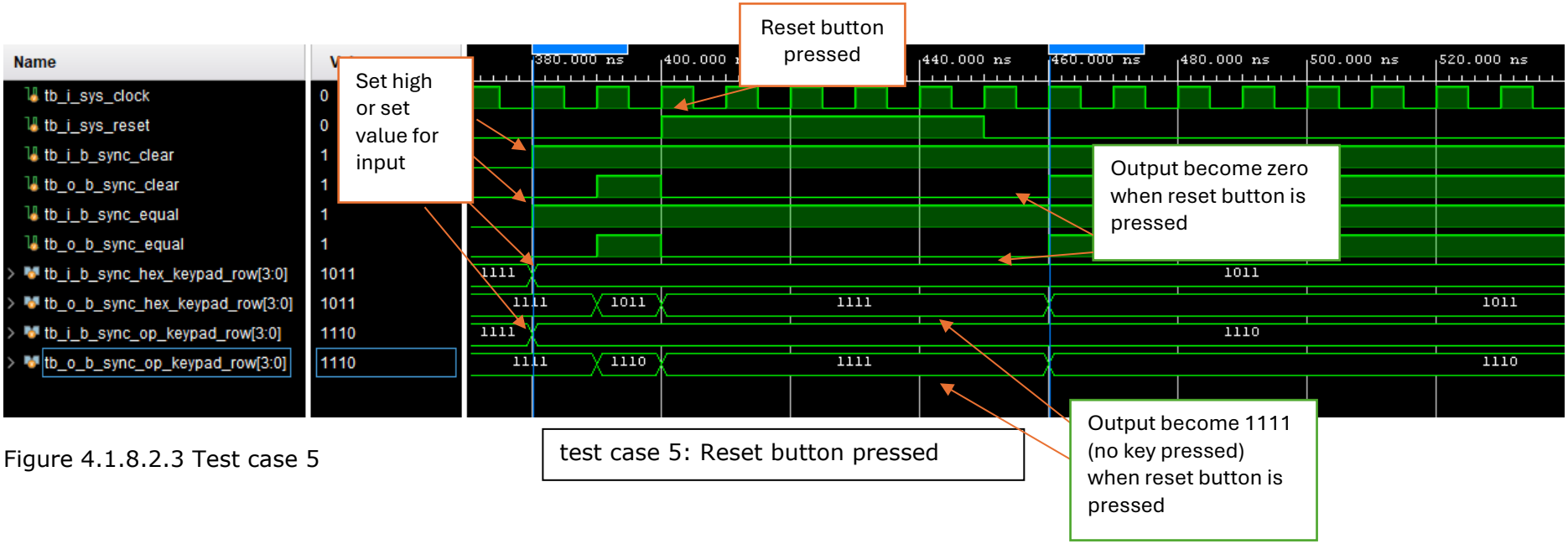

Figure 4.1.8.2.3 Test case 5

4.2 Operator keypad code generator

4.2.1 Functionality/features

- Converts the operator keypad row and column inputs into a 4-bit operator code
- Generates a valid code flag when the code is valid
- Generates a negative flag when minus sign is used as negative

4.2.2 Block interface and I/O pin description

4.2.2.1 Operator keypad code generator block interface

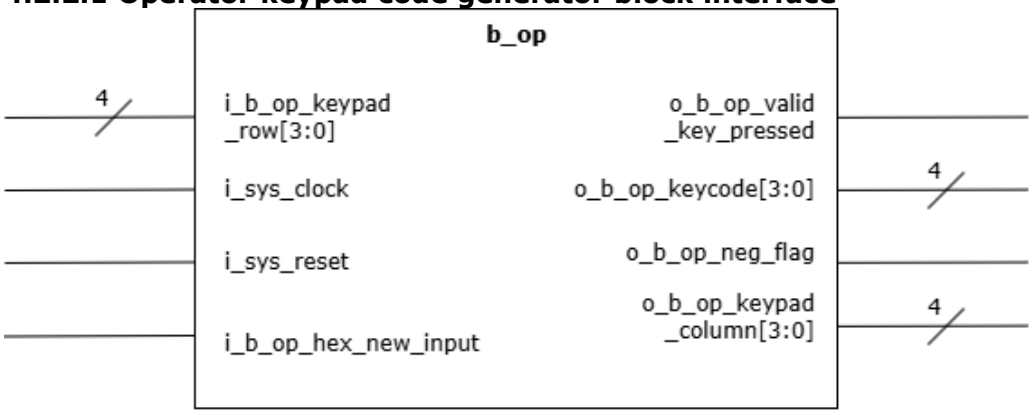

Diagram 4.2.2.1: Block interface of operator keypad code generator

4.2.2.2 I/O pin description

Pin name: **Pin class:** **Pin function:**	i_b _op_keypad_row[3:0] data to detect which row of operator keypad is pressed	**Source →Destination:**	Synchronizer →Operator keypad code generator
Pin name: **Pin class:** **Pin function:**	i_sys_clock control to provide a system clock to the calculator	**Source →Destination:**	Clock generator → Operator keypad code generator
Pin name: **Pin class:** **Pin function:**	i_sys_reset control to reset the calculator	**Source →Destination:**	ON/Reset button→ Operator keypad code generator
Pin name: **Pin class:** **Pin function:**	i_b_op_hex_new_input control to clear negative flag when new input is keyed in	**Source →Destination:**	Hex keypad code generator → Operator keypad code generator
Pin name: **Pin class:** **Pin function:**	o_b_op_keycode[3:0] data to pass the decoded operator code for operation later	**Source →Destination:**	Operator keypad code generator → ALU calculator
Pin name: **Pin class:** **Pin function:**	o_b_op_valid_key_pressed control to indicate whether the operator key is pressed	**Source →Destination:**	Operator keypad code generator → Hex keypad code

			generator & ALU calculator & 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	o_b_op_neg_flag control to provide negative flag	**Source →Destination:**	Operator keypad code generator → ALU calculator & 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	o_b_op_keypad_column[3:0] data to detect which column of operator keypad is pressed	**Source →Destination:**	Operator keypad code generator → Operator keypad

Table 4.2.2.2: I/O pin description for operator keypad code generator

4.2.3 Internal Operation: Function Table

Operator Keypad row i_b _op _keypad _row [3:0]	**New Input Flag i_b_op _hex_ new_in put**	**Operato r keypad column o_b_op _keypa d _colum n [3:0]**	**Negative Flag o_b _op_neg _flag**	**Keycode o_b_op _keycode [3:0]**	**Valid Flag o_b _op _valid _key _presse d**	**Operator keypad code generator function**
4’b1110	0	4’b1110	Previous value	4’b0000	1	“Add” operation
4’b1101	0	4’b1110	Previous value	4’b0001	1	“Minus” operation
4’b1011	0	4’b1110	Previous value	4’b0010	1	“Multiply” operation
4’b0111	0	4’b1110	Previous value	4’b0011	1	“Divide” operation
4’b1110	0	4’b1101	Previous value	4’b0100	1	“NOT” operation
4’b1101	0	4’b1101	Previous value	4’b0101	1	“AND” operation
4’b1011	0	4’b1101	Previous value	4’b0110	1	“OR” operation
4’b0111	0	4’b1101	Previous value	4’b0111	1	“XOR” operation
4’b1110	0	4’b1011	Previous value	4’b1000	1	“Bitwise right shift” operation
4’b1101	0	4’b1011	Previous value	4’b1001	1	“Bitwise left shift” operation

4’b1011	0	4’b1011	Previous value	4’b1010	1	“Arithmetic right shift” operation
4’b0111	0	4’b1011	Previous value	4’b1011	1	“Arithmetic left shift” operation
4’b1110	0	4’b0111	1	4’b1111	0	Change sign: + to - / - to +
4’bXXX1	0	4’b0111	0	4’b1111	0	Invalid operator input for empty key
4’bXXXX	1	4’bXXXX	0	4’bXXXX	0	Remove negative flag when new input is keyed in

Table 4.2.3: Function table of operator keypad code generator

4.2.4 Timing Requirement

Signal	**Requirement**
i_b_op_keypad_row[3:0]	**More than 10 clock cycles** - Must be stable while corresponding column is scanning for 10 (5x2) clock cycles in a loop (real hardware implementation)
i_b_op_hex_new_input	**1 clock cycle** -to read the new input flag from hex keypad code generator
o_b_op_keypad_column[3:0])	**1 clock cycles** - Must scan each column for 1 clock cycles in a loop
o_b_op_valid_key_pressed	**2 clock cycle** -to ensure other blocks read the flag
o_b_op_keycode[3:0]	**1 clock cycle** -to pass the input operator keycode to keycode register

Table 4.2.4: Timing requirement of operator keypad code generator

4.2.5 Schematic Diagram

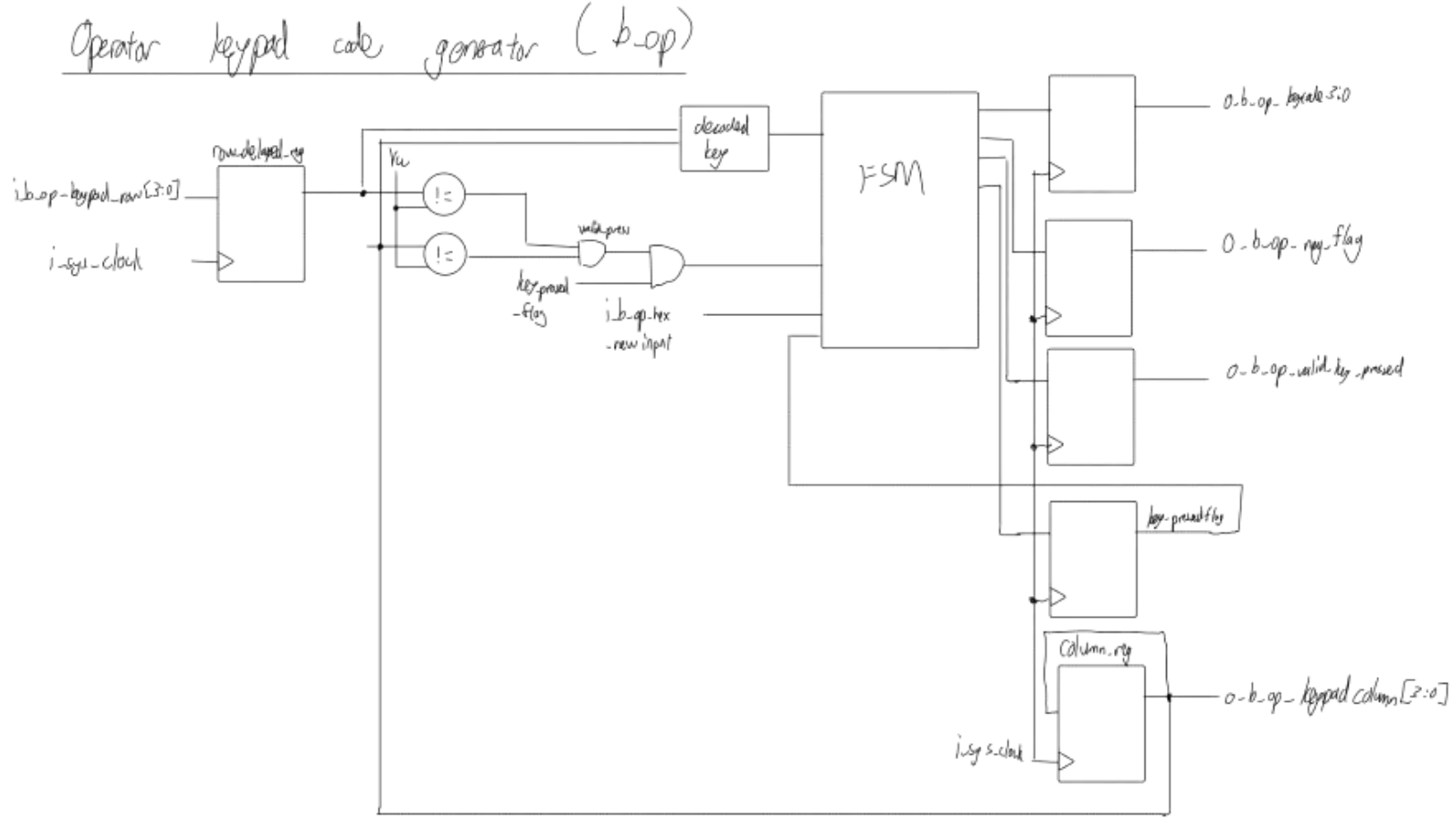

Figure 4.2.5.1 Microarchitecture of operator keypad code generator

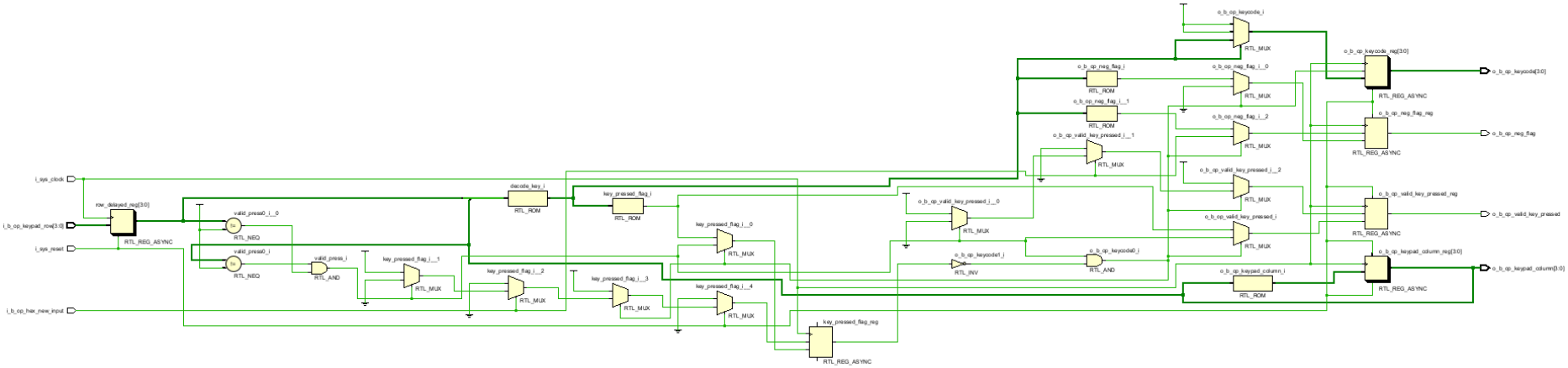

Figure 4.2.5.2 Schematic Diagram of operator keypad code generator

4.2.6 System Verilog Model

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Yu Heng
//
// Create Date: 20.04.2025 10:34:46
// File Name: b_op.sv
// Module Name: b_op
// Project Name: 8-bit integer calculator
// Code Type: RTL level
// Description: Modelling of operator keypad code generator
//
//////////////////////////////////////////////////////////////////////////////////

module b_op (
    input  logic       i_sys_clock,
    input  logic       i_sys_reset,
    input  logic       i_b_op_hex_new_input,
    input  logic [3:0]  i_b_op_keypad_row,
    output logic [3:0]  o_b_op_keycode,
    output logic       o_b_op_valid_key_pressed,
    output logic       o_b_op_neg_flag,
    output logic [3:0]  o_b_op_keypad_column
);

    logic [3:0] temp_op;
    logic key_pressed_flag;//to indicate if the button is pressed
    logic valid_press;//to indicate valid key press
    logic [3:0] row_delayed;      // delay row by one clock cycle to synchronize wave
output with the column

    // Delay the Row signal by one clock cycle
    always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin
        if (i_sys_reset)
            row_delayed <= 4'b1111; // idle state (all unpressed)
        else
            row_delayed <= i_b_op_keypad_row;
    end


    // Drive the column cycling
    always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin//loop to scan the
column keypad
        if (i_sys_reset)
        o_b_op_keypad_column <= 4'b1110; // Start with column 0 active
        else begin
            case (o_b_op_keypad_column)
                4'b1110: o_b_op_keypad_column <= 4'b1101; // col 1
                4'b1101: o_b_op_keypad_column <= 4'b1011; // col 2
                4'b1011: o_b_op_keypad_column <= 4'b0111; // col 3
                4'b0111: o_b_op_keypad_column <= 4'b1111; // idle
                default: o_b_op_keypad_column <= 4'b1110; // loop back to col 0
            endcase
        end
    end
```

```
   function logic [3:0] decode_key (input logic [3:0] row, input logic [3:0] col);
     // function to convert row and column read into hex decoded key
       case ({row, col})
          {4'b1110, 4'b1110}: decode_key = 4'b0000; // "Add" operation
          {4'b1101, 4'b1110}: decode_key = 4'b0001; // "Minus" operation
          {4'b1011, 4'b1110}: decode_key = 4'b0010; // "Multiply" operation
          {4'b0111, 4'b1110}: decode_key = 4'b0011; // "Divide" operation
          {4'b1110, 4'b1101}: decode_key = 4'b0100; // "NOT" operation
          {4'b1101, 4'b1101}: decode_key = 4'b0101; // "AND" operation
          {4'b1011, 4'b1101}: decode_key = 4'b0110; // "OR" operation
          {4'b0111, 4'b1101}: decode_key = 4'b0111; // "XOR" operation
          {4'b1110, 4'b1011}: decode_key = 4'b1000; // "Bitwise right shift" operation
          {4'b1101, 4'b1011}: decode_key = 4'b1001; // "Bitwise left shift" operation
          {4'b1011, 4'b1011}: decode_key = 4'b1010; // "Arithmetic right shift"
operation
          {4'b0111, 4'b1011}: decode_key = 4'b1011; // "Arithmetic left shift" operation
          {4'b1110, 4'b0111}: decode_key = 4'b1100; // Change sign: + to - / - to +
          default:            decode_key = 4'b1111; // Invalid operator input for empty key
        endcase
   endfunction

   assign temp_op = decode_key(row_delayed, o_b_op_keypad_column);
   assign valid_press = (o_b_op_keypad_column != 4'b1111 && row_delayed !=
4'b1111);

   always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin
      if (i_sys_reset) begin// Reset all states and outputs
         o_b_op_keycode <= 4'b1111;//default no operation key pressed
         o_b_op_valid_key_pressed <= 0;
         o_b_op_neg_flag <= 0;
      end

      else if(valid_press &&!key_pressed_flag) begin//start when key is pressed
         case (temp_op)
            4'b1100: begin//when the input is change sign function
               o_b_op_neg_flag <= 1;
               key_pressed_flag <= 0;
               o_b_op_keycode <= 4'b1111;//pass the decoded keycode to other blocks
               o_b_op_valid_key_pressed <= 0;
            end

            4'b1111: begin//when the input empty key
               o_b_op_neg_flag <= 0;
               key_pressed_flag <= 0;
               o_b_op_keycode <= 4'b1111;//pass the decoded keycode to other blocks
               o_b_op_valid_key_pressed <= 0;
            end

            default: begin//when the input is valid keys
               key_pressed_flag <= 1;
               o_b_op_keycode <= temp_op;//pass the decoded keycode to other blocks
               o_b_op_valid_key_pressed <= 1;//valid key
            end
         endcase
      end

      else if (i_b_op_hex_new_input)
```

```
            o_b_op_neg_flag <= 0;//clear the negative flag if new input is keyed in

        else if(!valid_press) begin
            key_pressed_flag <= 0;//release key
            o_b_op_valid_key_pressed <= 0;//only set high the flag for 1 clock cycle
        end
    end

endmodule
```

4.2.7 Test Plan

No.	Test Case	Description of test vector Generation	Expected Output	Status
1	No key pressed	1. Wait for o_b_op_keypad_column = 4'b1111 2. Set i_b_op_keypad_row = 4'b1111 for 1 clock cycle	o_b_op_keycode = 4'b1111, o_b_op_valid_key_pressed = 0, o_b_op_neg = 0	pass
2	Change sign button pressed	1. Wait for o_b_op_keypad_column = 4'b0111 2. Set i_b_op_keypad_row = 4'b1110 for 1 clock cycle	o_b_op_neg = 1, o_b_op_keycode = 4'b1111, o_b_op_valid_key_pressed = 0	pass
3	New input keyed in when negative flag is high	1. Wait for o_b_op_keypad_column = 4'b0111 2. Set i_b_op_keypad_row = 4'b1110 for 1 clock cycle 3. Set i_b_op_hex_new_input = 1 for 1 clock cycle	o_b_op_keycode = previous value, o_b_op_valid_key_pressed = previous value, o_b_op_neg = previous value	pass
4	"Add" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1110 2. Set i_b_op_keypad_row = 4'b1110 for 1 clock cycle	o_b_op_keycode = 4'b0000, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
5	"Minus" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1110 2. Set i_b_op_keypad_row = 4'b1101 for 1 clock cycle	o_b_op_keycode = 4'b0001, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
6	"Multiply" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1110 2. Set i_b_op_keypad_row = 4'b1011 for 1 clock cycle	o_b_op_keycode = 4'b0010, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
7	"Divide" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1110 2. Set i_b_op_keypad_row =	o_b_op_keycode = 4'b0011, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass

		4'b0111 for 1 clock cycle		
8	"NOT" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1101 2. Set i_b_op_keypad_row = 4'b1110 for 1 clock cycle	o_b_op_keycode = 4'b0100, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
9	"AND" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1101 2. Set i_b_op_keypad_row = 4'b1101 for 1 clock cycle	o_b_op_keycode = 4'b0101, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
10	"OR" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1101 2. Set i_b_op_keypad_row = 4'b1011 for 1 clock cycle	o_b_op_keycode = 4'b0110, o_b_op_valid_key_pressed = 1, o_b_op_neg = 0	pass
11	"XOR" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1101 2. Set i_b_op_keypad_row = 4'b0111 for 1 clock cycle	o_b_op_keycode = 4'b0111, o_b_op_valid_key_pressed = 1, o_b_op_neg = 0	pass
12	"Bitwise right shift" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1011 2. Set i_b_op_keypad_row = 4'b1110 for 1 clock cycle	o_b_op_keycode = 4'b1000, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
13	"Bitwise left shift" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1011 2. Set i_b_op_keypad_row = 4'b1101 for 1 clock cycle	o_b_op_keycode = 4'b1001, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
14	"Arithmetic right shift" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1011 2. Set i_b_op_keypad_row = 4'b1011 for 1 clock cycle	o_b_op_keycode = 4'b1010, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass
15	"Arithmetic left shift" operation chosen	1. Wait for o_b_op_keypad_column = 4'b1011 2. Set i_b_op_keypad_row =	o_b_op_keycode = 4'b1011, o_b_op_valid_key_pressed = 1, o_b_op_neg = previous value	pass

		4'b0111 for 1 clock cycle		
	Invalid operator input for empty key chosen	1. Wait for o_b_op_keypad_column = 4'b0111 2. Set i_b_op_keypad_row = 4'b1011 for 1 clock cycle	o_b_op_keycode = 4'b1111, o_b_op_valid_key_pressed = 0, o_b_op_neg = previous value	pass
17	Reset button pressed when operator is valid	1. Wait for o_b_op_keypad_column = 4'b1011 2. Set i_b_op_keypad_row = 4'b0111 for 1 clock cycle(set a random button pressed) 3. Set i_sys_reset = 1 for 1 clock cycle	o_b_op_keycode = 4'b1111, o_b_op_valid_key_pressed = 0, o_b_op_neg = 0	pass
18	Reset button pressed when negative flag is high	4. Wait for o_b_op_keypad_column = 4'b0111 5. Set i_b_op_keypad_row = 4'b1110 for 1 clock cycle 6. Set i_sys_reset = 1 for 1 clock cycle	o_b_op_keycode = 4'b1111, o_b_op_valid_key_pressed = 0, o_b_op_neg = 0	pass

Table 4.2.5: Test plan of operator keypad code generator

4.2.8 Testbench and Simulation results

4.2.8.1 Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Yu Heng
//
// Create Date: 20.04.2025 14:23:23
// File Name: tb_b_op.sv
// Module Name: tb_b_op
// Project Name: 8-bit integer calculator
// Code Type: Behavioural
// Description: Testbench for operator keypad code generator
//
//////////////////////////////////////////////////////////////////////////////////

module tb_b_op;

    logic       tb_i_sys_clock;
    logic       tb_i_sys_reset;
    logic [3:0] tb_i_b_op_keypad_row;
    logic       tb_i_b_op_hex_new_input;
    logic [3:0] tb_o_b_op_keypad_column;
    logic [3:0] tb_o_b_op_keycode;
    logic       tb_o_b_op_valid_key_pressed;
    logic       tb_o_b_op_neg_flag;

    // Instantiate the b_hex module
b_op
b_op_dut (
    .i_sys_clock(tb_i_sys_clock),
    .i_sys_reset(tb_i_sys_reset),
    .i_b_op_keypad_row(tb_i_b_op_keypad_row),
    .i_b_op_hex_new_input(tb_i_b_op_hex_new_input),
    .o_b_op_keycode(tb_o_b_op_keycode),
    .o_b_op_valid_key_pressed(tb_o_b_op_valid_key_pressed),
    .o_b_op_neg_flag(tb_o_b_op_neg_flag),
    .o_b_op_keypad_column(tb_o_b_op_keypad_column)
);


    // Clock generation
    always #5ns tb_i_sys_clock = ~tb_i_sys_clock; // 10ns period, 50MHz

    // Reset generation, changed to use tb_i_sys_reset

    // Testbench stimulus, changed to use tb_ signals
    // Reset generation
    initial begin
        tb_i_sys_clock = 1;
        tb_i_sys_reset = 1;
        #10;
        tb_i_sys_reset = 0;
    end

    initial begin
        // Initialize all input signals.
        tb_i_b_op_keypad_row = 4'b1111;
```

```
tb_o_b_op_keypad_column = 4'b1111;
tb_i_b_op_hex_new_input = 0;
//---------------------------------------
// Test Case 1: No key pressed
//---------------------------------------
#50; // Wait for a few clock cycles

//---------------------------------------
// Test Case 2: Change sign button pressed
//---------------------------------------
#30; // Wait for column to become 4'b0111
tb_i_b_op_keypad_row = 4'b1110;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//---------------------------------------
// Test Case 3: New input keyed in when negative flag is high
//---------------------------------------
#50
tb_i_b_op_hex_new_input = 1;//set new input flag to high for 1 clock cycle
#10
tb_i_b_op_hex_new_input = 0;
#40

//---------------------------------------
// Test Case 4: "Add" operation chosen
//---------------------------------------
#10; // Wait for column to become 4'b1110
tb_i_b_op_keypad_row = 4'b1110;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//---------------------------------------
// Test Case 5: "Minus" operation chosen
//---------------------------------------
#40; // Wait for column to become 4'b1110
tb_i_b_op_keypad_row = 4'b1101;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//---------------------------------------
// Test Case 6: "Multiply" operation chosen
//---------------------------------------
#40; // Wait for column to become 4'b1110
tb_i_b_op_keypad_row = 4'b1011;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//---------------------------------------
// Test Case 7: "Divide" operation chosen
//---------------------------------------
#40; // Wait for column to become 4'b1110
tb_i_b_op_keypad_row = 4'b0111;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//---------------------------------------
```

```
// Test Case 8: "NOT" operation chosen
//----------------------------------------
#50; // Wait for column to become 4'b1101
tb_i_b_op_keypad_row = 4'b1110;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//----------------------------------------
// Test Case 9: "AND" operation chosen
//----------------------------------------
#40; // Wait for column to become 4'b1101
tb_i_b_op_keypad_row = 4'b1101;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//----------------------------------------
// Test Case 10: "OR" operation chosen
//----------------------------------------
#40; // Wait for column to become 4'b1101
tb_i_b_op_keypad_row = 4'b1011;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the buttoncapture

//----------------------------------------
// Test Case 11: "XOR" operation chosen
//----------------------------------------
#40; // Wait for column to become 4'b1101
tb_i_b_op_keypad_row = 4'b0111;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//----------------------------------------
// Test Case 12: "Bitwise right shift" operation chosen
//----------------------------------------
#50; // Wait for column to become 4'b1011
tb_i_b_op_keypad_row = 4'b1110;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//----------------------------------------
// Test Case 13: "Bitwise left shift" operation chosen
//----------------------------------------
#40; // Wait for column to become 4'b1011
tb_i_b_op_keypad_row = 4'b1101;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//----------------------------------------
// Test Case 14: "Arithmetic right shift" operation chosen
//----------------------------------------
#40; // Wait for column to become 4'b1011
tb_i_b_op_keypad_row = 4'b1011;
#10; // Simulate holding the button for 1 clock cycle
tb_i_b_op_keypad_row = 4'b1111; // Release the button

//----------------------------------------
// Test Case 15: "Arithmetic left shift" operation chosen
```

```
        //----------------------------------------
        #40; // Wait for column to become 4'b1011
        tb_i_b_op_keypad_row = 4'b0111;
        #10; // Simulate holding the button for 1 clock cycle
        tb_i_b_op_keypad_row = 4'b1111; // Release the button

        //----------------------------------------
        // Test Case 16: Invalid operator input for empty key chosen
        //----------------------------------------
        #50; // Wait for column to become 4'b0111
        tb_i_b_op_keypad_row = 4'b1011;
        #10; // Simulate holding the button for 1 clock cycle
        tb_i_b_op_keypad_row = 4'b1111; // Release the button

        //----------------------------------------
        // Test Case 17: Reset button pressed when operator is valid
        //----------------------------------------
        #30; // Wait for column to become 4'b1011
        tb_i_b_op_keypad_row = 4'b0111;//set a random valid operator button pressed
        #10; // Simulate holding the button for 1 clock cycle
        tb_i_b_op_keypad_row = 4'b1111; // Release the button
        #50
        tb_i_sys_reset = 1;//Simulate reset button pressed
        #10
        tb_i_sys_reset = 0;

        //----------------------------------------
        // Test Case 18: Reset button pressed when negative flag is high
        //----------------------------------------
        #20; // Wait for column to become 4'b0111
        tb_i_b_op_keypad_row = 4'b1110;
        #10; // Simulate holding the button for 1 clock cycle
        tb_i_b_op_keypad_row = 4'b1111; // Release the button
        #50
        tb_i_sys_reset = 1;//Simulate reset button pressed
        #10
        tb_i_sys_reset = 0;

        #100

        $stop;
    end

endmodule
```

4.2.8.2 Simulation result

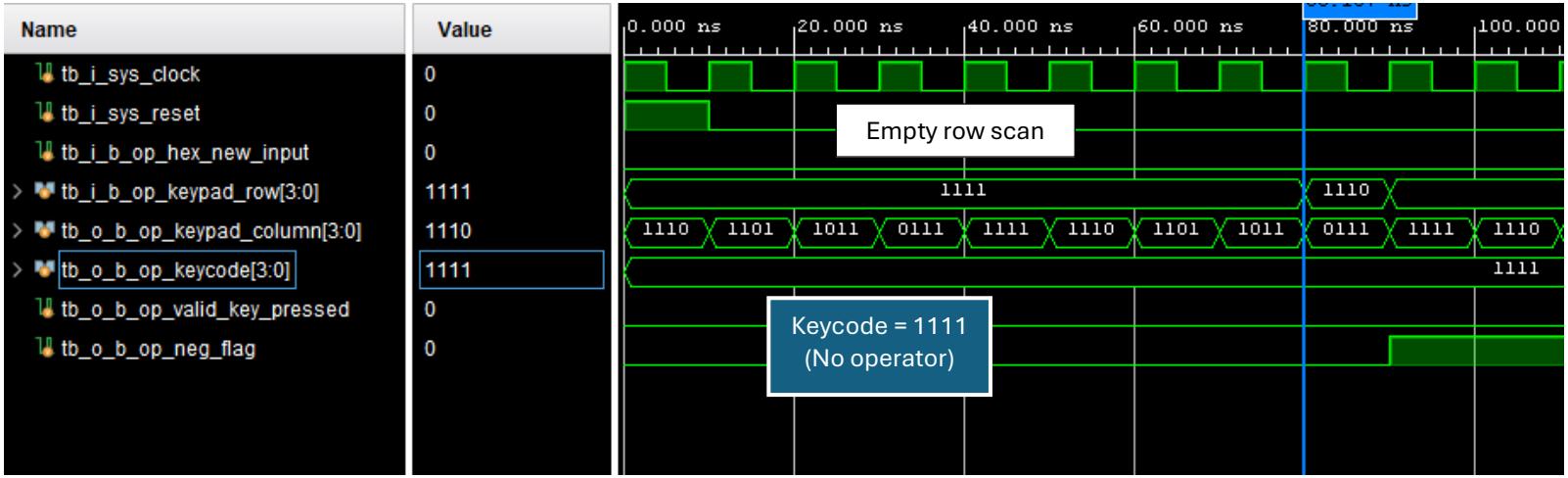

Figure 4.2.8.2.1 test case 1: No key pressed

v

test case 2: Change sign button pressed

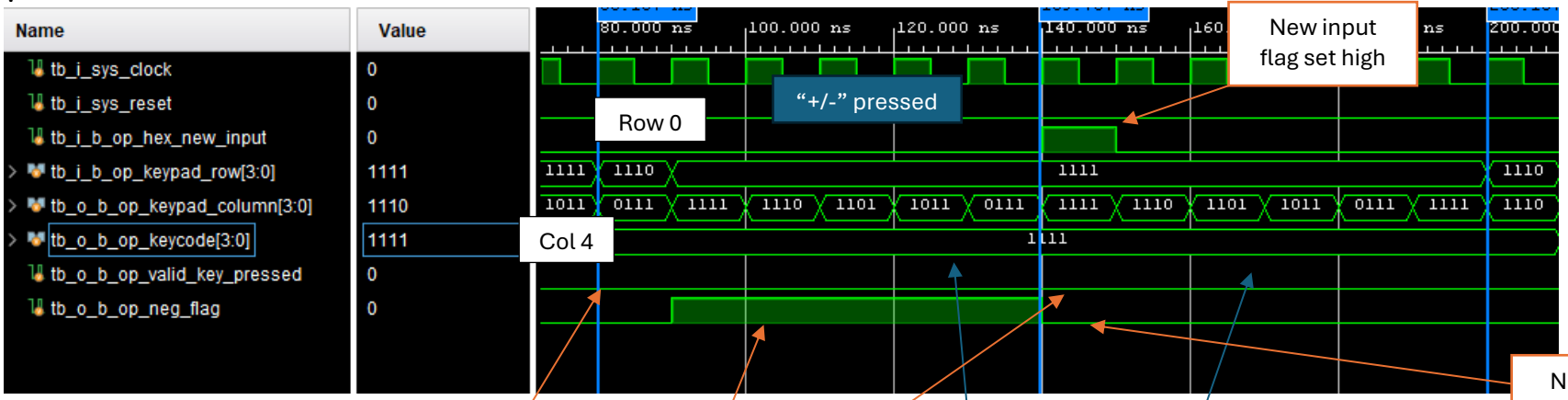

Figure 4.2.8.2.2 test case 2, 3

Negative flag becomes high

test case 3: New input keyed in when negative flag is high

Valid keypress flag remains low (invalid operator)

Keycode = 1111 (No operator)

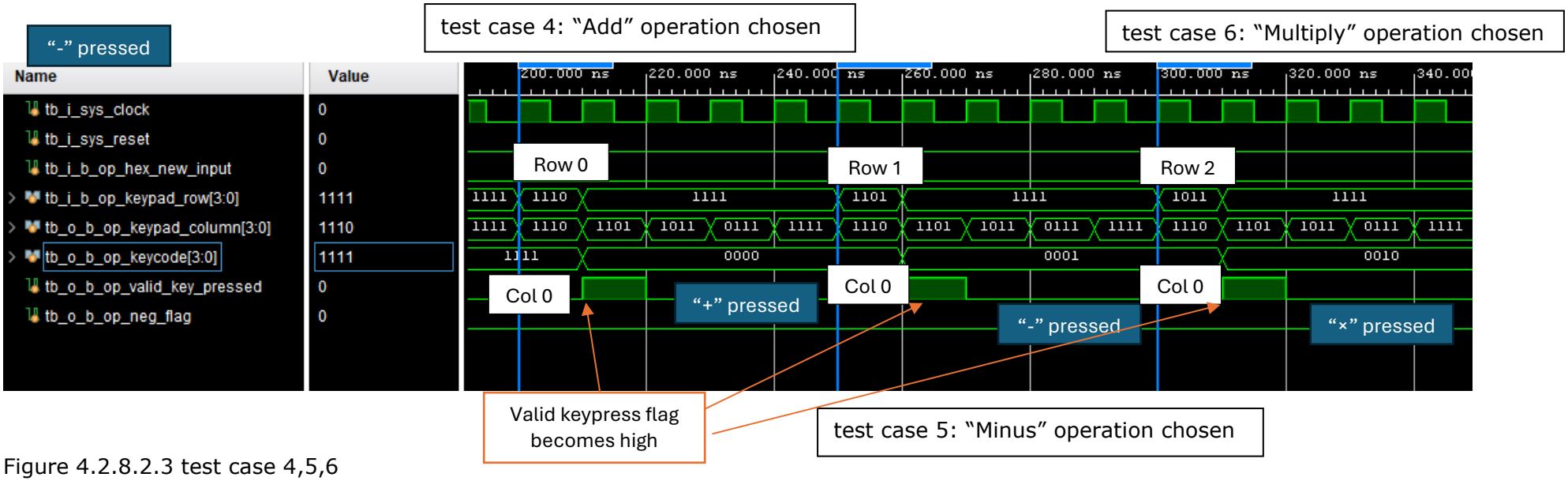

Figure 4.2.8.2.3 test case 4,5,6

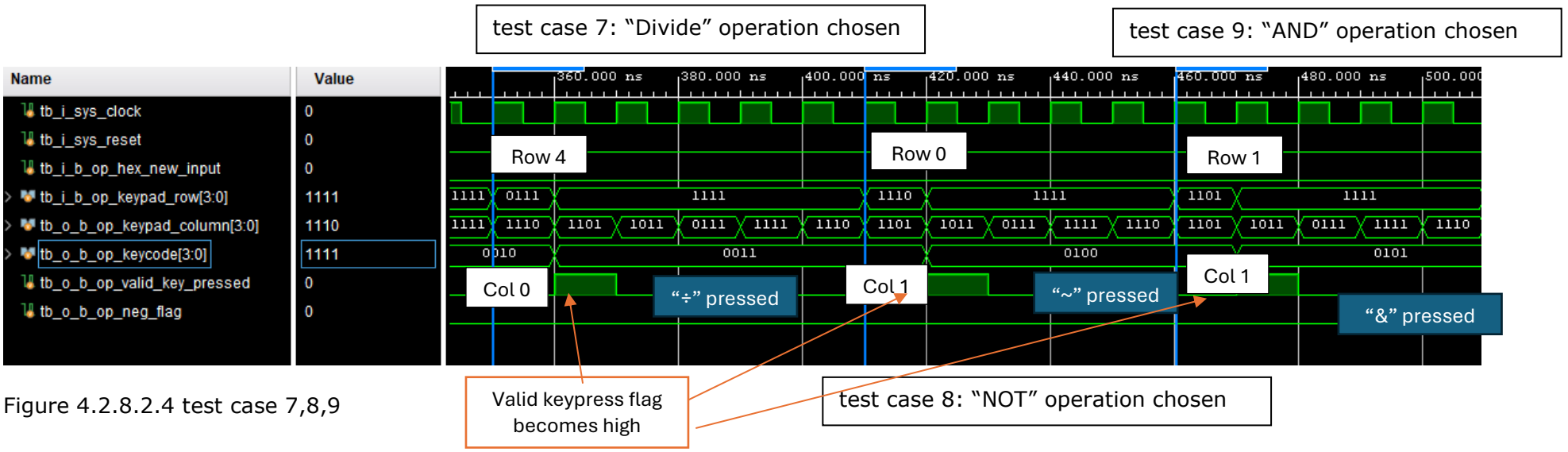

Figure 4.2.8.2.4 test case 7,8,9

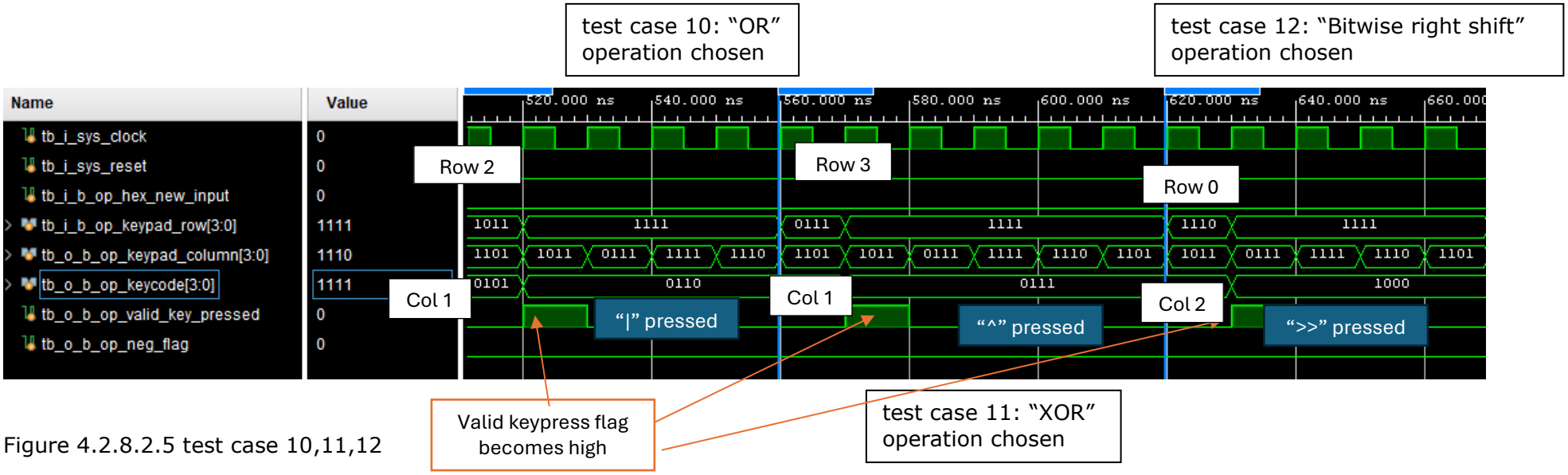

Figure 4.2.8.2.5 test case 10,11,12

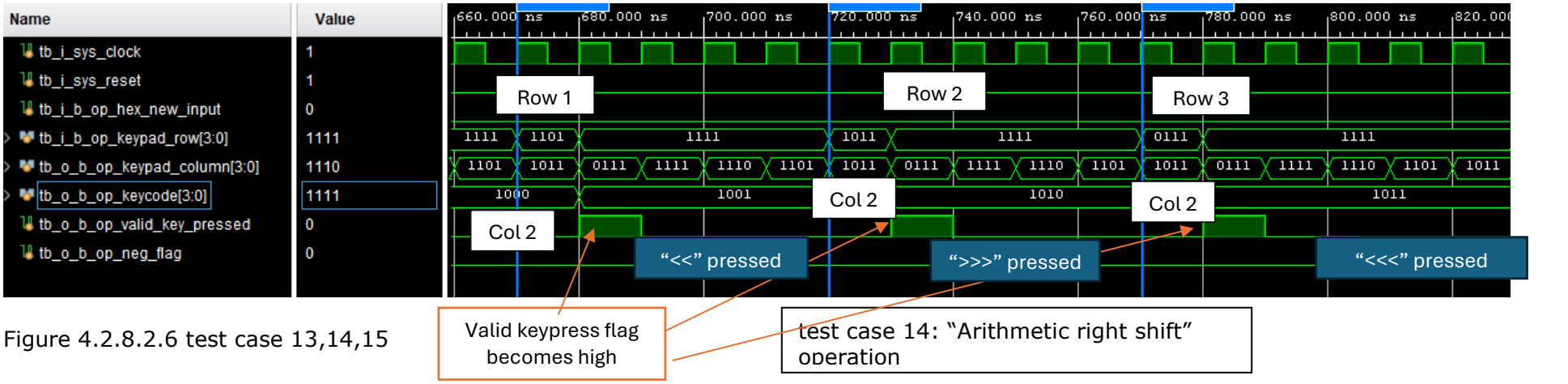

Figure 4.2.8.2.6 test case 13,14,15

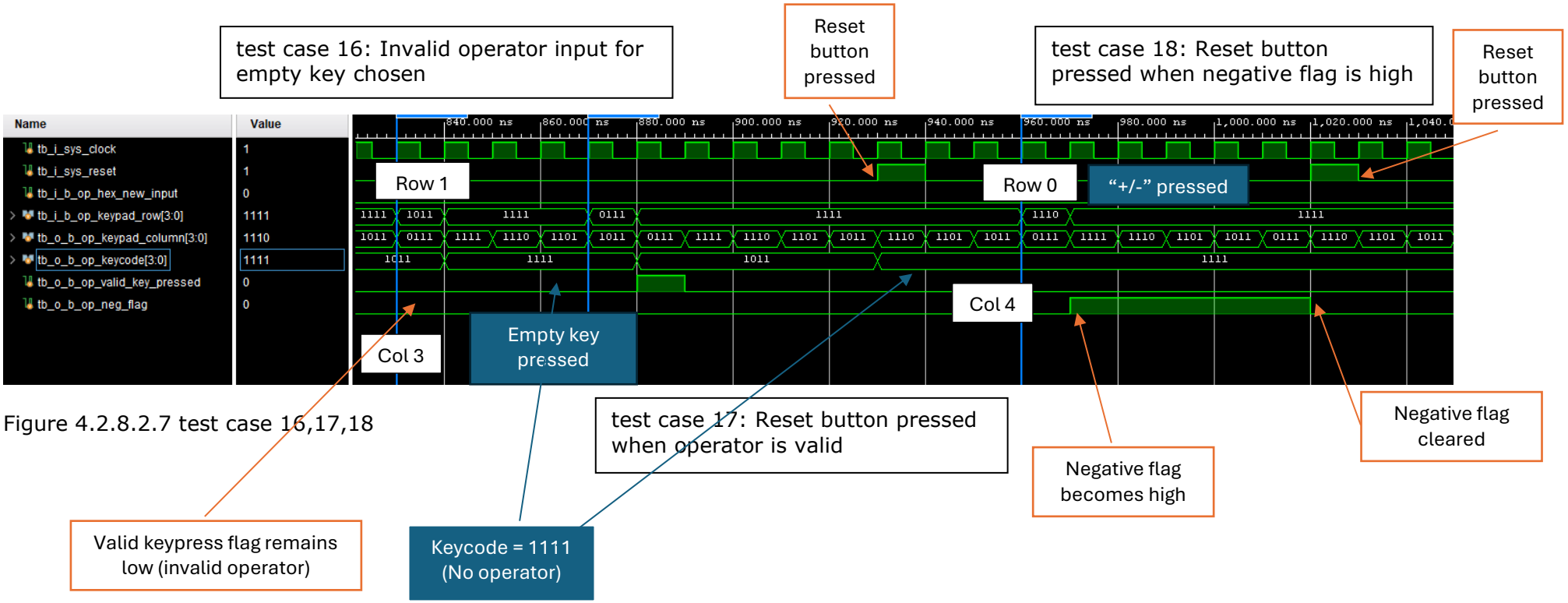

Figure 4.2.8.2.7 test case 16,17,18

4.3 Hex keypad code generator

4.3.1 Functionality/features

- Convert hexadecimal keypad row and column inputs into 8-bit hexadecimal value
- Indicate overflow if the input exceeded 8-bit.

4.3.2 Block interface and I/O pin description

4.3.2.1 Hex keypad code generator block interface

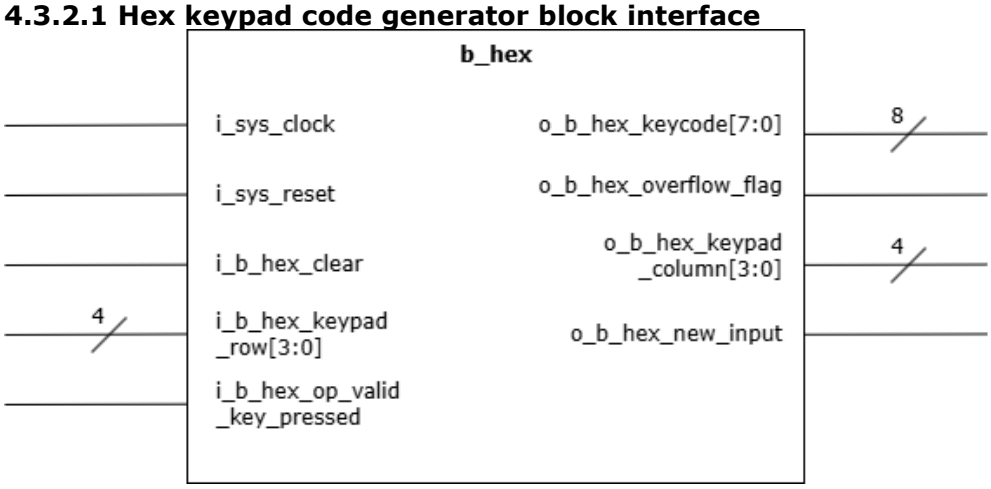

Diagram 4.3.2.1: Block interface of hex keypad code generator

4.3.2.2 I/O pin description

Pin name: **Pin class:** **Pin function:**	i_sys_clock control to provide a system clock to the calculator	**Source →Destination:**	Clock generator → Hex keypad code generator
Pin name: **Pin class:** **Pin function:**	i_sys_reset control to reset the calculator	**Source →Destination:**	ON/Reset button→ Hex keypad code generator
Pin name: **Pin class:** **Pin function:**	i_b_hex_clear control to clear current accumulated hexadecimal value and flags	**Source →Destination:**	Synchronizer →Hex keypad code generator
Pin name: **Pin class:** **Pin function:**	i_b_hex_keypad_row[3:0] data to detect which row of hex keypad is pressed	**Source →Destination:**	Synchronizer →Hex keypad code generator
Pin name: **Pin class:** **Pin function:**	i_b_hex_op_valid_key_pressed control to indicate valid operator key press to clear flags	**Source →Destination:**	Operator keypad code generator →Hex keypad code generator
Pin name: **Pin class:** **Pin function:**	o_b_hex_keycode[7:0] data to pass the 8-bit accumulated hexadecimal value in binary for operation later	**Source →Destination:**	Hex keypad code generator → ALU calculator & 7-segment

			display decoder
Pin name: **Pin class:** **Pin function:**	o_b_hex_overflow_flag control to pass overflow flag for error message display	**Source →Destination:**	Hex keypad code generator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	o_b_hex_keypad_column[3:0] data to detect which column of hex keypad is pressed	**Source →Destination:**	Hex keypad code generator → Hex keypad
Pin name: **Pin class:** **Pin function:**	o_b_hex_new_input control to inform other block when new input is keyed in	**Source →Destination:**	Hex keypad code generator → Operator keypad code generator

Table 4.3.2.2: I/O pin description of hex keypad code generator

4.3.3 Internal Operation: Function Table

Clear i_b_hex_clear	**Hex Keypad row i_b_hex_keypad_row [3:0]**	**Hex keypad column o_b_hex_keypad_column [3:0]**	**Hexadecimal Input Value In binary o_b_hex_keycode [7:0]**	**Invalidity flag o_b_hex_overflow**	**New Input Flag i_b_op_hex_new_input**	**Hex keypad code generator function**
1	4'bXXXX	4'bXXXX	8'b0	0	0	Reset input value to 0
0	Accumulated input within 8-bit		8-bit input value	0	1	Pass accumulated input value to other blocks when input is within 8-bit
0	Accumulated input over 8-bit		8'b0	1	1	Pass invalid flag to other block when input is more than 8-bit

Table 4.3.3: Function table of hex keypad code generator

4.3.4 Timing Requirement

Signal	**Requirement**
i_b_hex_keypad_row[3:0]	**More than 10 clock cycles** - Must be stable while corresponding column is scanning for 10 (5x2) clock cycles in a loop (real hardware implementation)

o_b_hex_keypad_column[3:0])	**More than 2 clock cycles** - Must scan each column for ≥ 2 clock cycles to allow row line to settle
i_b_hex_clear i_b_hex_op_valid_key_pressed i_b_op_hex_new_input	**1 clock cycle** -to ensure the flag is read
o_b_hex_keycode[7:0]	**1 clock cycle** -to pass the input data to alu calculator and 7 segment display decoder into register
o_b_hex_overflow_flag	**2 clock cycle** -to ensure other blocks read the flag

Table 4.3.4: Timing requirements of hex keypad code generator

4.3.5 Schematic Diagram

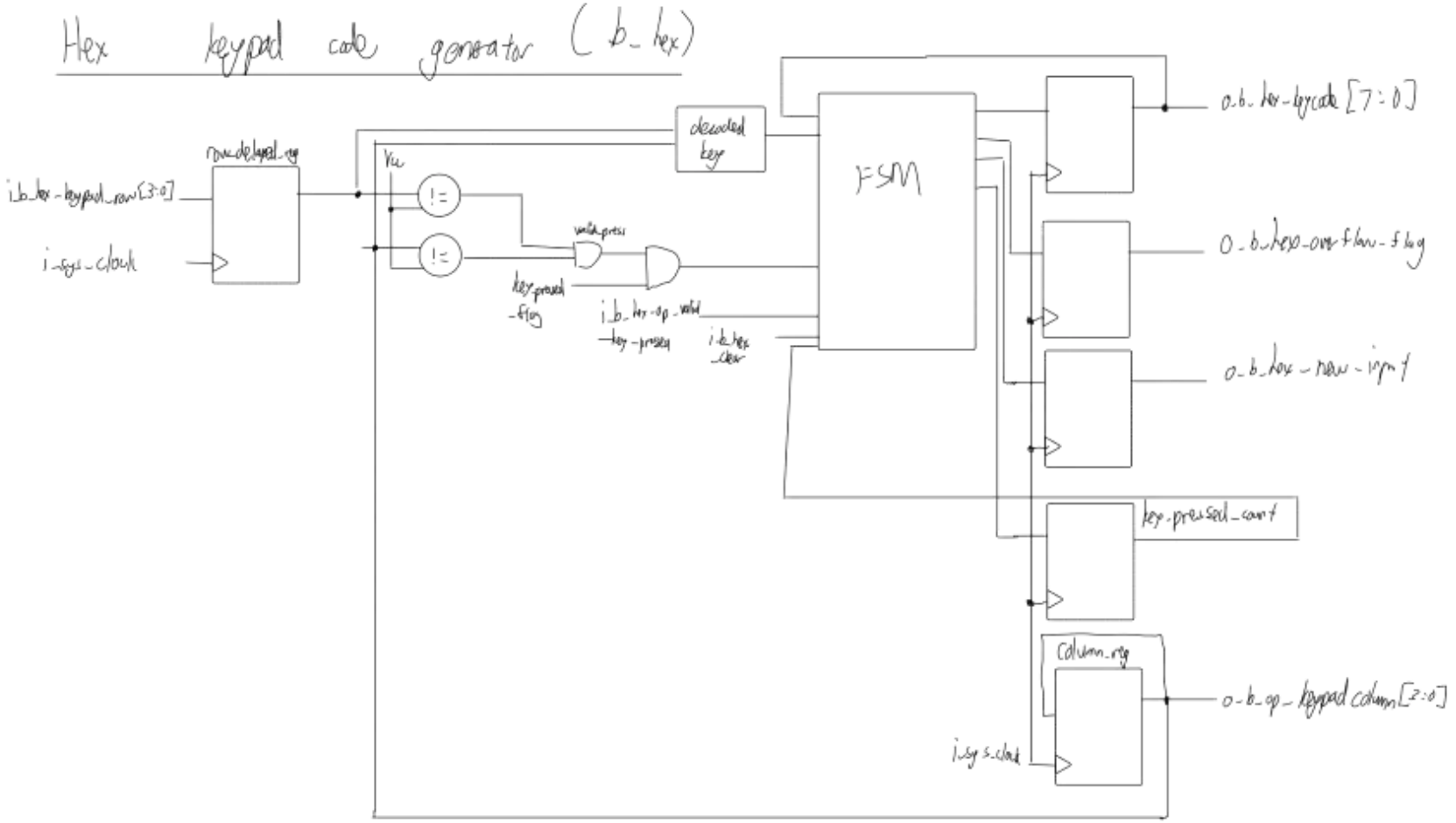

Figure 4.3.5.1 Microarchitecture of hex keypad code generator

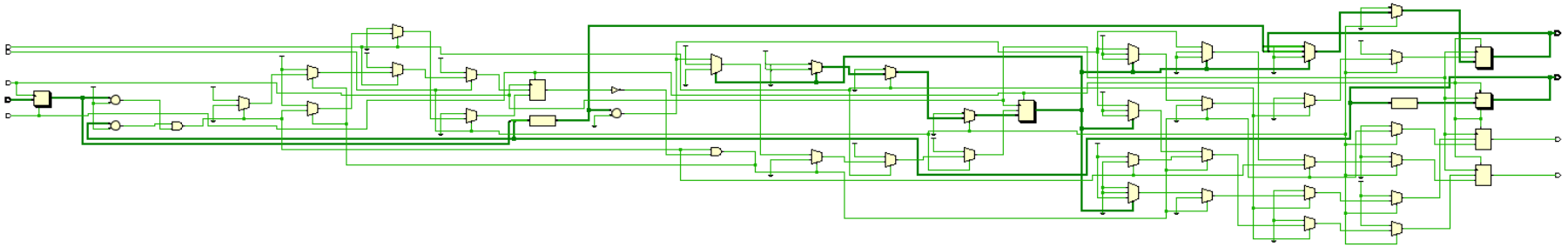

Figure 4.3.5.2.1 Overview of schematic diagram

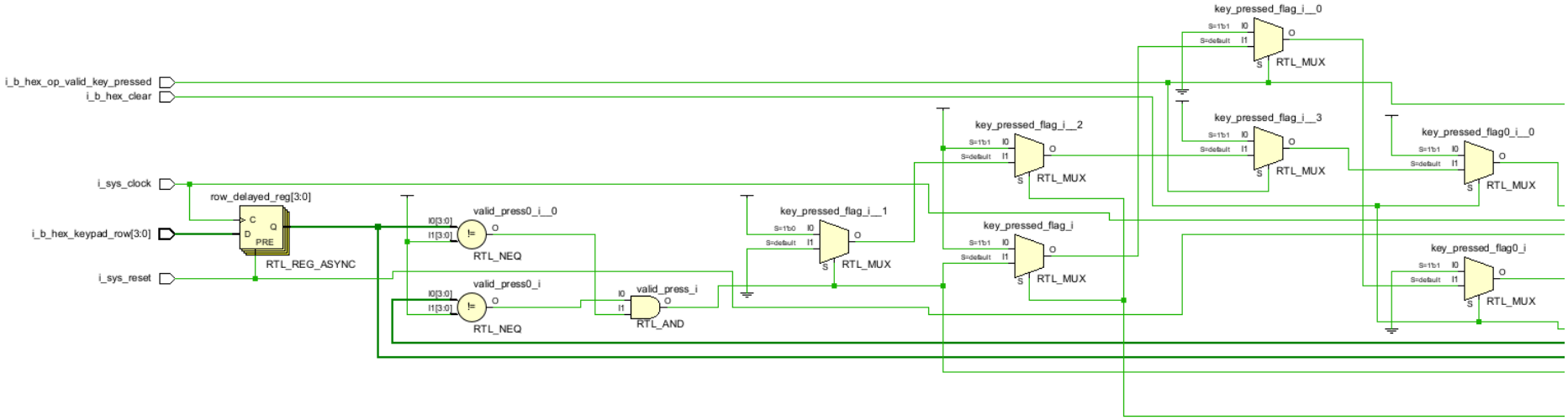

Figure 4.3.5.2.2 Parts of schematic diagram 1

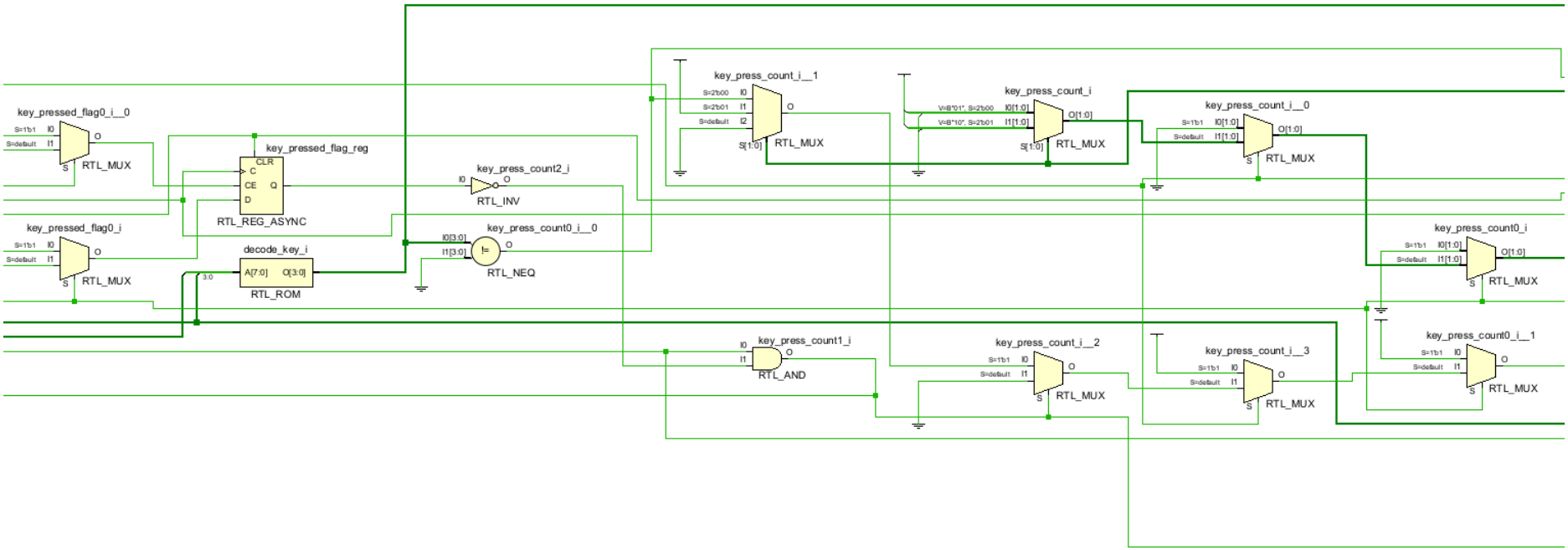

Figure 4.3.5.2.3 Parts of schematic diagram 2

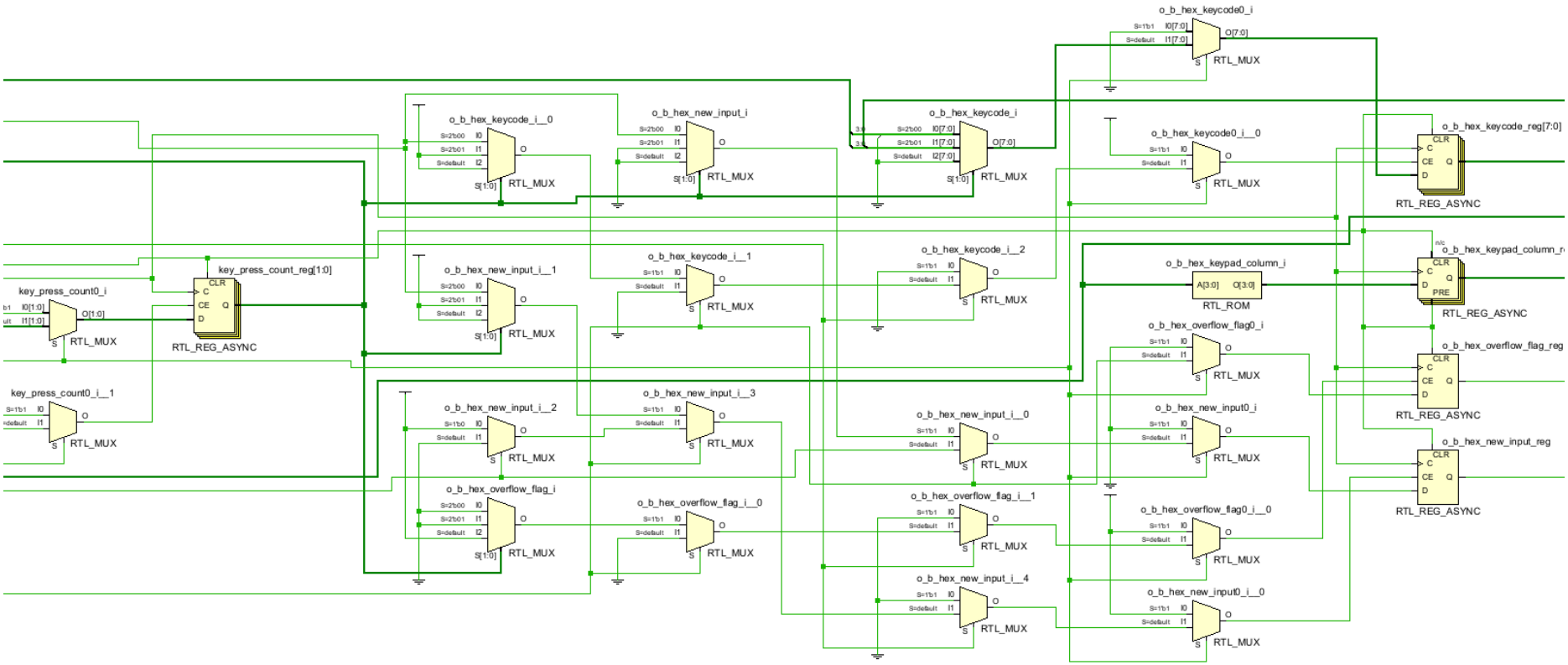

Figure 4.3.5.2.4 Parts of schematic diagram 3

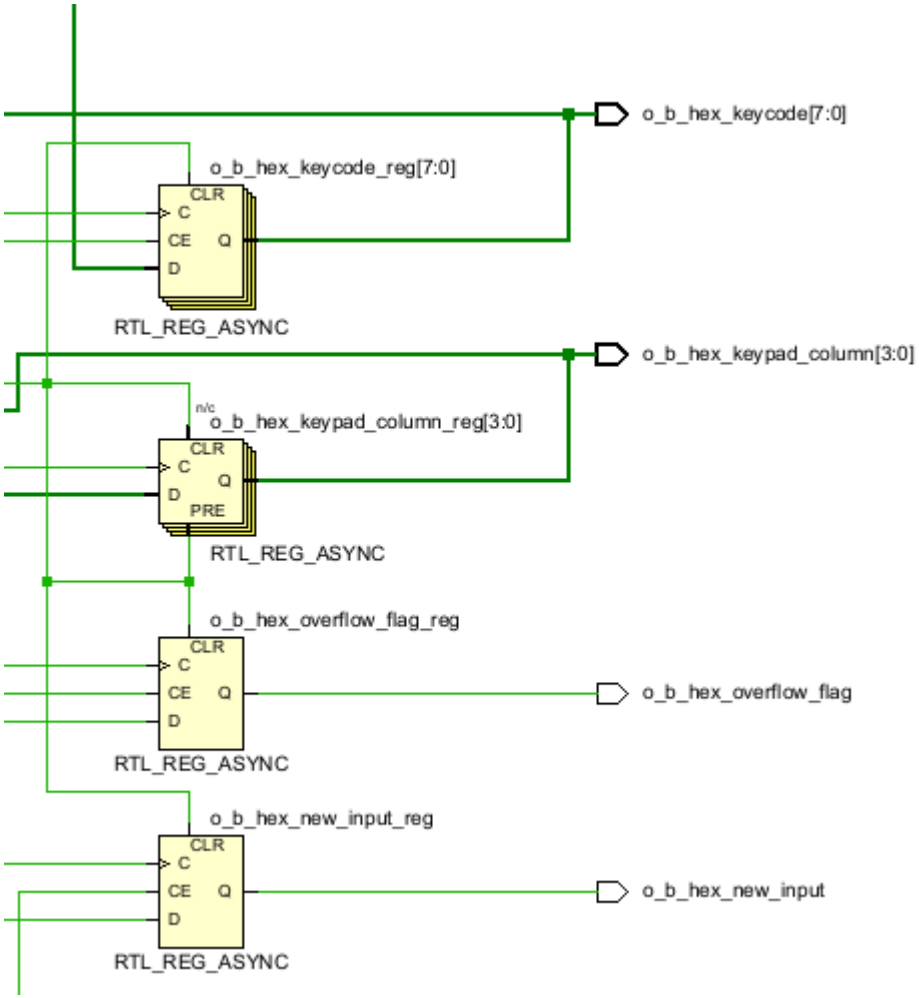

Figure 4.3.5.2.5 Parts of schematic diagram 4

4.3.6 System Verilog Model

```
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Yu Heng
//
// Create Date: 19.04.2025 22:54:00
// File Name: b_hex.sv
// Module Name: b_hex
// Project Name: 8-bit integer calculator
// Code Type: RTL level
// Description: Modelling of hex keypad code generator
//
//////////////////////////////////////////////////////////////////////////////////

module b_hex (
    input  logic       i_sys_clock,
    input  logic       i_sys_reset,
    input  logic [3:0]  i_b_hex_keypad_row,
    input  logic       i_b_hex_clear,
    input  logic       i_b_hex_op_valid_key_pressed,
    output logic [3:0]  o_b_hex_keypad_column,
    output logic [7:0]  o_b_hex_keycode,
    output logic       o_b_hex_overflow_flag,
    output logic       o_b_hex_new_input
);

    logic [1:0] key_press_count;//to indicate key press count
    logic [3:0] temp_hex;//register to store readed key in value
    logic key_pressed_flag;//to indicate if the button is pressed
    logic valid_press;//to indicate valid key press
    logic [3:0] row_delayed;      // delay row by one clock cycle to synchronize wave
output with the column

    // Delay the Row signal by one clock cycle
    always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin
        if (i_sys_reset)
            row_delayed <= 4'b1111; // idle state (all unpressed)
        else
            row_delayed <= i_b_hex_keypad_row;
    end


    // Drive the column cycling
    always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin//loop to scan the
column keypad
        if (i_sys_reset)
        o_b_hex_keypad_column <= 4'b1110; // Start with column 0 active
        else begin
            case (o_b_hex_keypad_column)
                4'b1110: o_b_hex_keypad_column <= 4'b1101; // col 1
                4'b1101: o_b_hex_keypad_column <= 4'b1011; // col 2
                4'b1011: o_b_hex_keypad_column <= 4'b0111; // col 3
                4'b0111: o_b_hex_keypad_column <= 4'b1111; // idle
                default: o_b_hex_keypad_column <= 4'b1110; // loop back to col 0
            endcase
        end
    end
```

```
   function logic [3:0] decode_key (input logic [3:0] row, input logic [3:0] col);
   //function to convert row and column read into hex decoded key
      case ({row, col})
         {4'b1110, 4'b1110}: decode_key = 4'h0;
         {4'b1110, 4'b1101}: decode_key = 4'h1;
         {4'b1110, 4'b1011}: decode_key = 4'h2;
         {4'b1110, 4'b0111}: decode_key = 4'h3;
         {4'b1101, 4'b1110}: decode_key = 4'h4;
         {4'b1101, 4'b1101}: decode_key = 4'h5;
         {4'b1101, 4'b1011}: decode_key = 4'h6;
         {4'b1101, 4'b0111}: decode_key = 4'h7;
         {4'b1011, 4'b1110}: decode_key = 4'h8;
         {4'b1011, 4'b1101}: decode_key = 4'h9;
         {4'b1011, 4'b1011}: decode_key = 4'hA;
         {4'b1011, 4'b0111}: decode_key = 4'hB;
         {4'b0111, 4'b1110}: decode_key = 4'hC;
         {4'b0111, 4'b1101}: decode_key = 4'hD;
         {4'b0111, 4'b1011}: decode_key = 4'hE;
         {4'b0111, 4'b0111}: decode_key = 4'hF;
         default:        decode_key = 4'h0;//input=0
      endcase
   endfunction

   assign temp_hex = decode_key(row_delayed, o_b_hex_keypad_column);
   assign valid_press = (o_b_hex_keypad_column != 4'b1111 && row_delayed !=
4'b1111);

   always_ff @(posedge i_sys_clock or posedge i_sys_reset) begin
      if (i_sys_reset||i_b_hex_clear) begin// Reset all states and outputs
         key_press_count <= 0;
         o_b_hex_keycode <= 0;
         o_b_hex_overflow_flag <= 0;
         key_pressed_flag <= 0;
         o_b_hex_new_input <= 0;
      end

      else if(i_b_hex_op_valid_key_pressed)begin
         key_press_count <= 2'd0;
         key_pressed_flag <= 0;
      end

      else if(valid_press &&!key_pressed_flag) begin//start when key is pressed
         key_pressed_flag<=1;

         case(key_press_count)
         2'b00: begin
            if (temp_hex!=4'h0)begin//remain input=0 if continuous 0 pressed
               o_b_hex_keycode<={4'h0,temp_hex};
               key_press_count<= 2'd1;
               o_b_hex_new_input <= 1;//set high a new input flag to clear negative flag
of operator
            end
         end
         2'b01: begin//1 key is pressed
            o_b_hex_keycode<={o_b_hex_keycode[3:0],temp_hex};//push the 1st digit
to most significant digit
```

```
            key_press_count<= 2'd2;
            o_b_hex_new_input <= 0;
         end
         default: begin
            o_b_hex_keycode <= 8'b0;
            o_b_hex_overflow_flag <= 1;
            o_b_hex_new_input <= 0;
         end
         endcase
      end
      else if(!valid_press) begin
         key_pressed_flag <= 0;
         o_b_hex_new_input <= 0;
      end
   end

endmodule
```

4.3.7 Test Plan

No.	Test Case	Description of test vector Generation	Expected Output	Status
1	Clear Input to Zero	1. Wait for o_b_hex_keypad_column to 4'b1101 2. Set i_b_hex_keypad_row to 4'b1011 for 1 clock cycle 3. Wait for o_b_hex_keypad_column = 4'b1011 4. Set i_b_hex_keypad_row = 4'b1110 for 1 clock cycle 5. Set i_b_hex_op_valid_key_pressed = 1 6. Set i_b_hex_clear = 1 for 1 clock cycle	o_b_hex_keycode = 8'b0, o_b_hex_overflow_flag = 0 o_b_hex_new_input = 1 (1 clock cycle when first input) o_b_hex_new_input = 0	pass
2	Use clear button to clear Input and flags when input overflow	1. Wait for o_b_hex_keypad_column = 4'b1011 2. Set i_b_hex_keypad_row = 4'b1011 for 1 clock cycle 3. Wait for next o_b_hex_keypad_column = 4'b1011 4. Set i_b_hex_keypad_row = 4'b1011 for 1 clock cycle 5. Wait again for o_b_hex_keypad_column = 4'b1011 6. Set i_b_hex_keypad_row = 4'b1011 for 1 clock cycle 7. Set i_b_hex_clear = 1 for 1 clock cycle	o_b_hex_keycode = 8'b0, o_b_hex_overflow_flag = 1 o_b_hex_new_input = 1 (1 clock cycle when first input) o_b_hex_new_input = 0	pass
3	Overflow on 3rd Hex Input	1. Wait for o_b_hex_keypad_column = 4'b0111 2. Set i_b_hex_keypad_row = 4'b0111 for 1 clock cycle (First input = F) 3. Wait for o_b_hex_keypad_column = 4'b1101 4. Set i_b_hex_keypad_row = 4'b1101 for 1 clock cycle (Second input = 5)	o_b_hex_keycode = 8'b00000000, o_b_hex_overflow_flag = 1 o_b_hex_new_input = 1 (1 clock cycle when first input) o_b_hex_new_input = 0	pass

		5. Wait for o_b_hex_keypad_column = 4'b1011 6. Set i_b_hex_keypad_row = 4'b1101 for 1 clock cycle (third input=6) (accumulated input=F56, overflow)		
4	Valid Accumulated 8-bit Input	1. Reset the input 2. Set i_b_hex_clear = 0 3. Wait for o_b_hex_keypad_column = 4'b0111 4. Set i_b_hex_keypad_row = 4'b0111 for 1 clock cycle (First input = F) 5. Wait for o_b_hex_keypad_column = 4'b1101 6. Set i_b_hex_keypad_row = 4'b1101 for 1 clock cycle (Second input = 5) 7. Set i_b_hex_op_valid_key_pressed = 1 (input=F5, no overflow)	o_b_hex_keycode = 8'b11110101, o_b_hex_overflow_flag = 0 o_b_hex_new_input = 1 (1 clock cycle when first input) o_b_hex_new_input = 0	pass
5	Second input after operator button is pressed	1. Set i_b_hex_op_valid_key_pressed = 1 for 1 clock cycle (remain previous input without clear or reset) 2. Wait for o_b_hex_keypad_column = 4'b1101 3. Set i_b_hex_keypad_row = 4'b1011 for 1 clock cycle (third input=9) 4. Wait for o_b_hex_keypad_column = 4'b1101 5. Set i_b_hex_keypad_row = 4'b1101 for 1 clock cycle (Second input = 5)	o_b_hex_keycode = 8'b10010101, o_b_hex_overflow_flag = 0 o_b_hex_new_input = 1 (1 clock cycle when first input) o_b_hex_new_input = 0	pass
6	Reset all input and flags using asynchronous reset button	1. Set i_sys_reset = 1 for 1 clock cycle	o_b_hex_keycode = 8'b00000000, o_b_hex_overflow_flag = 0 o_b_hex_new_input = 1 (1 clock cycle when first input) o_b_hex_new_input = 0	pass

Table 4.3.5: Test plan of hex keypad code generator

4.3.8 Testbench and Simulation results

4.3.8.1 Testbench

```
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Yu Heng
//
// Create Date: 20.04.2025 00:48:59
// File Name: tb_b_hex.sv
// Module Name: tb_b_hex
// Project Name: 8-bit integer calculator
// Code Type: Behavioural
// Description: Testbench for hex keypad code generator
//
//////////////////////////////////////////////////////////////////////////////////

module tb_b_hex;
  // Signals for the b_hex module
    logic tb_i_sys_clock;
    logic tb_i_sys_reset;
    logic [3:0] tb_i_b_hex_keypad_row;
    logic tb_i_b_hex_clear;
    logic tb_i_b_hex_op_valid_key_pressed;
    logic [3:0] tb_o_b_hex_keypad_column;
    logic [7:0] tb_o_b_hex_keycode;
    logic tb_o_b_hex_overflow_flag;
    logic tb_o_b_hex_new_input;

  // Instantiate the b_hex module
b_hex dut (
    .i_sys_clock(tb_i_sys_clock),
    .i_sys_reset(tb_i_sys_reset),
    .i_b_hex_keypad_row(tb_i_b_hex_keypad_row),
    .i_b_hex_clear(tb_i_b_hex_clear),
    .i_b_hex_op_valid_key_pressed(tb_i_b_hex_op_valid_key_pressed),
    .o_b_hex_keypad_column(tb_o_b_hex_keypad_column),
    .o_b_hex_keycode(tb_o_b_hex_keycode),
    .o_b_hex_overflow_flag(tb_o_b_hex_overflow_flag),
    .o_b_hex_new_input(tb_o_b_hex_new_input)
);
  // Clock generation
  always #5ns tb_i_sys_clock = ~tb_i_sys_clock; // 10ns period, 50MHz

  // Reset generation, changed to use tb_i_sys_reset

  // Testbench stimulus, changed to use tb_ signals
// Reset generation
  initial begin
    tb_i_sys_clock = 1;
    tb_i_sys_reset = 1;
    #10;
    tb_i_sys_reset = 0;
  end

  // Testbench stimulus
  initial begin
    // Initialize all input signals.
    tb_i_b_hex_keypad_row = 4'b1111;
```

```
tb_o_b_hex_keypad_column=4'b1111;
tb_i_b_hex_clear = 0;
tb_i_b_hex_op_valid_key_pressed = 0;

//----------------------------------------
// Test Case 1: Clear Input to Zero
//----------------------------------------
#20;
tb_i_b_hex_keypad_row = 4'b1011; // Key '9'
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#50;
tb_i_b_hex_keypad_row = 4'b1110;// Key '1'
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#50
tb_i_b_hex_clear = 1;
#10;
tb_i_b_hex_clear = 0;//reset

//----------------------------------------
// Test Case 2: Use clear to reset Input and flags to Zero when input overflow
//----------------------------------------
#20;
tb_i_b_hex_keypad_row = 4'b1011; // First press
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#40;
tb_i_b_hex_keypad_row = 4'b1011; // Second press
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#40;
tb_i_b_hex_keypad_row = 4'b1011; // Third press (causing overflow)
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#50
tb_i_b_hex_clear = 1;
#10
tb_i_b_hex_clear = 0;

//----------------------------------------
// Test Case 3: Overflow on 3rd Hex Input (Input F59)
//----------------------------------------
#40
tb_i_b_hex_keypad_row = 4'b0111; // 'F'
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#70;
tb_i_b_hex_keypad_row = 4'b1101; // '5'
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#40;
tb_i_b_hex_keypad_row = 4'b1011; // '6' -> Overflow
#10;
tb_i_b_hex_keypad_row = 4'b1111;//reset
#40;
```

```
    //----------------------------------------
    tb_i_b_hex_clear = 1;//reset accumulated input
    #10//wait
    tb_i_b_hex_clear = 0;

    //----------------------------------------
    // Test Case 4: Valid Accumulated 8-bit Input (Input F5)
    //----------------------------------------
    #10
    tb_i_b_hex_keypad_row = 4'b0111; // 'F'
    #10;
    tb_i_b_hex_keypad_row = 4'b1111;//reset
    #70;
    tb_i_b_hex_keypad_row = 4'b1101; // '5'
    #10;
    tb_i_b_hex_keypad_row = 4'b1111;//reset
    #50

    //----------------------------------------
    #50;//wait
    //----------------------------------------
    // Test Case 5: Second input after operator button is pressed
    //----------------------------------------
    //not resetting previous input
    tb_i_b_hex_op_valid_key_pressed = 1;//set to high simulating operator button
pressed to reset input count flag
    #10
    tb_i_b_hex_op_valid_key_pressed = 0;
    #30
    tb_i_b_hex_keypad_row = 4'b1011; // '9'
    #10;
    tb_i_b_hex_keypad_row = 4'b1111;//reset
    #40;
    tb_i_b_hex_keypad_row = 4'b1101; // '5'
    #10;
    tb_i_b_hex_keypad_row = 4'b1111;//reset
    #30;


    //----------------------------------------
    #50//wait
    //----------------------------------------
    // Test Case 5: Second input after operator button is pressed
    //----------------------------------------
    //not resetting previous input
    tb_i_sys_reset = 1;
    #10
    tb_i_sys_reset = 0;

    #100

    $stop;  // Keep this, it stops the simulation after the stimulus.

  end

endmodule
```

4.3.8.2 Simulation result

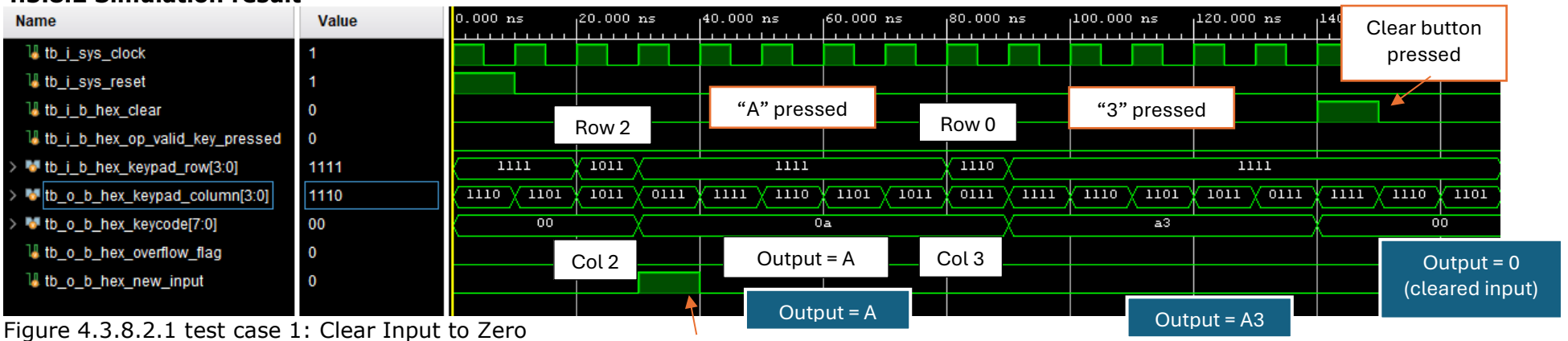

Figure 4.3.8.2.1 test case 1: Clear Input to Zero

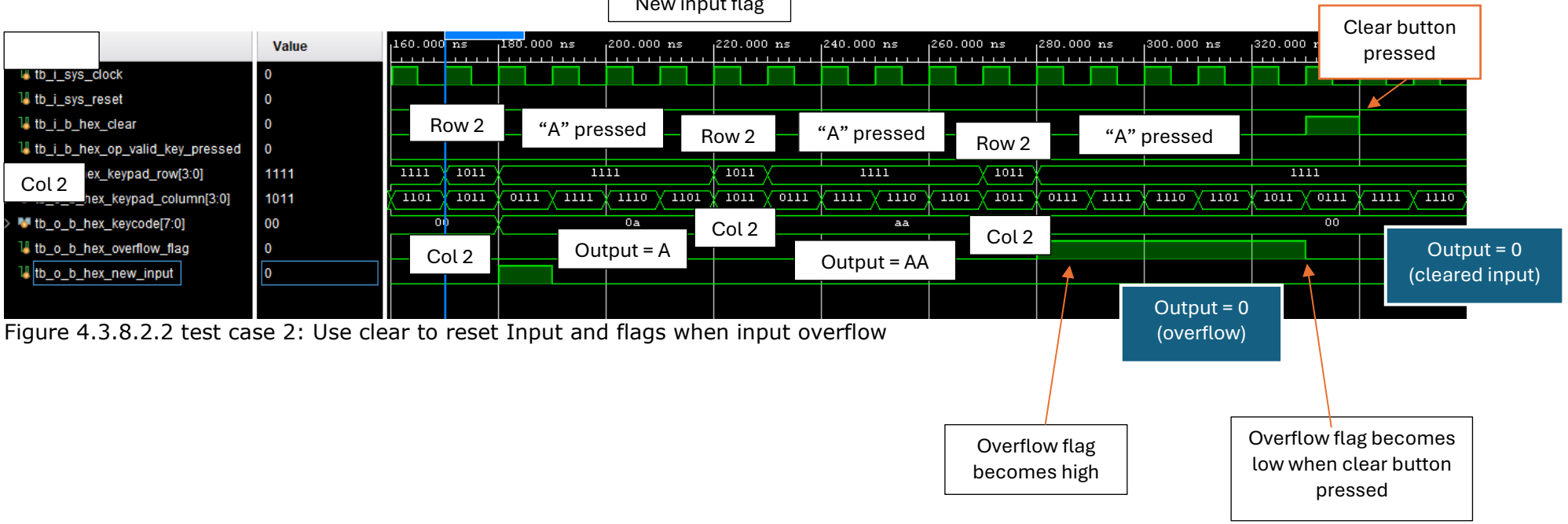

Figure 4.3.8.2.2 test case 2: Use clear to reset Input and flags when input overflow

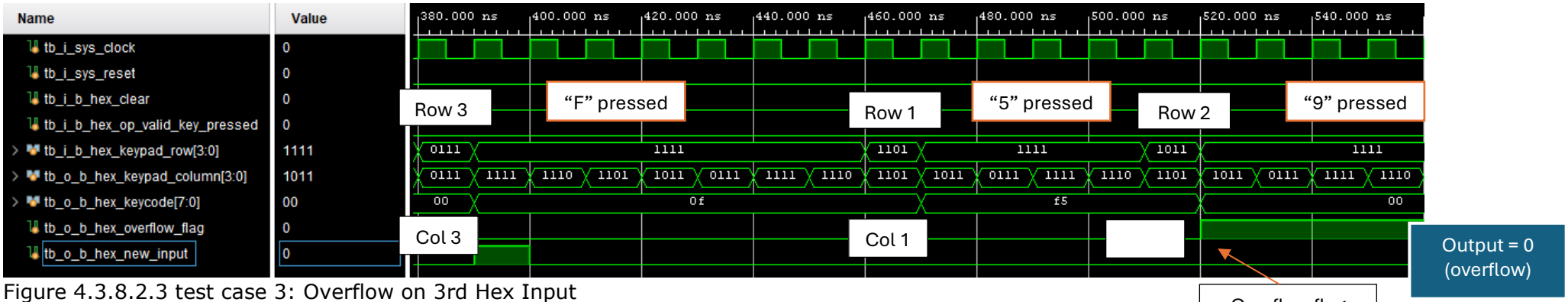

Figure 4.3.8.2.3 test case 3: Overflow on 3rd Hex Input

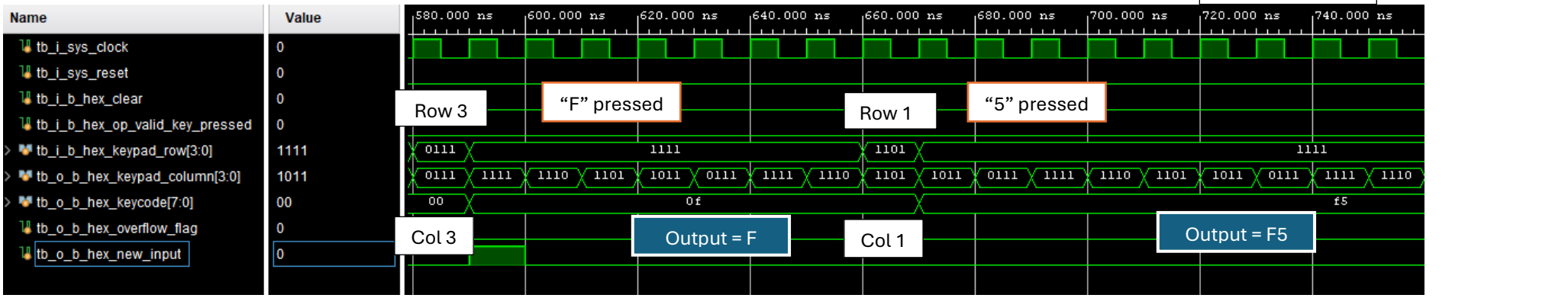

Figure 4.3.8.2.4 test case 4: Valid Accumulated 8-bit Input

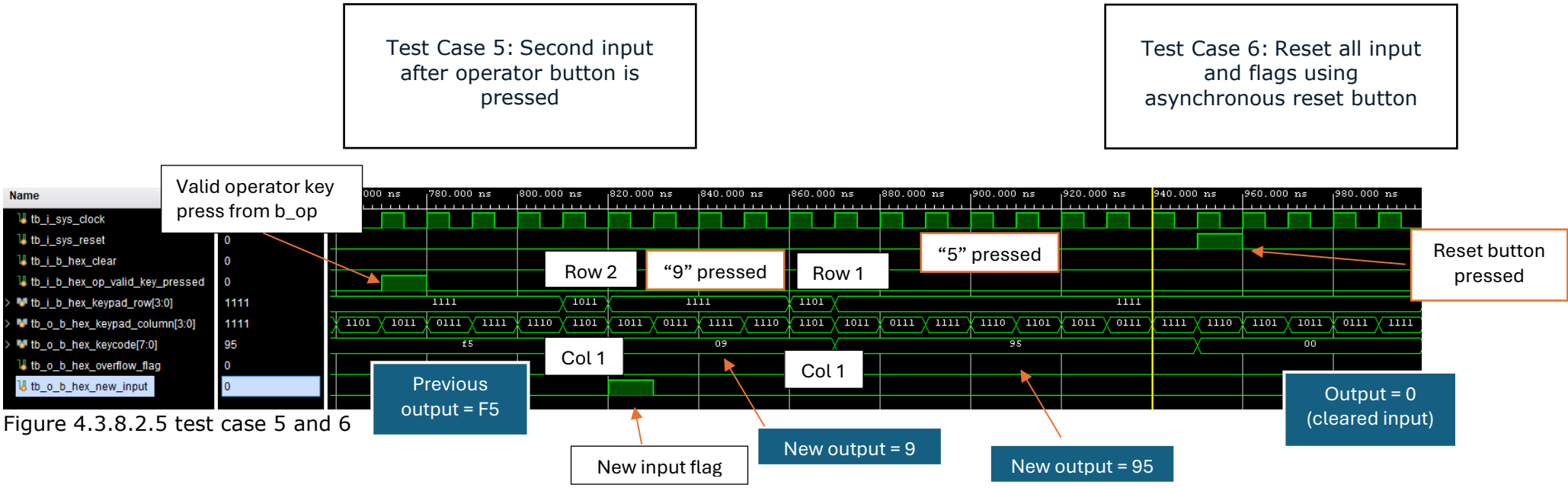

Figure 4.3.8.2.5 test case 5 and 6

4.4 ALU calculator

4.4.1 Functionality / features

- Doing arithmetic calculations such as + - x ÷
- Doing bitwise logic calculations including invert, and, or, exclusive or
- It will force all the input become
- Doing shifting such as logic swift and arithmetic swift
- It gives an overflow flag and sign flag

4.4.2 Block Interface and I/O Pin Description

4.4.2.1 block interface

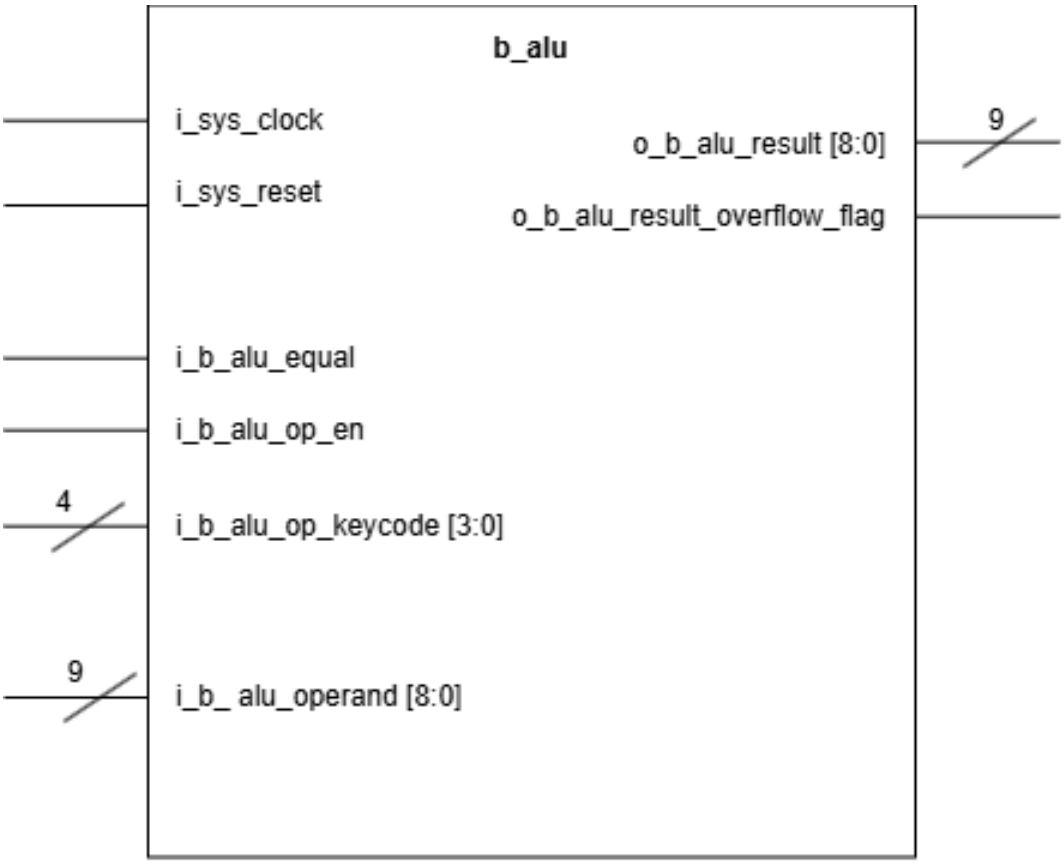

Diagram 4.3.2.1: Block interface of alu calculator

4.4.2.2 I/O pin description

Pin name: **Pin class:** **Pin function:**	i_sys_clock control provide a clock signal to the	**Source →Destination:**	Clock generator→ ALU calculator
Pin name: **Pin class:** **Pin function:**	I_sys_reset Control to reset the alu block	**Source →Destination:**	ON/Reset button→ ALU calculator
Pin name: **Pin class:** **Pin function:**	i_b_alu_equal control to trigger the calculation	**Source →Destination:**	Hex keypad generator→ ALU calculator
Pin name: **Pin class:** **Pin function:**	i_b_alu_op_en control to trigger the alu block	**Source →Destination:**	Operator keypad code generator→ ALU calculator
Pin name: **Pin class:** **Pin function:**	i_b_alu_op_keycode[3:0] control to select which algorithm to run	**Source →Destination:**	Operator keypad code generator→ ALU calculator
Pin name: **Pin class:** **Pin function:**	i_b_alu_operand[8:0] data used to doing arithmetic and logic calculation	**Source →Destination:**	r_result → ALU calculator
Pin name: **Pin class:** **Pin function:**	o_b_alu_result[8:0] data the result of the alu and store to a register	**Source →Destination:**	ALU calculator → 7 segment display decoder
Pin name: **Pin class:** **Pin function:**	o_b_alu_overflow_flag control to give err signal to 7-segment display	**Source →Destination:**	ALU calculator → 7- segment display decoder

Table 4.4.2: I/O pin description of alu calculator

4.4.3 Internal Operation: Function Table

a: operand
OP_n : current state of operator

equal i_b_alu_equal	**enable** i_b_alu_op_en	**operator keycode** i_b_alu_op_keycode[3:0]	**Operand(a)** i_b_alu_operand[8:0]	**result** o_b_alu_result[8:0]	**overflow flag** o_b_alu_overflow_flag	**ALU calculator function**
0	0	4'bxxxx	x	Previous result	Previous overflow flag	Output remains unchanged
0	1	4'b00xx	a	previous result OP_{n-1} a	>255? 1: Previous overflow flag	Calculate the result with previous operator
0	1	4'b1000	a= 9'b0_xxxx_xxxx	~a[7:0]	Previous overflow flag	Invert unsigned operand
0	1	4'b1000	a= 9'b1_xxxx_xxxx	~a[8:0]	Previous overflow flag	Invert signed operand
0	1	4'b0101	a	previous result OP_{n-1} a	>255? 1: Previous overflow flag	Calculate the result with previous operator
0	1	4'b011x	a	previous result OP_{n-1} a	>255? 1: Previous overflow flag	
0	1	4'b10xx	a	previous result OP_{n-1} a	>255? 1: Previous overflow flag	
0	1	4'b11xx	x	previous result	Previous overflow flag	Result remains unchanged
0	1	4'b00xx	a	previous result OP_n a	>255? 1: Previous overflow flag	Calculate the result with current operator
1	0	4'b1000	a	~ previous result[7:0]	Previous overflow flag	Invert result
1	0	4'b1000	a	~ previous result[8:0]	Previous overflow flag	Invert result
1	0	4'b0101	a	previous result OP_n a	>255? 1: Previous overflow flag	Calculate the result with current operator
1	0	4'b011x	a	previous result OP_n a	>255? 1: Previous overflow flag	
1	0	4'b10xx	a	previous result OP_n a	>255? 1: Previous overflow flag	
1	0	4'b11xx	x	previous result	Previous overflow flag	Result remains unchanged
1	1	x	x	previous result	Previous overflow flag	

Table 4.4.3: function table of alu calculator

4.4.4 Timing Requirement

Signal	Requirement
i_b_alu_equal	**1 clock cycle** -to trigger the calculation
i_b_alu_op_en	**1 clock cycle** -To trigger the calculation
i_b_alu_op_keycode[3:0]	**2 clock cycles** - to ensure a stable signal can be received
i_b_alu_operand[8:0]	**2 clock cycles** - to ensure a stable signal can be received
o_b_alu_result[8:0]	**1 clock cycle** -to pass the data to 7 segment display decoders
o_b_alu_overflow_flag	**2 clock cycle** -to ensure 7 segment display decoder can read the flag

Table 4.4.4: time requirement of alu calculator

4.4.5 Schematic Diagram

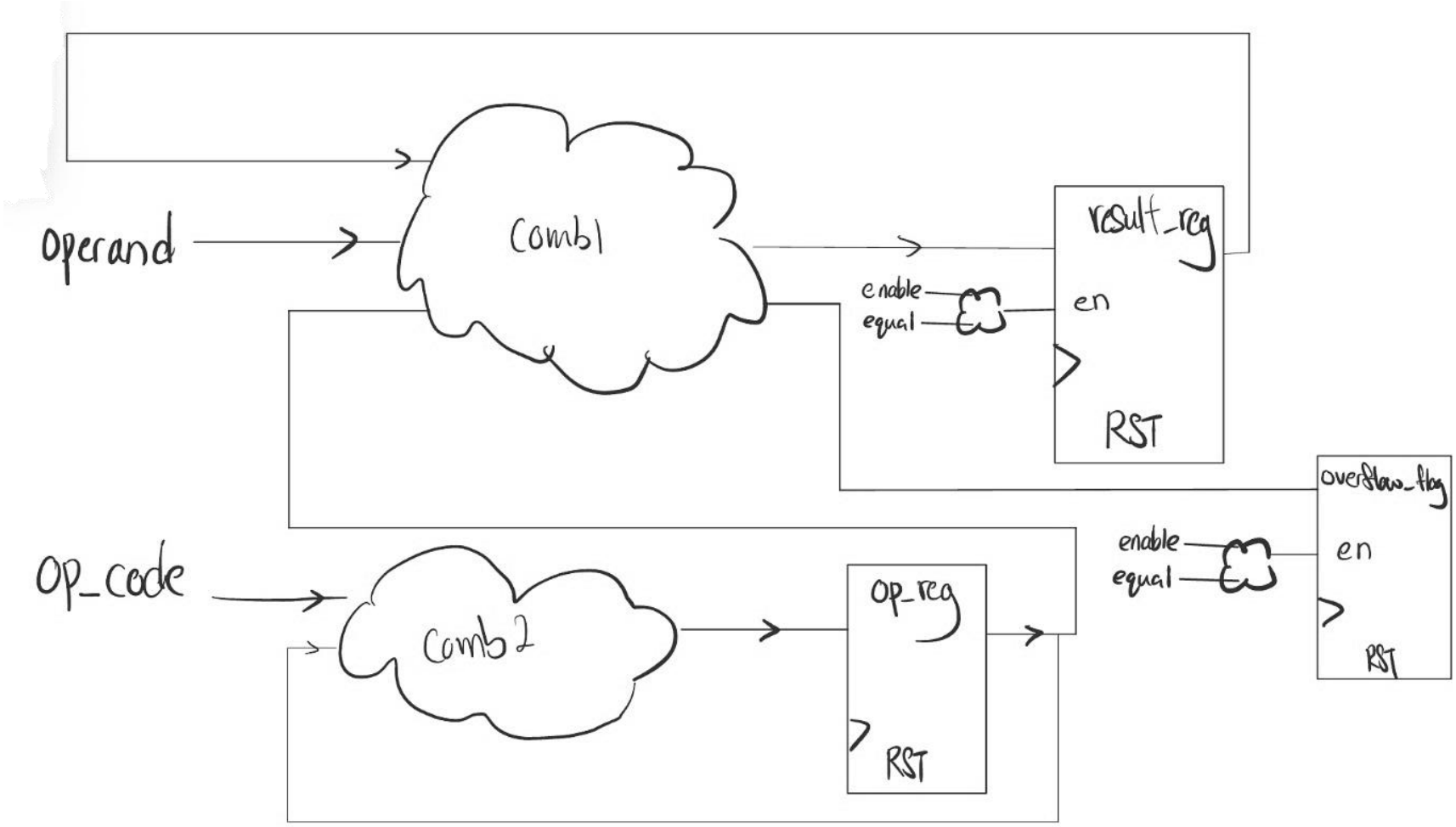

Figure 4.4.5.1 Micro architecture of ALU calculator

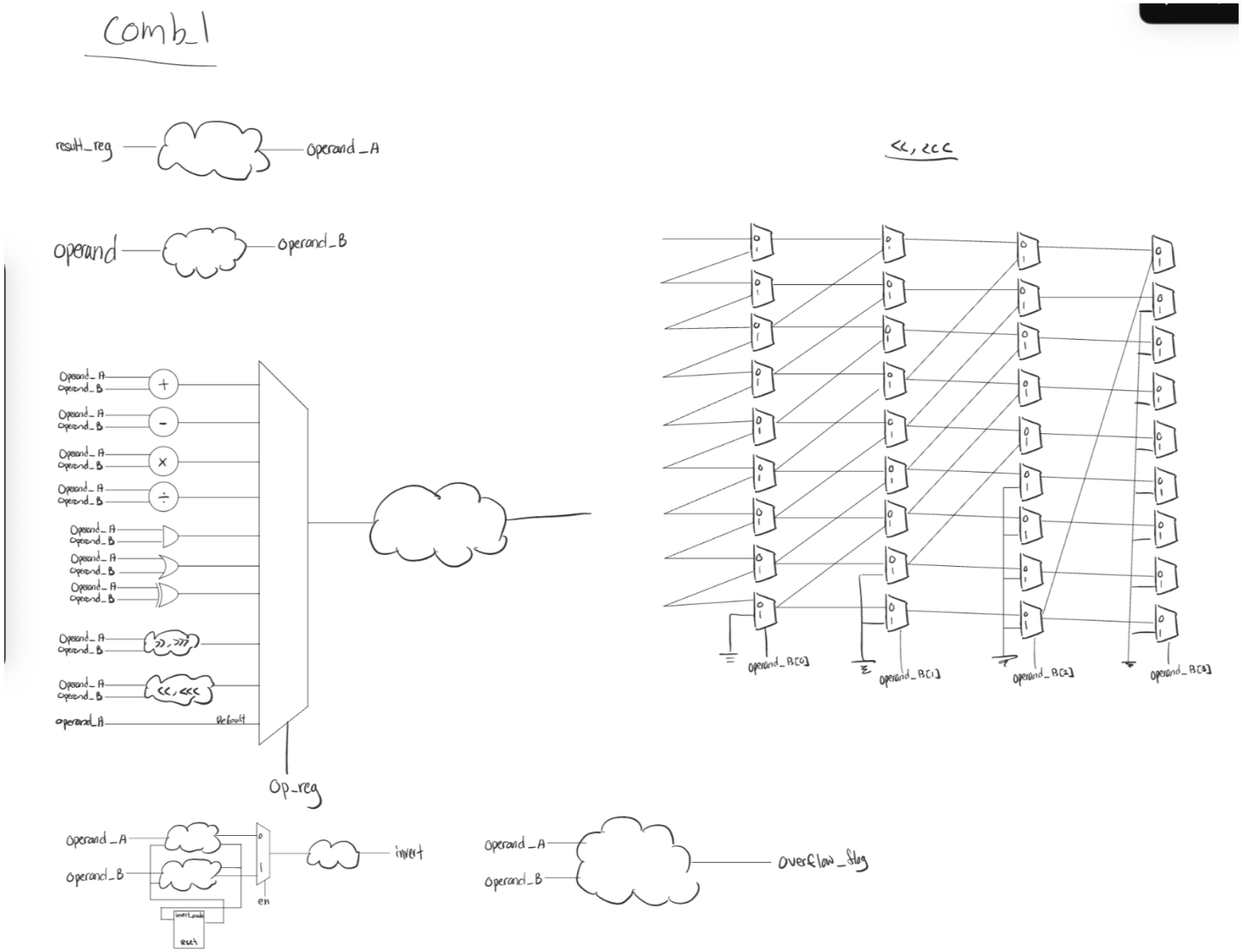

Figure 4.4.5.2 Micro architecture of ALU calculator

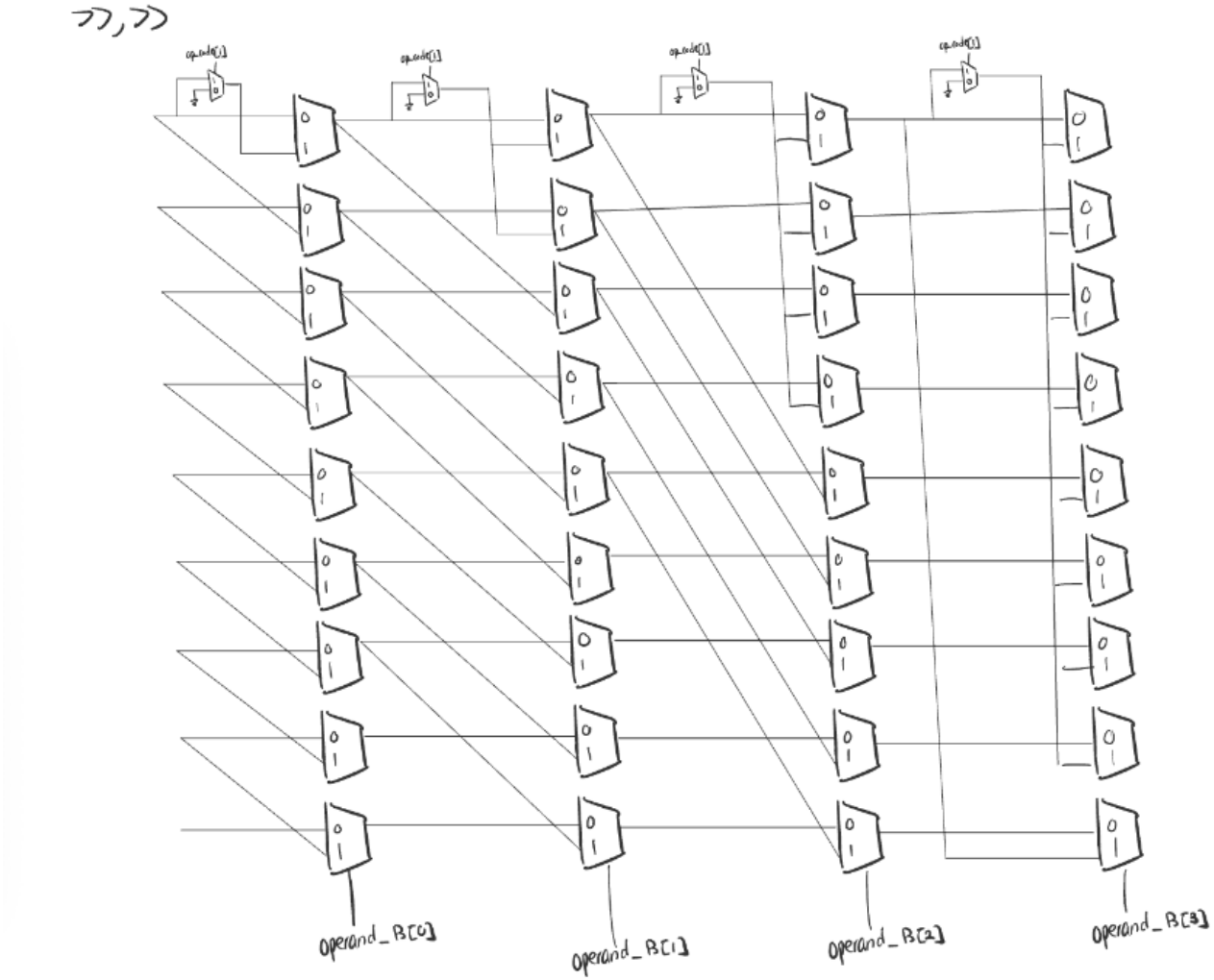

Figure 4.4.5.3 Micro architecture of ALU calculator

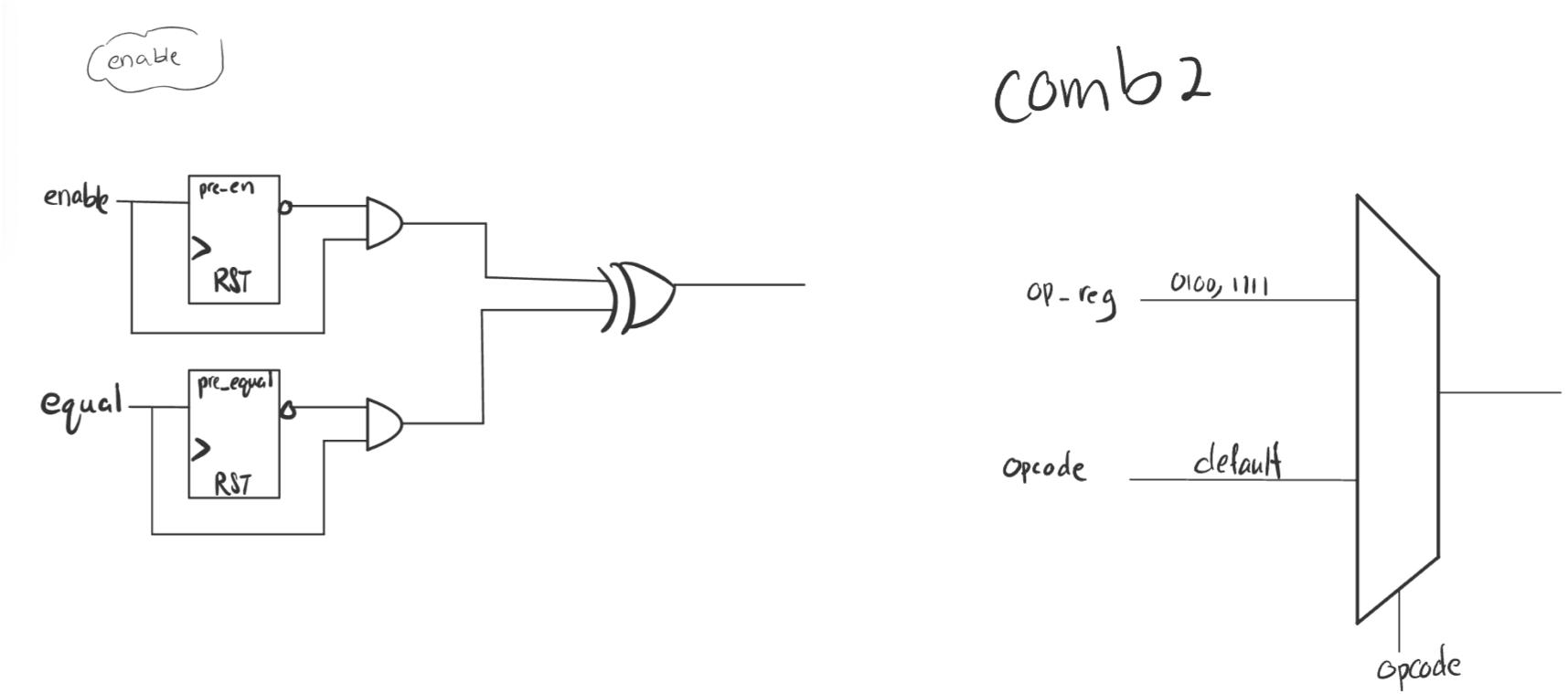

Figure 4.4.5.4 Micro architecture of ALU calculator

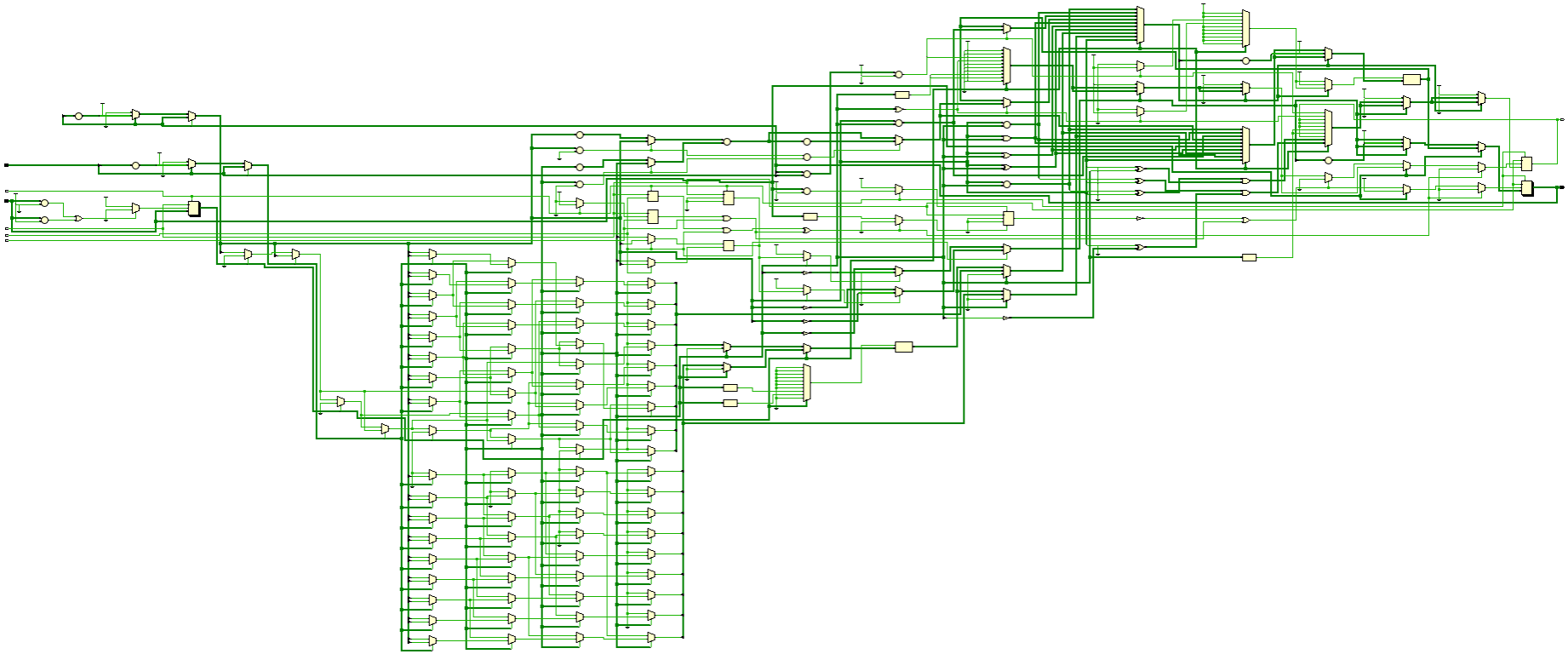

Figure 4.4.5.1: Overall view of schematic diagram of alu calculator

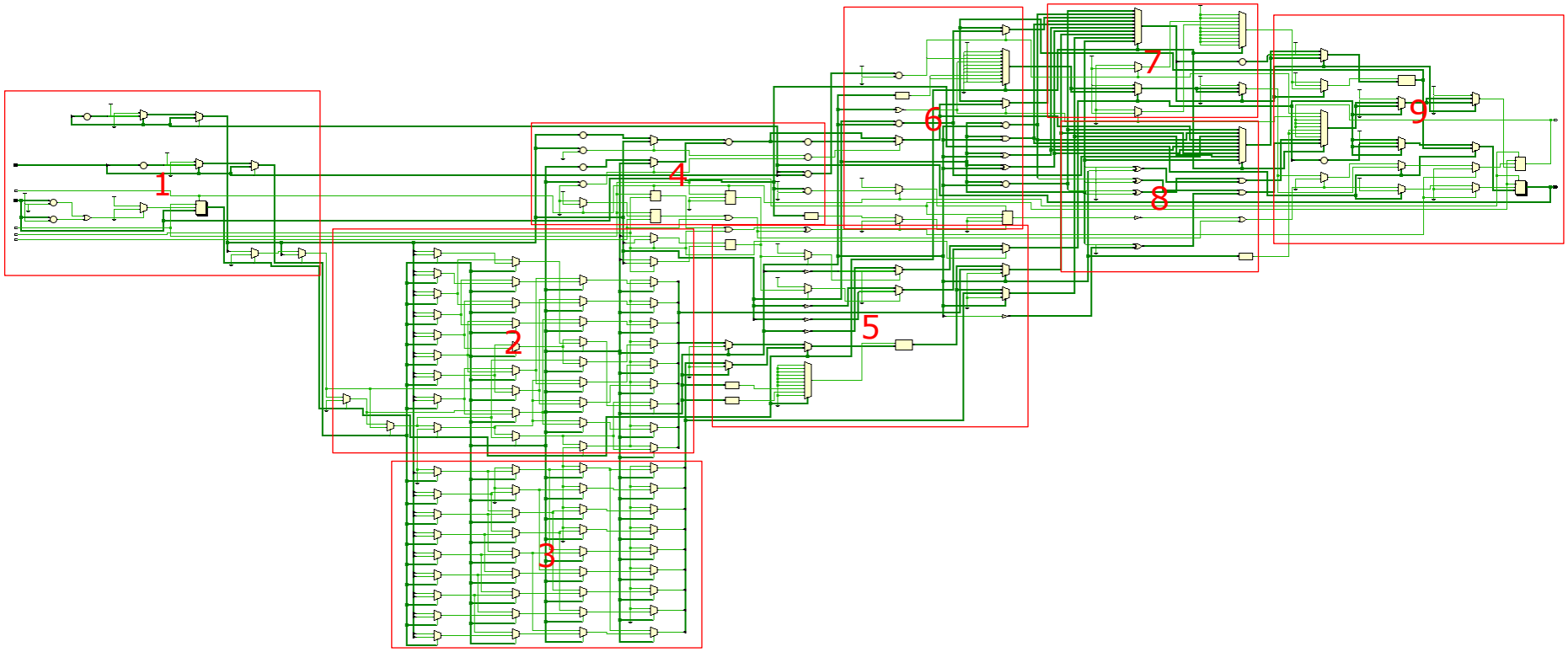

Figure 4.4.5.2: Parts of the schematic diagram

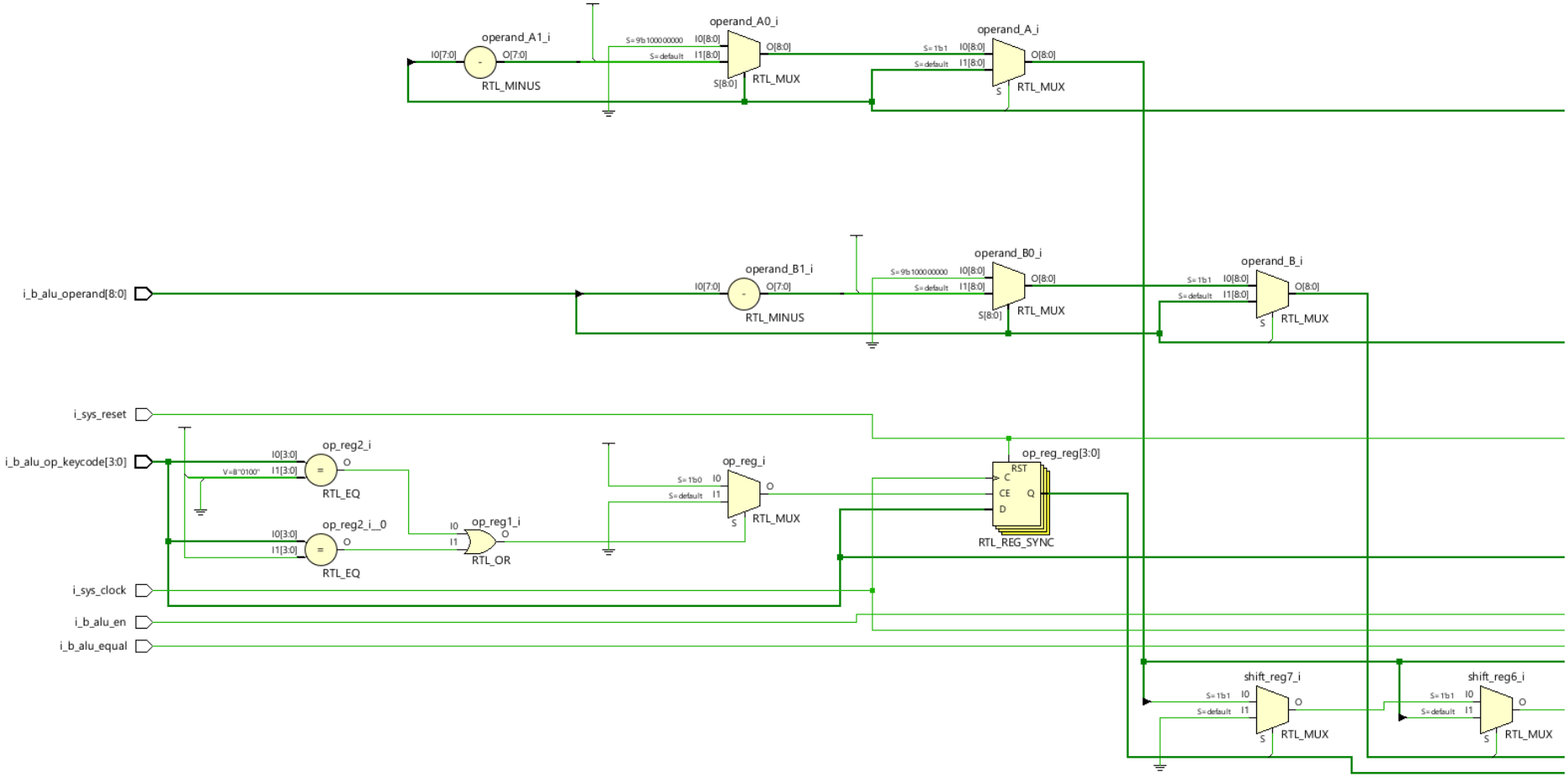

Figure 4.4.5.3: schematic diagram part 1

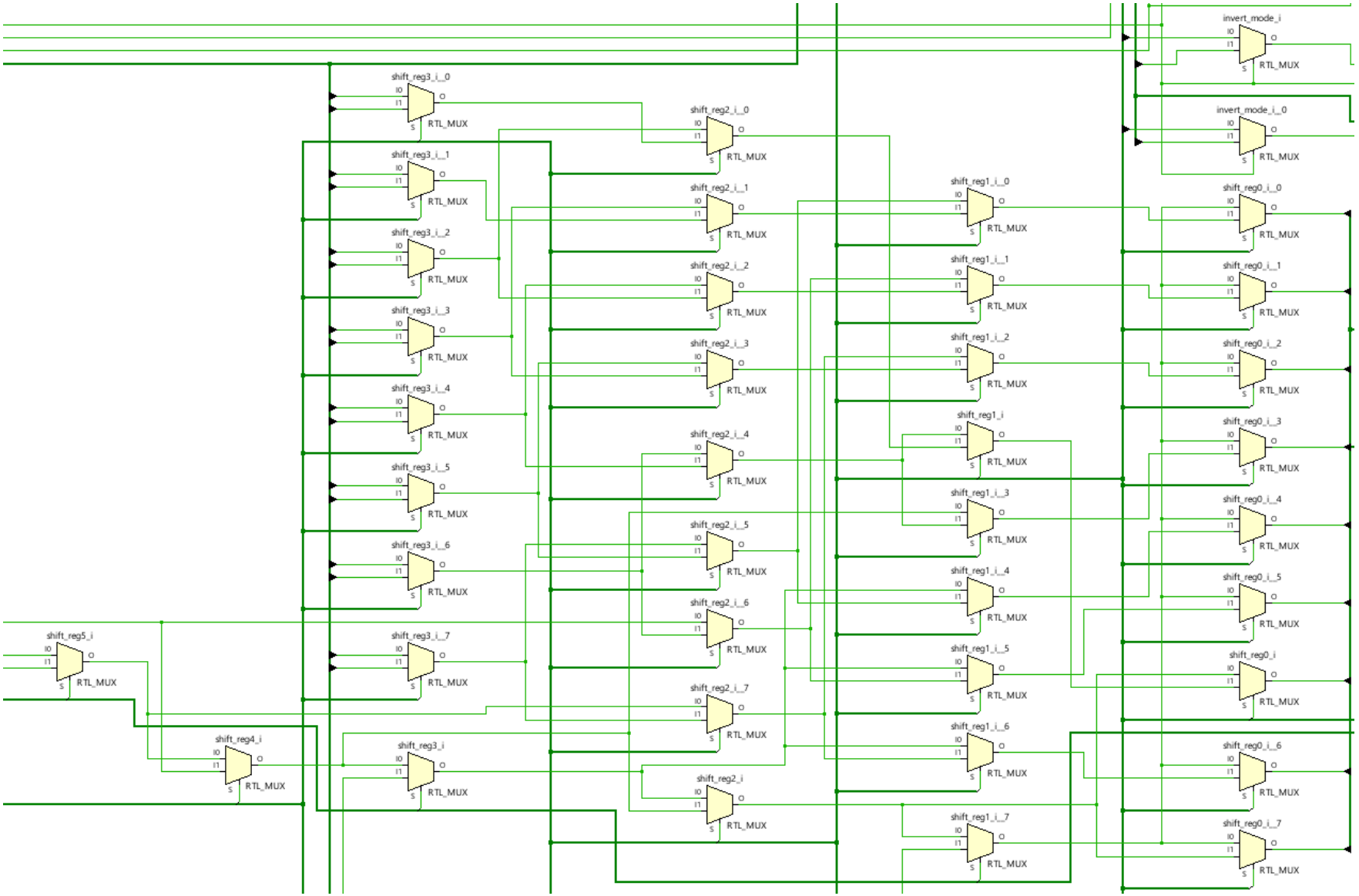

Figure 4.4.5.4: schematic diagram part 2

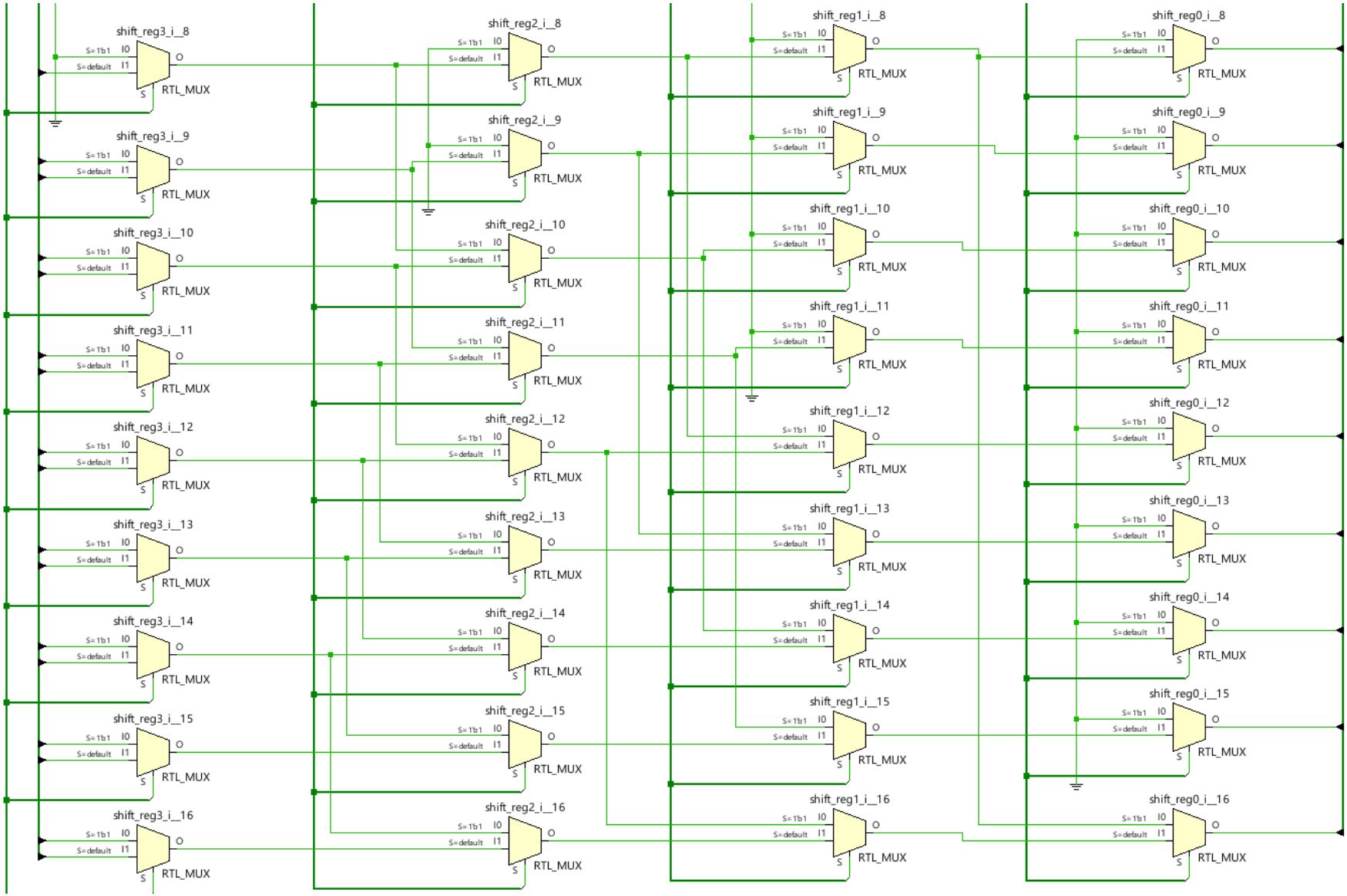

Figure 4.4.5.5: schematic diagram part 3

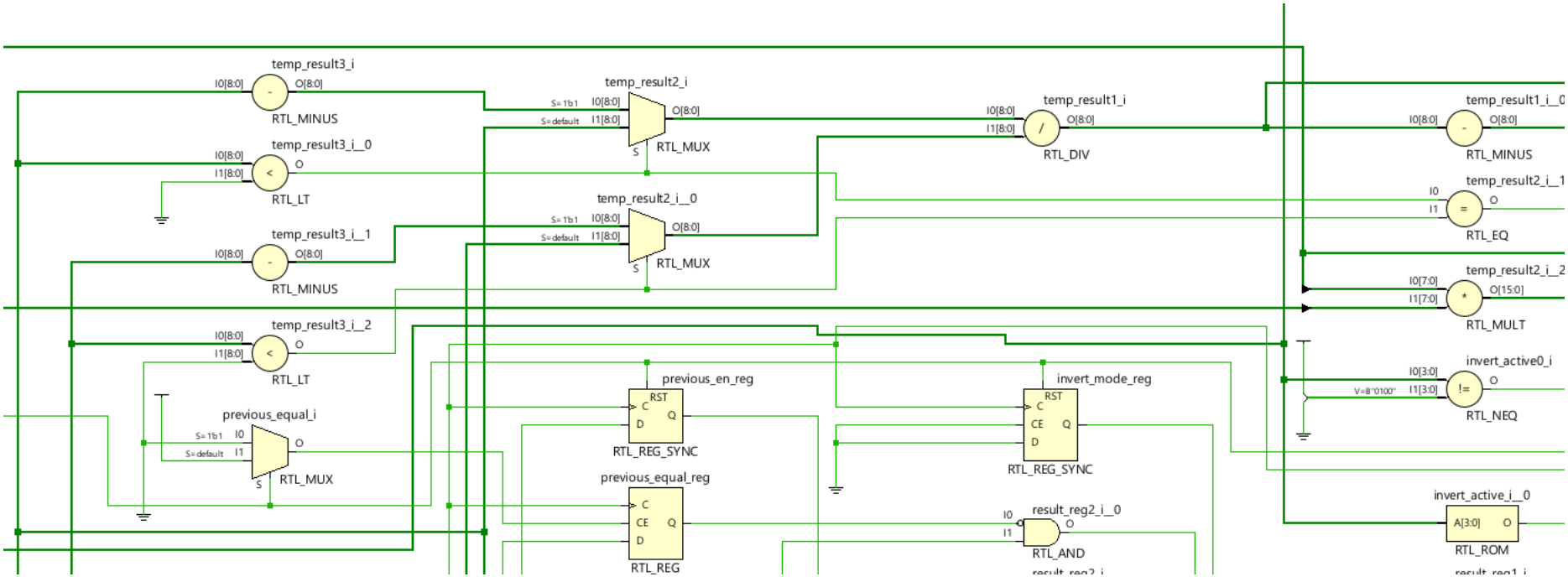

Figure 4.4.5.6: schematic diagram part 4

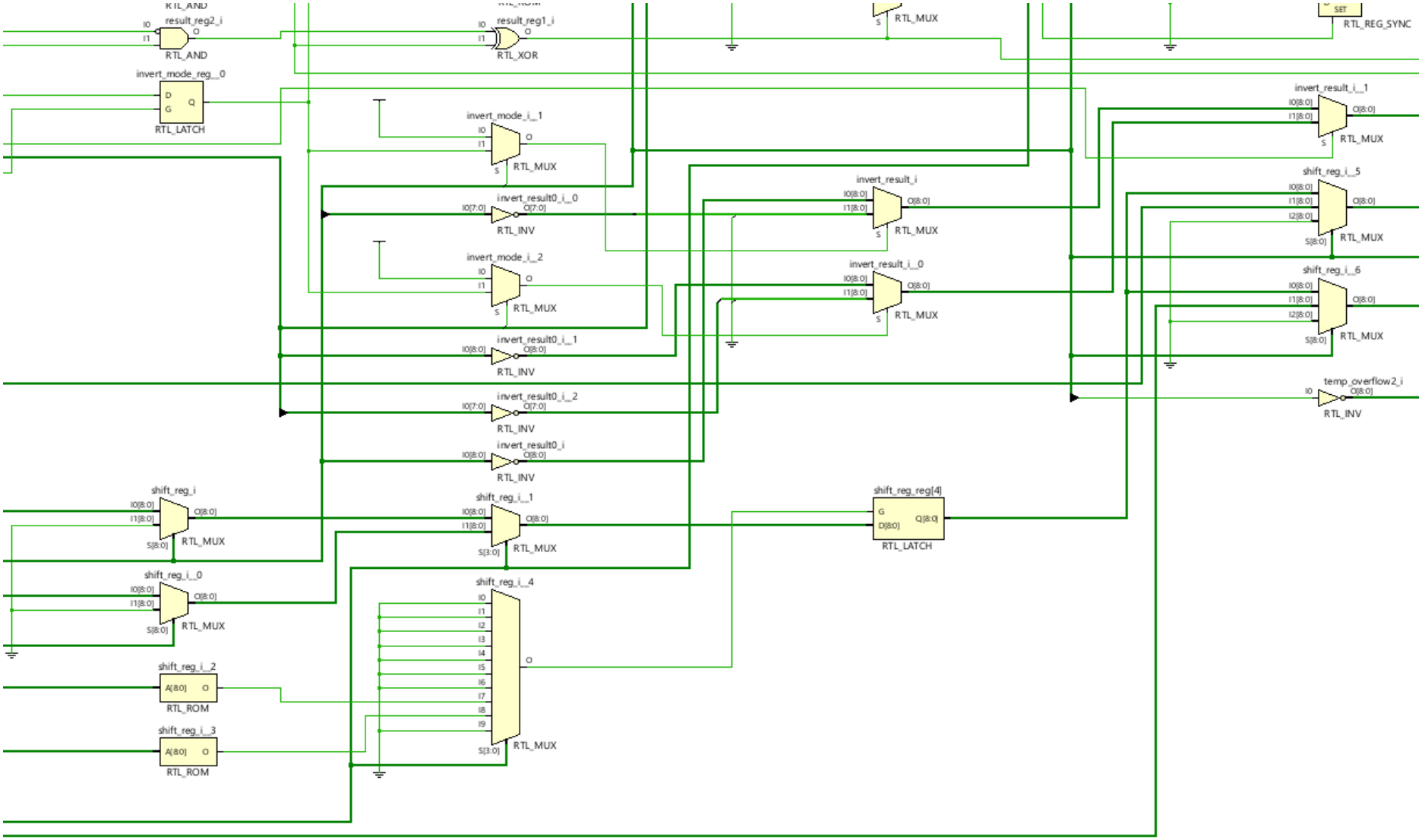

Figure 4. 4.5.7: schematic diagram part 5

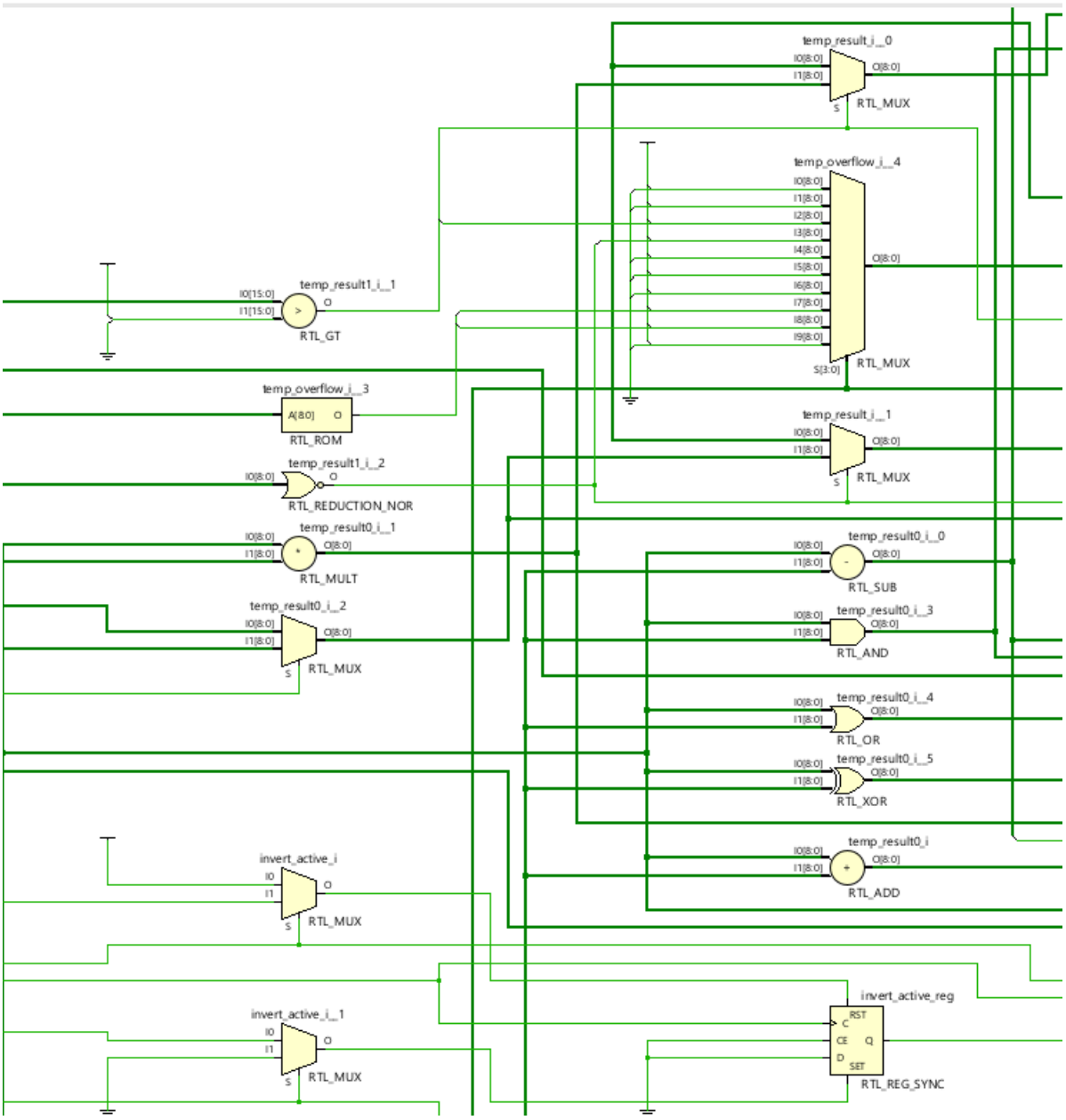

Figure 4.4.5.8: schematic diagram part 6

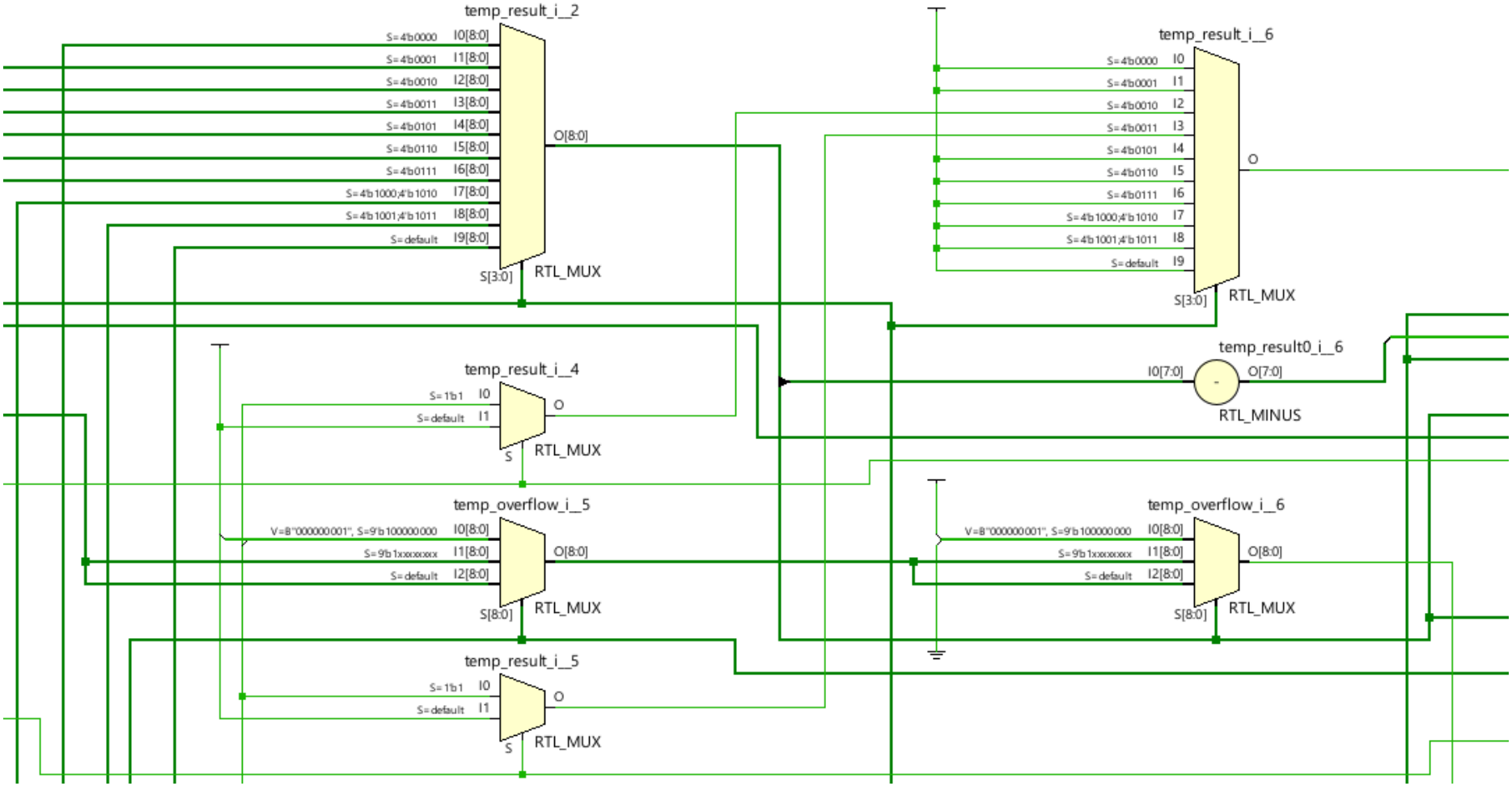

Figure 4.4.5.9: schematic diagram part 7

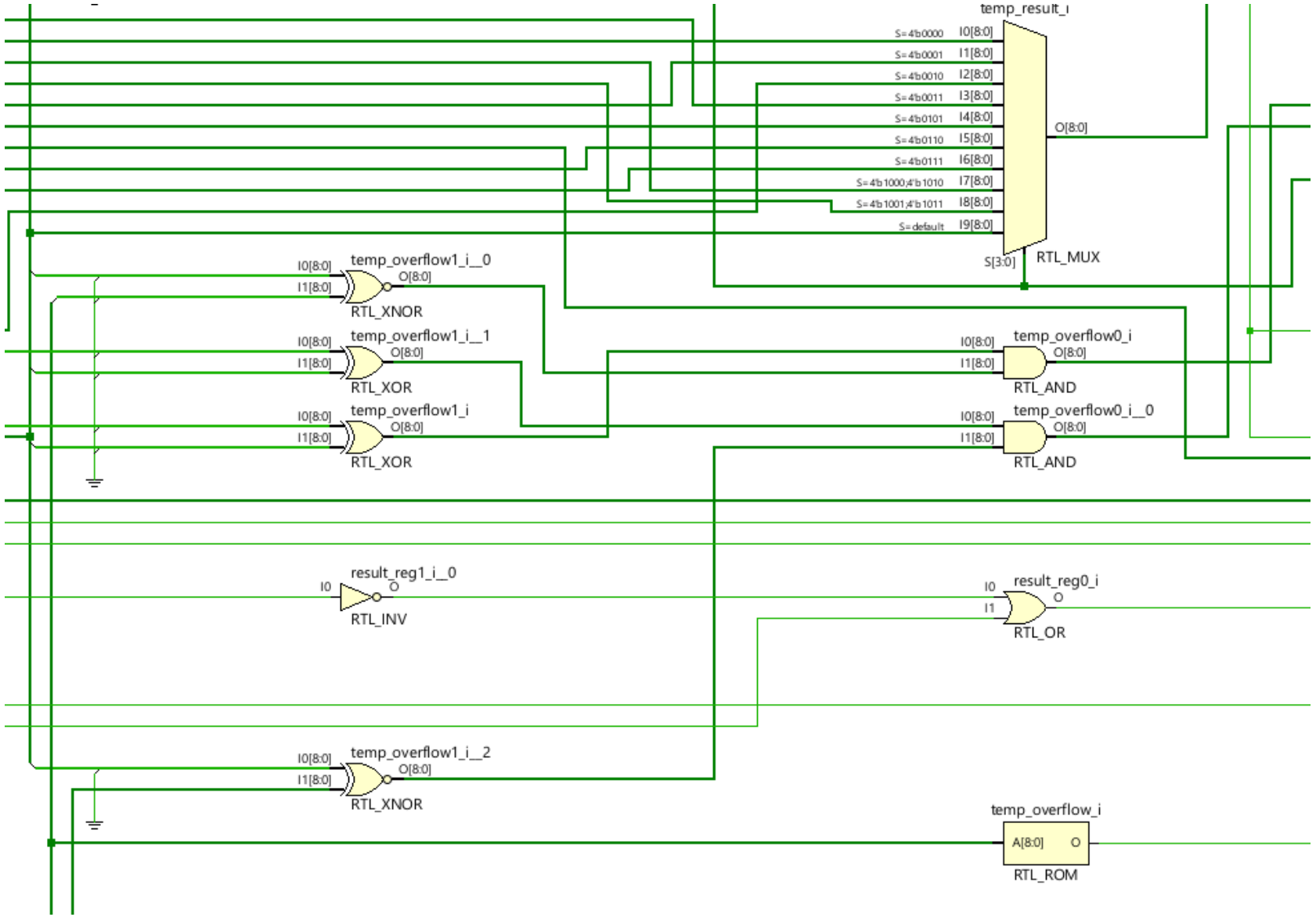

Figure 4.4.5.10: schematic diagram part 8

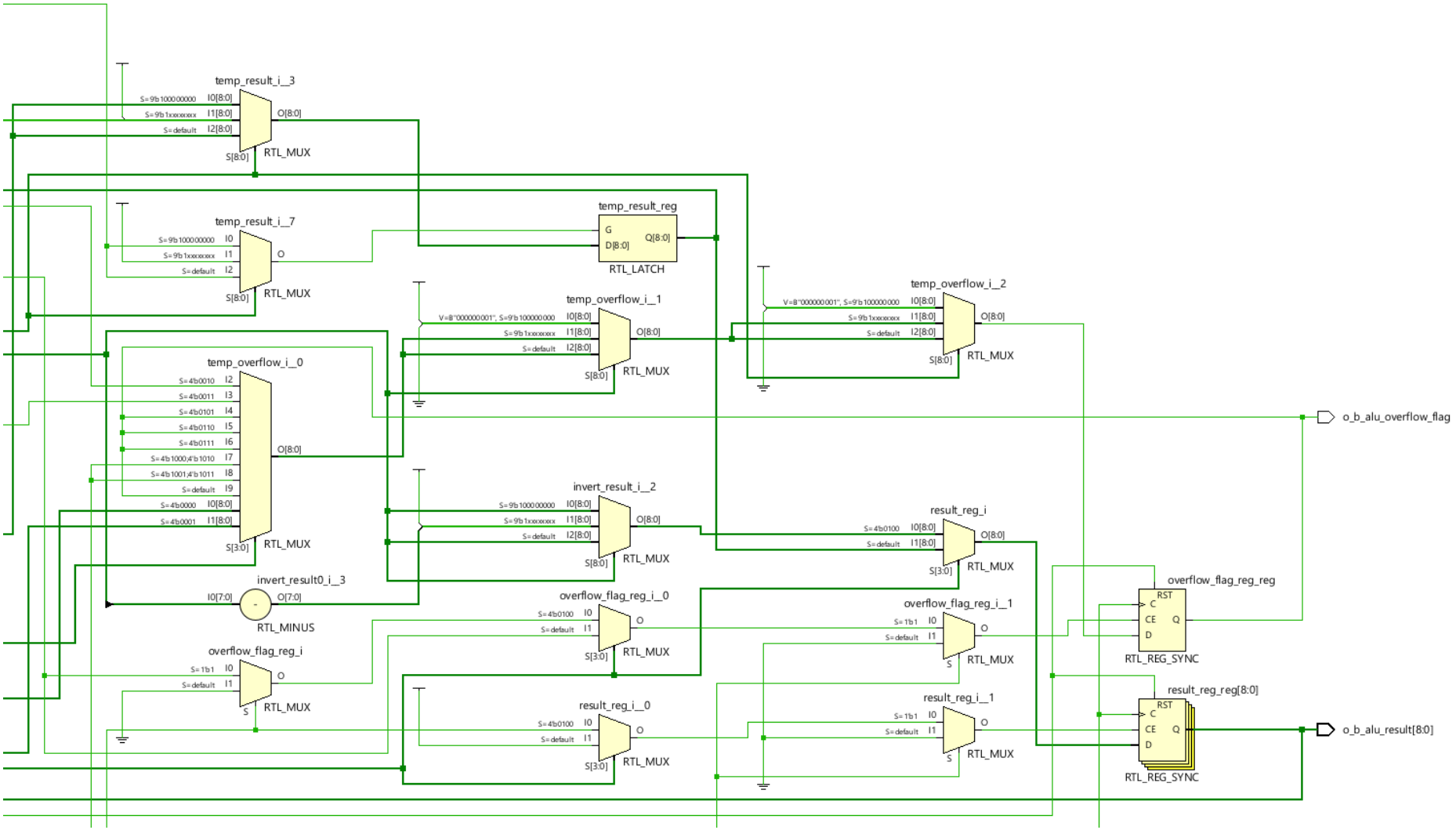

Figure 4.4.5.11: schematic diagram part 9

4.4.6 System Verilog Model

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Chow Bin Lin
//
// Create Date: 04/21/2025 01:29:31 AM
// File Name:b_alu.sv
// Module Name: b_alu
// Project Name: 8-bit integer calculator
// Code type: RTL Level
// Description: Modelling of ALU calculator
//////////////////////////////////////////////////////////////////////////////////


module b_alu(
output logic[8:0] o_b_alu_result, // result output
output logic o_b_alu_overflow_flag,// overflow or invalid
input logic  i_sys_clock, // system clock
i_sys_reset, // system reset
i_b_alu_equal, // equal button is trigger
i_b_alu_en,//
input logic[3:0] i_b_alu_op_keycode, // operator keycode
input logic[8:0] i_b_alu_operand// operand get
);

logic invert_active,overflow_flag_reg,invert_mode,previous_en,previous_equal;//
internal reg
logic[3:0] op_reg;
logic[8:0] result_reg,temp_result,temp_overflow, invert_result;
logic[8:0] shift_reg[4:0];
logic signed [8:0]  operand_A, operand_B;

assign operand_A = result_reg[8]? (result_reg == 9'b1_0000_0000 ?'0:{1'b1,-
result_reg[7:0]}):result_reg;
assign operand_B = i_b_alu_operand[8]? (i_b_alu_operand ==
9'b1_0000_0000 ?'0:{1'b1,-i_b_alu_operand[7:0]}):i_b_alu_operand;
assign o_b_alu_result = result_reg;
assign o_b_alu_overflow_flag = overflow_flag_reg;


always_comb begin
        temp_overflow = overflow_flag_reg;
        shift_reg[0]= operand_A;// ~
        if(i_b_alu_en) begin// input
            if (operand_B[8])
                invert_mode = 1'b1;
            if (invert_mode)
                invert_result = ~operand_B;
            else
                invert_result = {1'b0,~operand_B[7:0]};
        end
        else begin
            if (operand_A[8])
                invert_mode = 1'b1;
            if (invert_mode)// result
```

```
            invert_result = ~operand_A;
        else
            invert_result = {1'b0,~operand_A[7:0]};
    end

    case(op_reg)
    4'b0000:// +
    begin
        temp_result = operand_A + operand_B;
        temp_overflow =
(temp_result[8]^operand_A[8])&(operand_A[8]~^operand_B[8]);
    end
    4'b0001://-
    begin
        temp_result = operand_A - operand_B;
        temp_overflow =
(temp_result[8]^operand_A[8])&(operand_A[8]~^(~operand_B[8]));
    end
    4'b0010: // *
    begin
        if(result_reg[7:0]*i_b_alu_operand[7:0]>255)
            temp_overflow = 1'b1;
        else
            temp_result = operand_A * operand_B;
    end
    4'b0011:// /
    begin
        if(~|operand_B)// if zero
            temp_overflow = 1'b1;
        else
            temp_result = operand_A / operand_B; // as default
    end
    4'b0101:// &
    temp_result = operand_A & operand_B;
    4'b0110:// |
    temp_result = operand_A | operand_B;
    4'b0111:// ^
    temp_result = operand_A ^ operand_B;
    4'b1000,4'b1010:// >> or >>>
    begin
    casez(operand_B)
    9'b1_????_????: temp_overflow = 1'b1;

    9'b0_0000_0???,9'b0_0000_100?: begin
    for (int col = 0; col < 4; col++) begin
        for (int row = 0; row < 9; row++) // Each column
            if (row >= 9-2**col)

            shift_reg[col+1][row] = operand_B[col] ? (op_reg[1]?shift_reg[col][8]:0):
shift_reg[col][row] ;
            else

            shift_reg[col+1][row] = operand_B[col] ? shift_reg[col][row+2**col] :
shift_reg[col][row];

        end // end of for loop
    end
```

```
        default:
        shift_reg[4] = 9'b0;
        endcase
        temp_result = shift_reg[4];
        end
        4'b1001,4'b1011:// << or <<<
           begin

        casez(operand_B)
        9'b1_????_????: temp_overflow = 1'b1;

        9'b0_0000_0???,9'b0_0000_100?: begin
        for (int col = 0; col < 4; col++) begin
           for (int row = 0; row < 9; row++)begin // Each column

              if (row < 2**col)

              shift_reg[col+1][row] = operand_B[col] ? 0 : shift_reg[col][row];
              else

              shift_reg[col+1][row] = operand_B[col] ? shift_reg[col][row-2**col] :
shift_reg[col][row];
              end
           end // end of for loop
        end
        default:
        shift_reg[4] = 9'b0;
        endcase
        temp_result = shift_reg[4];
        end
        default:
        temp_result = operand_A;
        endcase

      casez(invert_result) // convert back to signed
      9'b1_0000_0000: temp_overflow = 1; // special case -256
      9'b1_????_????: invert_result = {1'b1,-invert_result[7:0]};
      default : invert_result = invert_result;
      endcase

      casez(temp_result) // convert back to signed
      9'b1_0000_0000: temp_overflow = 1; // special case -256
      9'b1_????_????: temp_result = {1'b1,-temp_result[7:0]};
      default: temp_result = temp_result;
      endcase
end


always_ff @(posedge i_sys_clock)begin
   if(i_sys_reset)begin
      invert_mode <= 1'b0;
      result_reg <= 9'b0;
      overflow_flag_reg <=1'b0;
      previous_en<=1'b0;
      invert_active<=1'b0;
      op_reg <= 4'b0;
   end
```

```
    else begin
    previous_en<= i_b_alu_en;
    previous_equal<=i_b_alu_equal;

    if(~(i_b_alu_op_keycode== 4'b0100||i_b_alu_op_keycode== 4'b1111))
        op_reg<= i_b_alu_op_keycode;


    if((~previous_en&&i_b_alu_en)^(~previous_equal&&i_b_alu_equal))begin // ensure
only run once
        if (i_b_alu_op_keycode == 4'b0100)begin
            invert_active<=1;
            if(!invert_active||(~previous_equal&&i_b_alu_equal))begin
                result_reg<=invert_result;
                overflow_flag_reg<=temp_overflow;
            end
        end
        else begin
            result_reg<=temp_result;
            overflow_flag_reg<=temp_overflow;
        end
    end
    if (i_b_alu_op_keycode != 4'b0100)
        invert_active<=0;
    end
    end
endmodule
```

4.4.7 Test Plan

No.	Test Case	Description of test vector Generation	Expected Output	Status
1	Reset	1. i_sys_reset = 1	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_overflow_flag =0	pass
2	Input	1. i_b_alu_operand= 9'b0_0000_0010 2. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] =9'b0_0000_0010	pass
3	+	1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode= 4'b0000 3. i_b_alu_operand= 9'b0_0000_0100 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010 4. o_b_alu_result[8:0] = 9'b0_0000_0110	pass
4	-	1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode= 4'b0001 3. i_b_alu_operand= 9'b0_0000_0100 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010 4. o_b_alu_result[8:0] = 9'b1_0000_0010	pass
5	X	1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode= 4'b0010 3. i_b_alu_operand= 9'b0_0000_0100 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010 4. o_b_alu_result[8:0] = 9'b0_0000_ 1000	pass
6	÷	1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode=	1. o_b_alu_result[8:0] = 9'b0_0000_0000	pass

		4’b0011 3. i_b_alu_operand= 9’b0_0000_0100 4. press equal button	2. o_b_alu_result[8:0] = 9’b0_0000_0010 3. o_b_alu_result[8:0] = 9’b0_0000_0010 4. o_b_alu_result[8:0] = 9’b0_0000_ 0000	
7a	~ With positive value	1. i_b_alu_operand= 9’b0_0000_0010 2. i_b_alu_op_keycode= 4’b0100 3. press equal button	1. o_b_alu_result[8:0] = 9’b0_0000_0000 2. o_b_alu_result[8:0] = 9’b0_1111_1101 3. o_b_alu_result[8:0] = 9’b0_0000_0010	pass
7b	~ With negative value	1. i_b_alu_operand= 9’b1_0000_0010 2. i_b_alu_op_keycode= 4’b0100 3. press equal button	1. o_b_alu_result[8:0] = 9’b0_0000_0000 2. o_b_alu_result[8:0] = 9’b1a_0000_0011 3. o_b_alu_result[8:0] = 9’b0_0000_0010	pass
8	&	1. i_b_alu_operand= 9’b0_1111_0110 2. i_b_alu_op_keycode= 4’b0101 3. i_b_alu_operand= 9’b0_0000_0010 4. press equal button	1. o_b_alu_result[8:0] = 9’b0_0000_0000 2. o_b_alu_result[8:0] = 9’b0_1111_0110 3. o_b_alu_result[8:0] = 9’b0_1111_0110 4. o_b_alu_result[8:0] = 9’b0_0000_0010	pass
9	|	1. i_b_alu_operand= 9’b0_1111_0110 2. i_b_alu_op_keycode= 4’b0110 3. i_b_alu_operand= 9’b0_0000_0011 4. press equal button	1. o_b_alu_result[8:0] = 9’b0_0000_0000 2. o_b_alu_result[8:0] = 9’b0_1111_0110 3. o_b_alu_result[8:0] = 9’b0_1111_0110 4. o_b_alu_result[8:0] = 9’b0_1111_0111	pass
10	^	1. i_b_alu_operand= 9’b0_1111_0110 2. i_b_alu_op_keycode= 4’b0111	1. o_b_alu_result[8:0] = 9’b0_0000_0000 2. o_b_alu_result[8:0] = 9’b0_1111_0110	pass

		3. i_b_alu_operand= 9'b0_0000_0011 4. press equal button	3. o_b_alu_result[8:0] = 9'b0_1111_0110 4. o_b_alu_result[8:0] = 9'b0_1111_0101	
11	>>	1. i_b_alu_operand= 9'b0_0010_0000 2. i_b_alu_op_keycode= 4'b1000 3. i_b_alu_operand= 9'b0_0000_0100 1. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0010_0000 3. o_b_alu_result[8:0] = 9'b0_0010_0000 4. o_b_alu_result[8:0] = 9'b0_0000_0010	pass
12	<<	2. i_b_alu_operand= 9'b0_0000_0010 3. i_b_alu_op_keycode= 4'b1001 4. i_b_alu_operand= 9'b0_0000_0100 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010 4. o_b_alu_result[8:0] = 9'b0_0010_0000	pass
13	>>>	1. i_b_alu_operand= 9'b1_0000_0010 2. i_b_alu_op_keycode= 4'b1010 3. i_b_alu_operand= 9'b0_0000_0100 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b1_0000_0010 3. o_b_alu_result[8:0] = 9'b1_0000_0010 4. o_b_alu_result[8:0] = 9'b1_0000_0001	pass
14	<<<	After reset 1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode= 4'b1011 3. i_b_alu_operand= 9'b0_0000_0100 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010 4. o_b_alu_result[8:0] = 9'b0_0010_0000	pass
15	Continuous Calculation	1. i_b_alu_operand= 9'b0_0000_1010	1. o_b_alu_result[8:0] = 9'b0_0000_0000	pass

	and press equal button multiple times	2. i_b_alu_op_keycode= 4'b0000 3. i_b_alu_operand= 9'b0_0000_1000 4. i_b_alu_op_keycode= 4'b0010 5. i_b_alu_operand= 9'b0_0000_0010 6. press equal button 7. press equal button	2. o_b_alu_result[8:0] = 9'b0_0000_1010 3. o_b_alu_result[8:0] = 9'b0_0000_1010 4. o_b_alu_result[8:0] = 9'b0_0001_0010 5. o_b_alu_result[8:0] = 9'b0_0001_0010 6. o_b_alu_result[8:0] = 9'b0_0010_0100 7. o_b_alu_result[8:0] = 9'b0_0100_1000	
16	Overflow	1. i_b_alu_operand= 9'b0_1111_1111 2. i_b_alu_op_keycode= 4'b0000 3. i_b_alu_operand= 9'b0_1111_1111 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_1111_1111 3. o_b_alu_result[8:0] = 9'b0_1111_1111 4. o_b_alu_overflow_flag = 1 5. o_b_alu_result[8:0] = 9'b1_1111_1110	pass
17	Invalid calculation	1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode= 4'b0011 3. i_b_alu_operand= 9'b0_0000_0000 4. press equal button	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010 4. o_b_alu_overflow_flag = 1 5. o_b_alu_result[8:0] = 9'b0_0000_0010	
18	Invalid input for swifter	1. i_b_alu_operand= 9'b0_0000_0010 2. i_b_alu_op_keycode= 4'b1011 3. i_b_alu_operand= 9'b1_0000_001	1. o_b_alu_result[8:0] = 9'b0_0000_0000 2. o_b_alu_result[8:0] = 9'b0_0000_0010 3. o_b_alu_result[8:0] = 9'b0_0000_0010	

		4. press equal button	4. o_b_alu_overflow_flag = 1 5. o_b_alu_result[8:0] = 9'b0_0000_0010	

Table 4.4.5: Test plan of alu calculator

4.4.8 Testbench and Simulation results

4.4.8.1 Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Chow Bin Lin
//
// Create Date: 05/05/2025 12:28:34 PM
// File Name:tb_b_alu.sv
// Module Name: tb_b_alu
// Project Name: 8-bit integer calculator
// Code type: Behavioural
// Description: Teshbench for ALU calculator
//
//////////////////////////////////////////////////////////////////////////////////


module b_alu_testbench();
logic tb_i_sys_clock=0, tb_i_sys_reset=0, tb_i_b_alu_equal=0, tb_i_b_alu_en=0,
tb_o_b_alu_overflow_flag;
logic [3:0] tb_i_b_alu_op_keycode=0;
logic [8:0] tb_i_b_alu_operand=0, tb_o_b_alu_result;



always
 #5 tb_i_sys_clock = ~tb_i_sys_clock;

 b_alu
alu(.o_b_alu_result(tb_o_b_alu_result),.o_b_alu_overflow_flag(tb_o_b_alu_overflow_fla
g),.i_sys_clock(tb_i_sys_clock),.i_sys_reset(tb_i_sys_reset)
 ,.i_b_alu_equal(tb_i_b_alu_equal),.i_b_alu_en(tb_i_b_alu_en),.i_b_alu_op_keycode(tb
_i_b_alu_op_keycode),.i_b_alu_operand(tb_i_b_alu_operand));

 initial begin
    // case 1:  reset
    #2 tb_i_sys_reset <= 1;

    // case 2: input
    #10 tb_i_sys_reset = 0;
    #10 tb_i_b_alu_operand<= 9'b0_0000_0010;
        tb_i_b_alu_equal<= 1;

    // case 3: +
    #10 tb_i_sys_reset <= 1; // reset
        tb_i_b_alu_op_keycode<=4'b0000;// when reset is trigger 4'b0000 would be
received
        tb_i_b_alu_equal<= 0;
    #10 tb_i_sys_reset <= 0;
        tb_i_b_alu_operand<= 9'b0_0000_0010;
    #10 tb_i_b_alu_op_keycode<=4'b0000;
        tb_i_b_alu_en<= 1;
    #10 tb_i_b_alu_en <= 0;
        tb_i_b_alu_operand <=9'b0_0000_0100;
    #10 tb_i_b_alu_equal<=1;
```

```
// case 4: -
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b0001;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
#10 tb_i_b_alu_equal<=1;

// case 5: *
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b0010;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
#10 tb_i_b_alu_equal<=1;

// case 6: /
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b0011;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
#10 tb_i_b_alu_equal<=1;

// case 7a: ~ With positive value

#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b0100;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_equal<=1;

// case 7b: ~ With negative value
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b1_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b0100;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
```

```
   tb_i_b_alu_equal<=1;

// case 8: &
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_1111_0110;
#10 tb_i_b_alu_op_keycode<=4'b0101;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0010;
#10 tb_i_b_alu_equal<=1;

// case 9: |
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_1111_0110;
#10 tb_i_b_alu_op_keycode<=4'b0110;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0011;
#10 tb_i_b_alu_equal<=1;

// case 10: ^
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_1111_0110;
#10 tb_i_b_alu_op_keycode<=4'b0111;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0011;
#10 tb_i_b_alu_equal<=1;

   // case 11: >>
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0010_0000;
#10 tb_i_b_alu_op_keycode<=4'b1000;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
 #10 tb_i_b_alu_equal<=1;

// case 12: <<
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b1001;
```

```
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
#10 tb_i_b_alu_equal<=1;



// case 13: >>>
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b1_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b1010;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
#10 tb_i_b_alu_equal<=1;


// case 14: <<<
#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_0010;
#10 tb_i_b_alu_op_keycode<=4'b1011;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0100;
#10 tb_i_b_alu_equal<=1;



// case 15: Continuous Calculation and press equal button multiple times

#10 tb_i_sys_reset <= 1;
   tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_equal<= 0;
#10 tb_i_sys_reset <= 0;
   tb_i_b_alu_operand<= 9'b0_0000_1010;
#10 tb_i_b_alu_op_keycode<=4'b0000;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_1000;
#10 tb_i_b_alu_op_keycode<=4'b0010;
   tb_i_b_alu_en<= 1;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0010;
#10 tb_i_b_alu_en <= 0;
   tb_i_b_alu_operand <=9'b0_0000_0010;
#10 tb_i_b_alu_equal<=1;
#10 tb_i_b_alu_equal<=0;
#10 tb_i_b_alu_equal<=1;
```

```
    // case 16: overflow
    #10 tb_i_sys_reset <= 1;
       tb_i_b_alu_op_keycode<=4'b0000;
       tb_i_b_alu_equal<= 0;
    #10 tb_i_sys_reset <= 0;
       tb_i_b_alu_operand<= 9'b0_1111_1111;
    #10 tb_i_b_alu_op_keycode<=4'b0000;
       tb_i_b_alu_en<= 1;
    #10 tb_i_b_alu_en <= 0;
       tb_i_b_alu_operand <=9'b0_1111_1111;
    # 10tb_i_b_alu_equal<=1;


    // case 17: Invalid calculation
    #10 tb_i_sys_reset <= 1;
       tb_i_b_alu_op_keycode<=4'b0000;
      tb_i_b_alu_equal<= 0;
    #10 tb_i_sys_reset <= 0;
       tb_i_b_alu_operand<= 9'b0_0000_0010;
    #10 tb_i_b_alu_op_keycode<=4'b0011;
       tb_i_b_alu_en<= 1;
    #10 tb_i_b_alu_en <= 0;
       tb_i_b_alu_operand <=9'b0_0000_0000;
    #10 tb_i_b_alu_equal<=1;


    // case 18: Invalid input for swifter
    #10 tb_i_sys_reset <= 1;
       tb_i_b_alu_op_keycode<=4'b0000;
       tb_i_b_alu_equal<= 0;
    #10 tb_i_sys_reset <= 0;
       tb_i_b_alu_operand<= 9'b0_0000_0010;
    #10 tb_i_b_alu_op_keycode<=4'b1011;
       tb_i_b_alu_en<= 1;
    #10 tb_i_b_alu_en <= 0;
       tb_i_b_alu_operand <=9'b1_0000_0001;
    #10 tb_i_b_alu_equal<=1;

end
endmodule
```

4.4.8.2 Simulation result

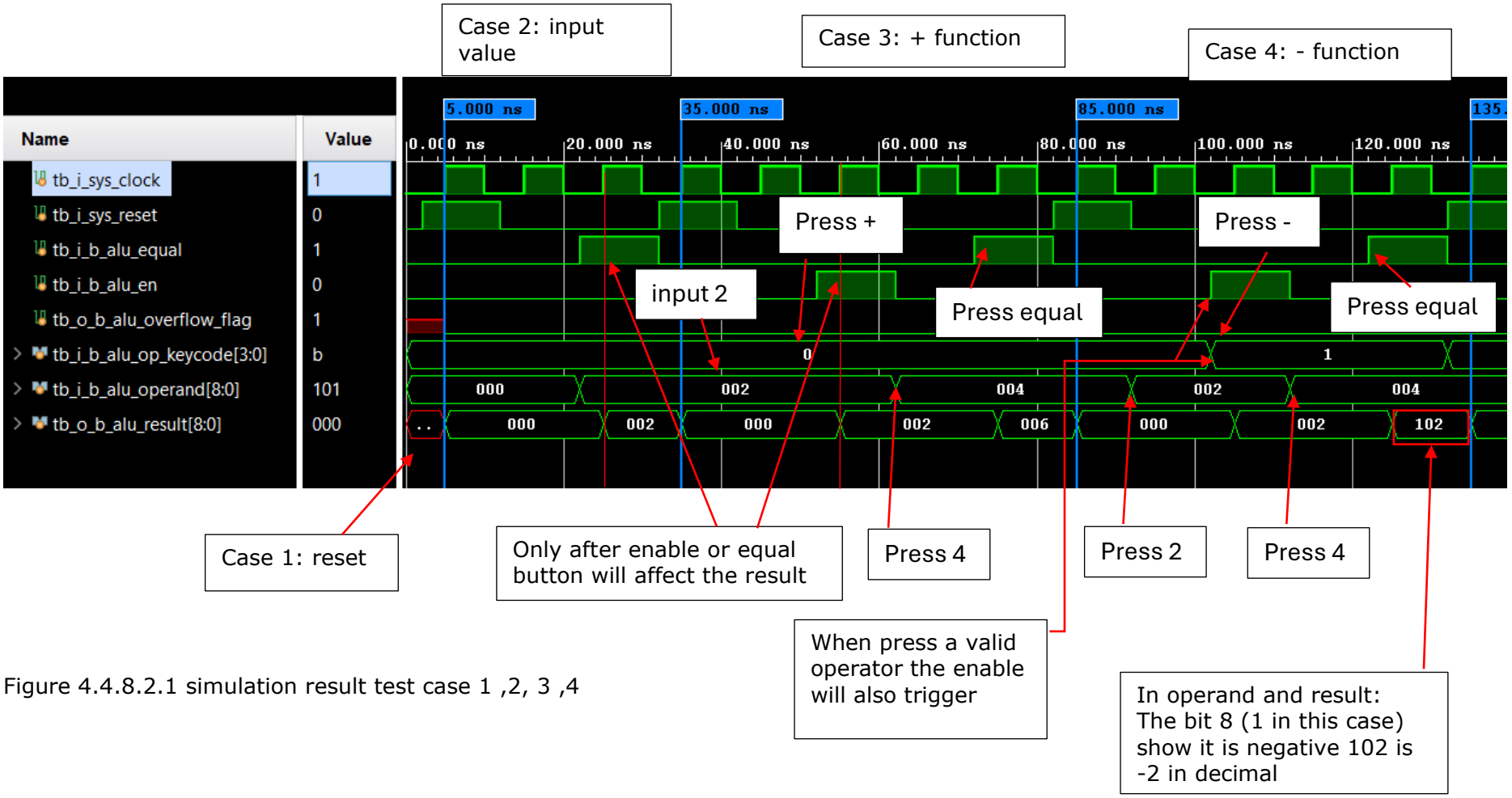

Figure 4.4.8.2.1 simulation result test case 1 ,2, 3 ,4

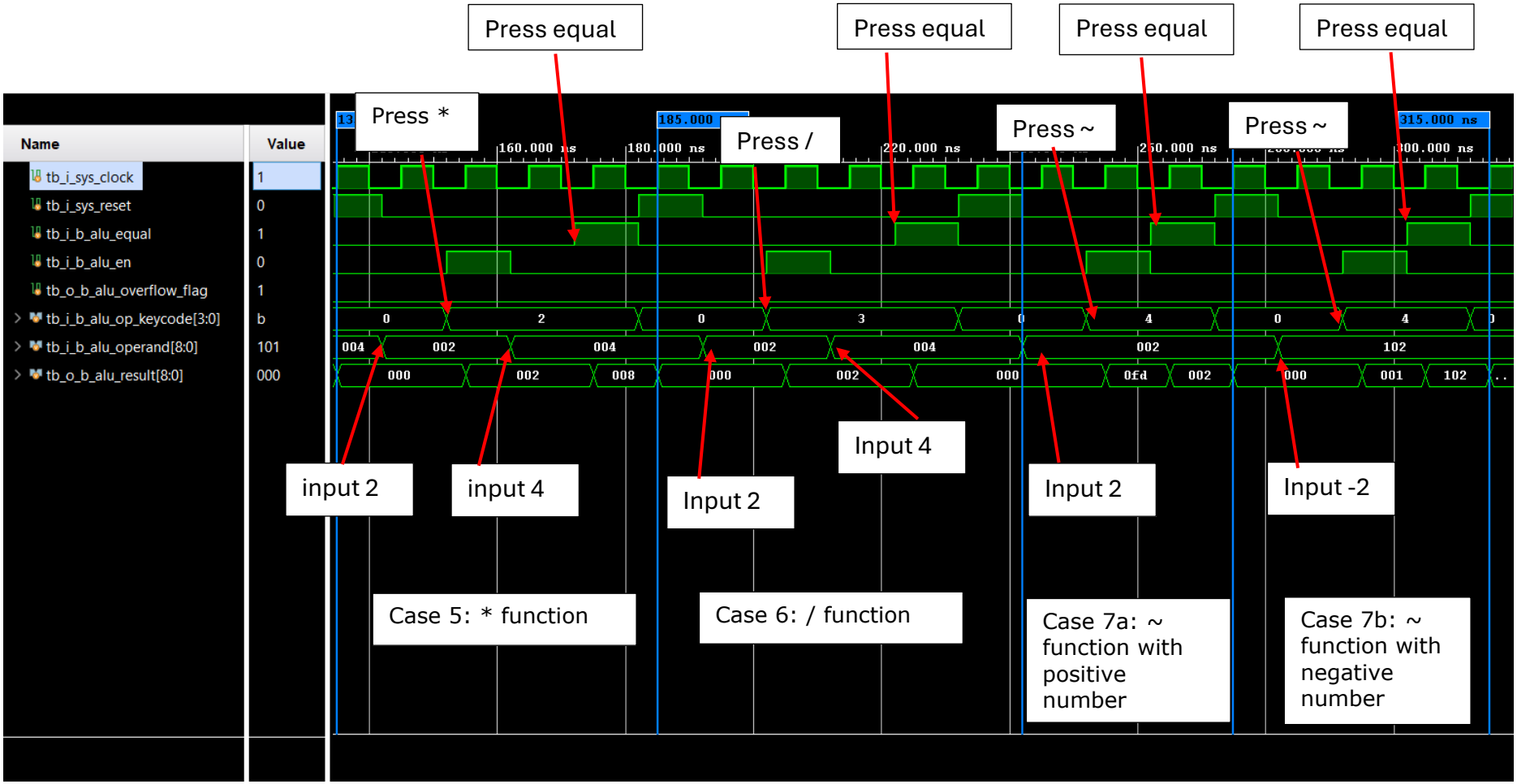

Figure 4.4.8.2.2 simulation result test case 5 ,6, 7a ,7b

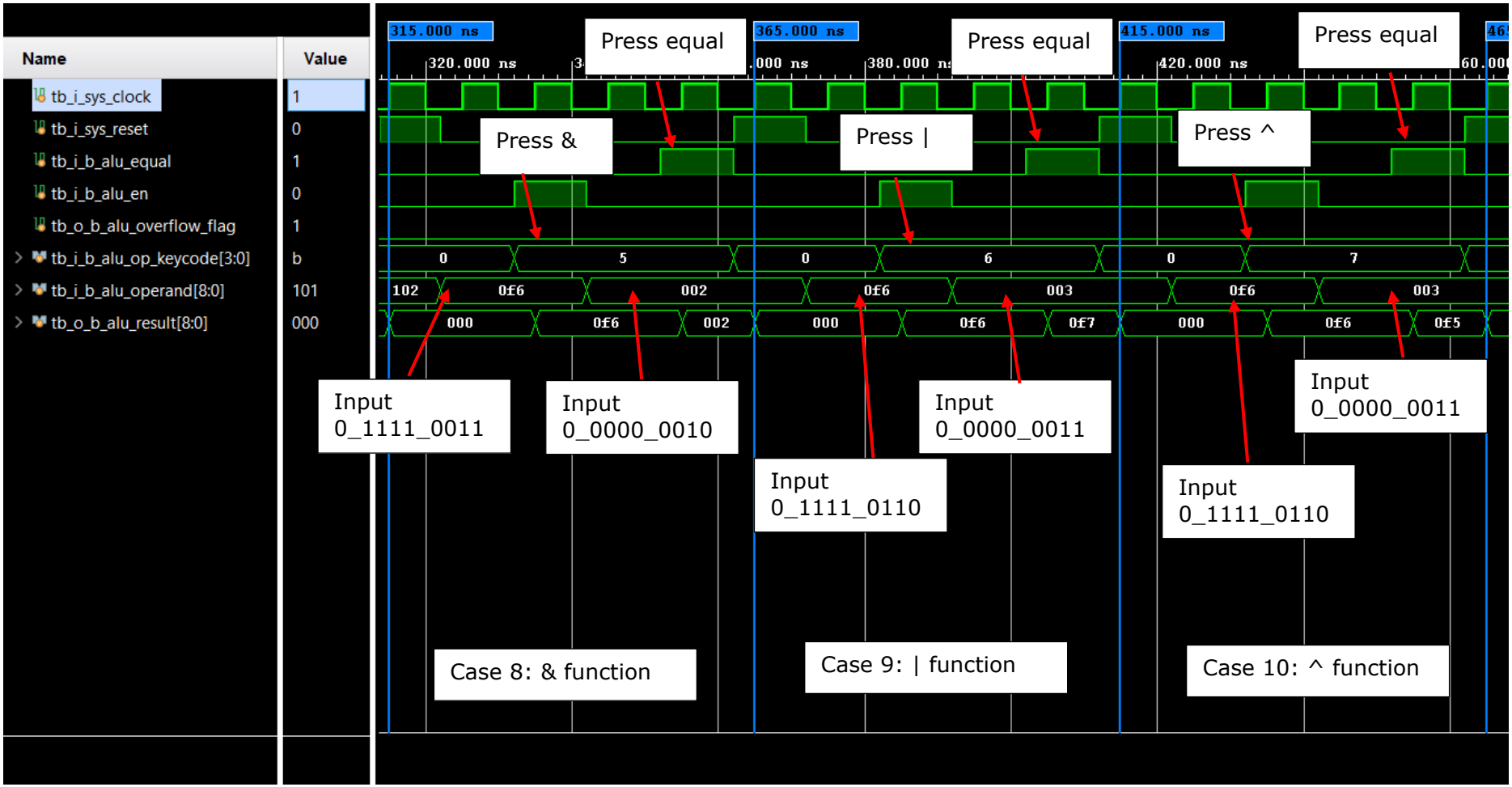

Figure 4.4.8.2.3 simulation result test case 8, 9, 10

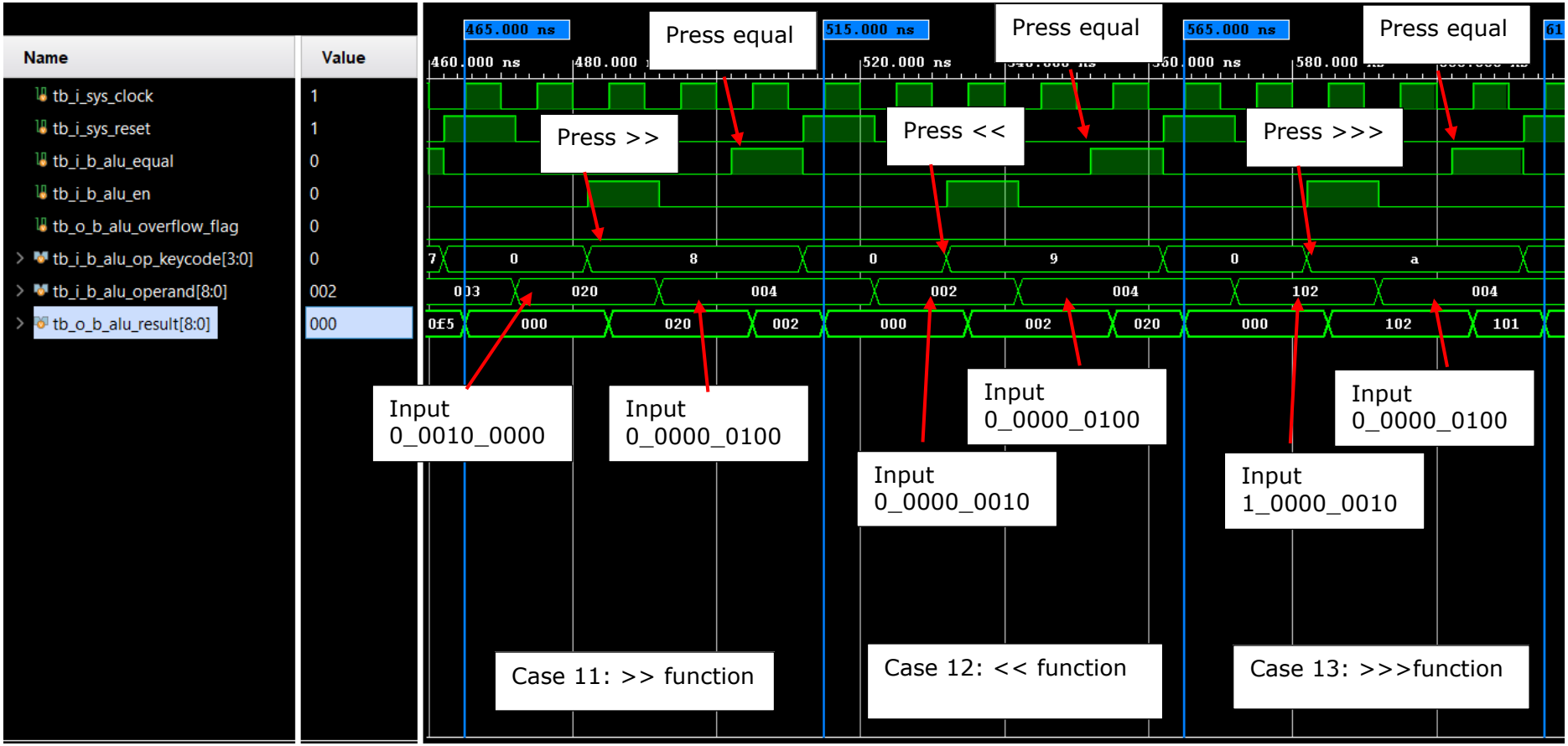

Figure 4.4.8.2.4 simulation result test case 11, 12, 13

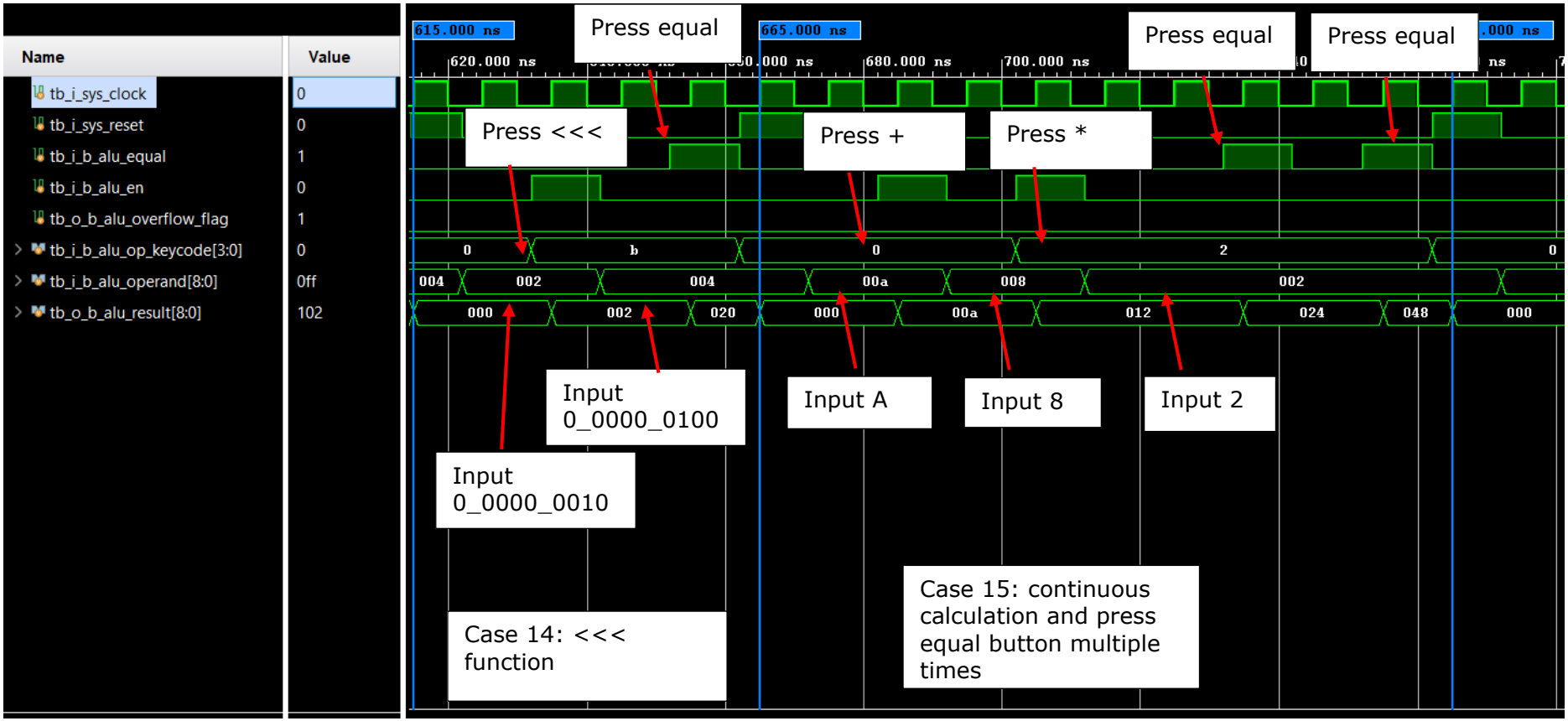

Figure 4.4.8.2.5 simulation result test case 14, 15

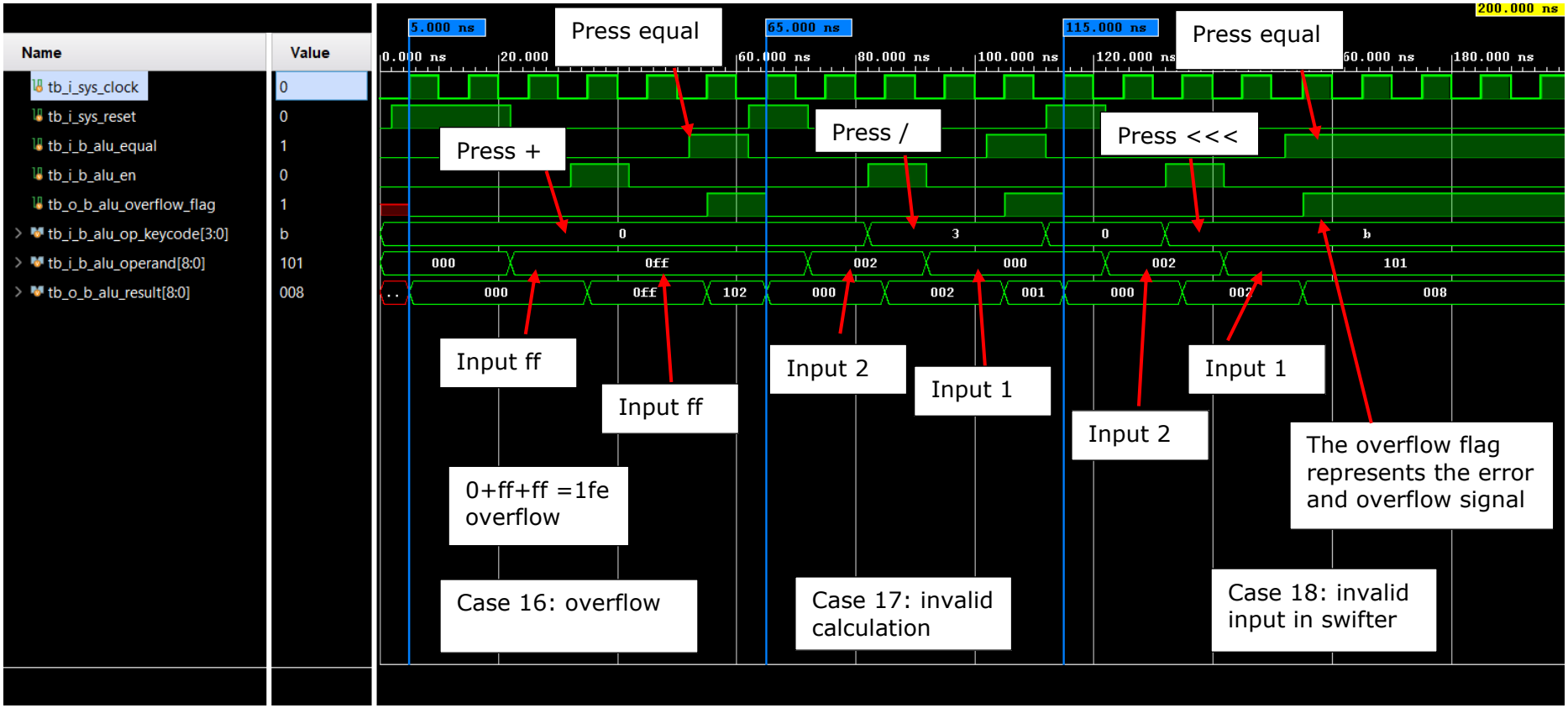

Figure 4.4.8.2.5 simulation result test case 16, 17, 18

4.5 7-segment display decoder

4.5.1 Functionality/features

- Converts binary output from the ALU calculator or Operator into Binary Coded Decimal
- Convert Binary Coded Decimal into displayable 7-segment codes
- Display error message if overflow or invalid
- Turn on or off each 7-segment display respectively

4.5.2 Block interface and I/O pin description

4.5.2.1 7-segment display decoder block interface

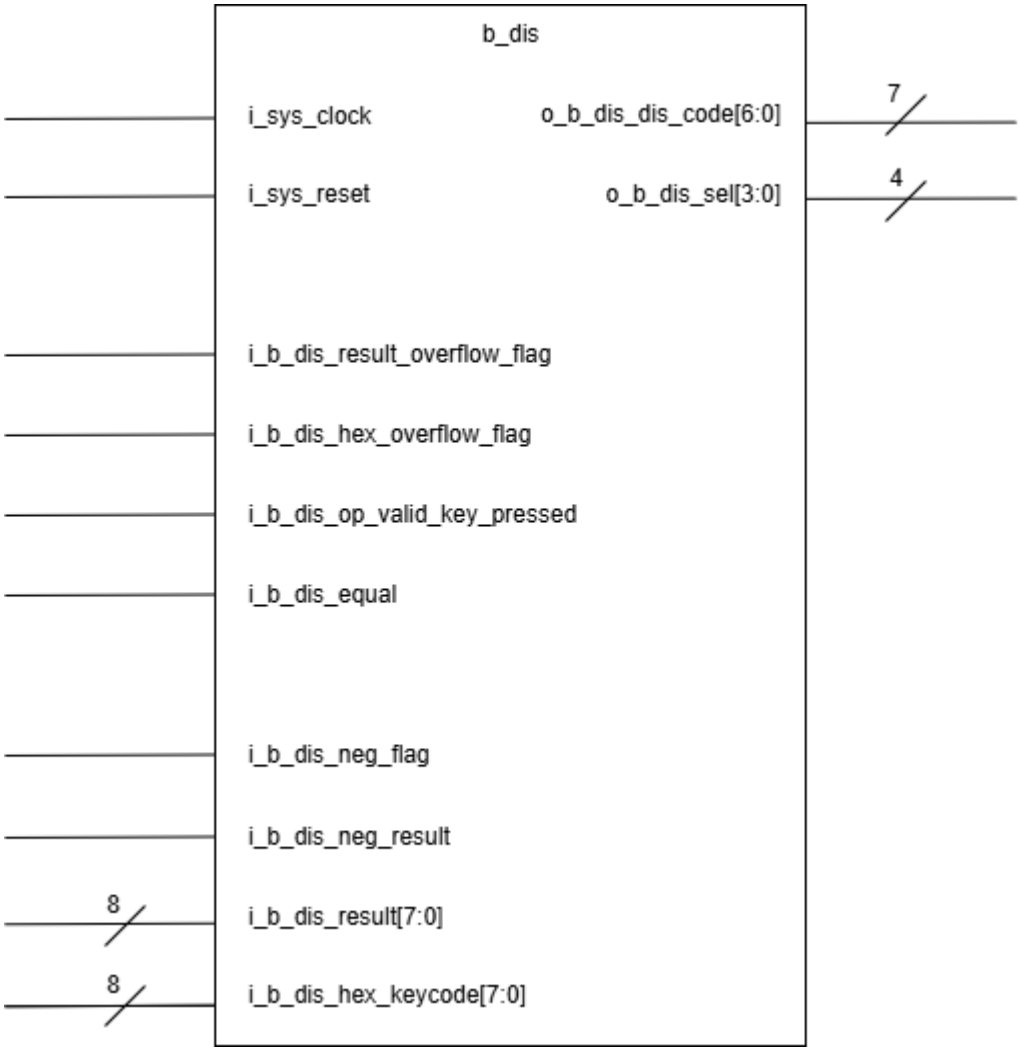

Diagram 4.5.2.1: Block interface of 7-segment display decoder

4.5.2.2 I/O pin description

Pin name: **Pin class:** **Pin function:**	i_sys_clock control provide a system clock to light up each 7-segment display respectively	**Source →Destination:**	Clock generator → 7-segment display decoder

Pin name: **Pin class:** **Pin function:**	i_sys_reset data provide reset signal for clearing all the 7-segment display	**Source →Destination:**	ON/Reset button → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_result_overflow_flag data provide signal to indicate if the output is overflow or not	**Source →Destination:**	ALU calculator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_hex_overflow_flag data provide signal to indicate if the input is overflow or not	**Source →Destination:**	Hex keypad code generator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_op_valid_key_pressed data provide signal to indicate if valid operation button is pressed	**Source →Destination:**	Operator keypad code generator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_equal data provide signal to indicate if equal button is pressed	**Source →Destination:**	Synchronizer → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_neg_flag data provide signal to indicate the output is negative	**Source →Destination:**	Operator keypad code generator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_neg_result data provide signal to indicate the output is negative	**Source →Destination:**	ALU calculator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_result[7:0] data get data from ALU calculator	**Source →Destination:**	ALU calculator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	i_b_dis_hex_keycode[7:0] data get data from operator keypad code generator	**Source →Destination:**	Hex keypad code generator → 7-segment display decoder
Pin name: **Pin class:** **Pin function:**	o_b_dis_dis_code[6:0] data provide result to display on 7-segment display	**Source →Destination:**	7-segment display decoder → 7 segment display
Pin name: **Pin class:** **Pin function:**	o_b_dis_sel[3:0] data to determine which 7_segment display is light up	**Source →Destination:**	7-segment display decoder → 7 segment display

Table 4.5.2.2: I/O pin description of 7-segment display decoder

4.5.3 Internal Operation: Function Table

Reset i_b_dis_reset	**Overflow i_b_dis_result_overflow_flag || i_b_dis_hex_overflow_flag**	**Operator i_b_dis_op_valid_hex_key_pressed || i_b_dis_equal**	**Negative i_b_dis_neg_flag || i_b_dis_neg_result**	**Display i_b_dis_hex_keycode [7:0]**	**Result i_b_dis_result [7:0]**	**7-segment display decoder function**

1	x	x	x	x	x	Clear all 7-segment display
0	1	x	x	x	x	Display "Err"
0	0	0	0	8'bxxxx xxxx	8'bxxxx xxxx	Display hex_key_code
0	0	0	1	8'bxxxx xxxx	8'bxxxx xxxx	Display negative hex_key_code
0	0	1	0	8'bxxxx xxxx	8'bxxxx xxxx	Display result
0	0	1	1	8'bxxxx xxxx	8'bxxxx xxxx	Display negative result

Table 4.5.3: Function table of 7-segment display decoder

4.5.4 Timing Requirement

Signal	**Requirement**
i_b_dis_result_overflow_flag	**1 clock cycle** -to trigger the error display
i_b_dis_hex_overflow_flag	**1 clock cycle** -to trigger the error display
i_b_dis_op_valid_key_pressed	**1 clock cycle** - indicate to display result
i_b_dis_equal	**1 clock cycle** - indicate to display result
i_b_dis_neg_flag	**1 clock cycle** -to indicate the input is negative
i_b_dis_neg_result	**1 clock cycle** -to indicate the result is negative
i_b_dis_result[7:0]	**1 clock cycle** -to receive data from ALU calculator
i_b_dis_hex_keycode[7:0]	**1 clock cycle** -to receive data from hex keypad code generator
o_b_dis_dis_code[6:0]	**1 clock cycle** -to pass the data to 7 segment display
o_b_dis_sel[3:0]	**1 clock cycle** -to pass the data to 7 segment display

Table 4.5.4: Timing requirements of 7-segment display decoder

4.5.5 Schematic Diagram

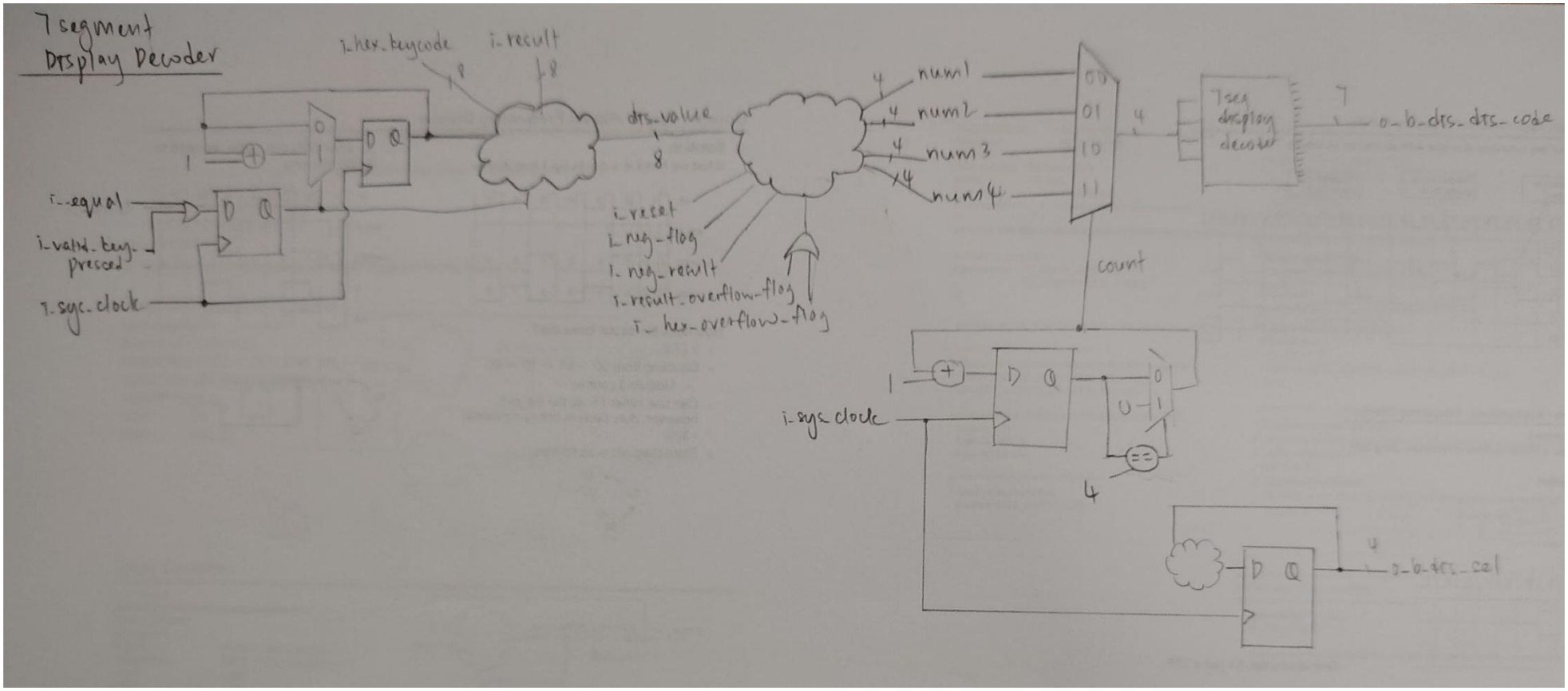

Figure 4.5.5.1 Micro architecture of 7-segment display decoder

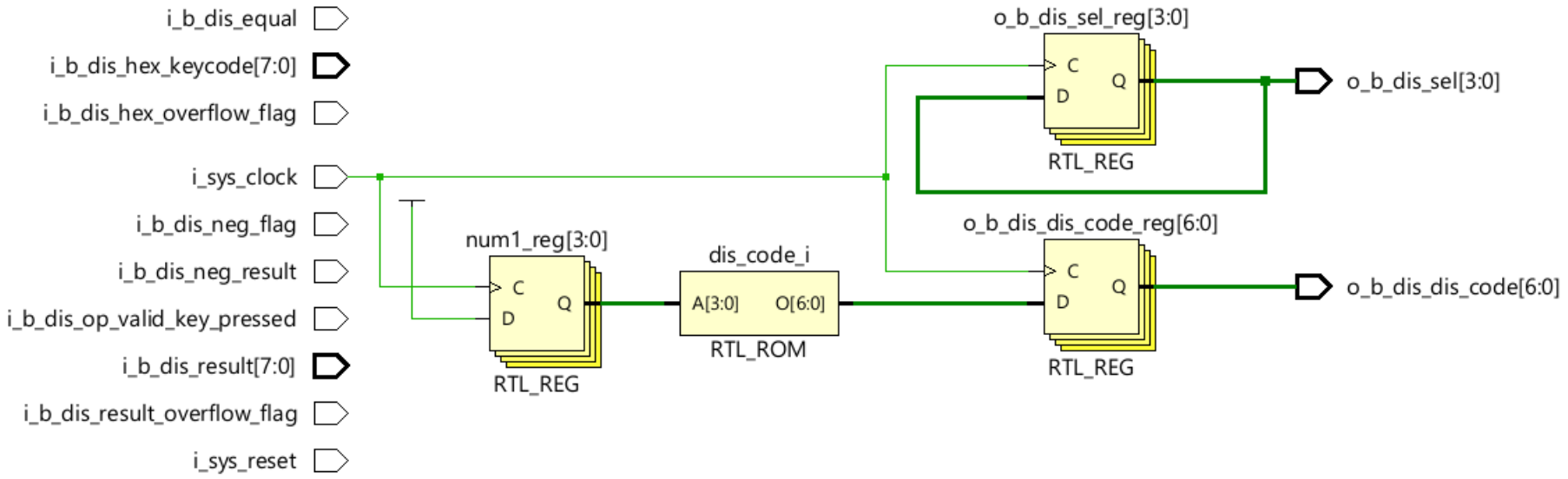

Figure 4.5.5.2 Schematic Diagram of 7-segment display decoder

4.5.6 System Verilog Model

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Jing Xuan
//
// Create Date: 20.04.2025 12:23:51
// File Name: b_dis.sv
// Module Name: b_dis
// Project Name: 8-bit integer calculator
// Code Type: RTL level
// Description: Modelling of 7-segment display decoder
//
//////////////////////////////////////////////////////////////////////////////////

module b_dis
(input logic i_sys_clock,
input logic i_sys_reset,
input logic i_b_dis_result_overflow_flag,
input logic i_b_dis_hex_overflow_flag,
input logic i_b_dis_op_valid_key_pressed,
input logic i_b_dis_equal,
input logic i_b_dis_neg_flag,
input logic i_b_dis_neg_result,
input logic [7:0] i_b_dis_result,
input logic [7:0] i_b_dis_hex_keycode,
output logic [6:0] o_b_dis_dis_code,
output logic [3:0] o_b_dis_sel);

logic [7:0] dis_value;
logic [7:0] dis_decimal;
logic [3:0] num1,num2,num3,num4;
logic [2:0] count = 3'b0;
logic keycode = 1'b0, result = 1'b0;  //use to determine whether the displayed output is
result or hex keypad input
logic [8:0] is_posedge = 1'b0;  //set as condition to change display value

initial begin  //initialization
   o_b_dis_sel = 4'b1000;
   o_b_dis_dis_code = 7'h00;
end

always @(posedge i_sys_clock) //change to select different 7-segement display to turn
on each cycle
   o_b_dis_sel <= {o_b_dis_sel[2:0],o_b_dis_sel[3]};


always @(posedge i_sys_clock)begin
   if(i_b_dis_equal || i_b_dis_op_valid_key_pressed)
   is_posedge += 1;
end


always @(i_b_dis_hex_keycode, i_b_dis_result,is_posedge) begin
   if (i_b_dis_equal || i_b_dis_op_valid_key_pressed)begin
      dis_value = i_b_dis_result;
   result <= 1'b1;  //displayed output is result
```

```
    keycode <= 1'b0;
    end
    else begin  //if condition true display hex keycode value
    dis_value = i_b_dis_hex_keycode;
    keycode <= 1'b1;  //displayed output is hex keycode value
    result <= 1'b0;
    end
end


//function to convert binary to correct value for display
function [6:0] dis_code(input [3:0] num);
    case(num)
        4'b0000: dis_code = 7'h3F;  //"0"
        4'b0001: dis_code = 7'h06;  //"1"
        4'b0010: dis_code = 7'h5B;  //"2"
        4'b0011: dis_code = 7'h4F;  //"3"
        4'b0100: dis_code = 7'h66;  //"4"
        4'b0101: dis_code = 7'h6D;  //"5"
        4'b0110: dis_code = 7'h7D;  //"6"
        4'b0111: dis_code = 7'h07;  //"7"
        4'b1000: dis_code = 7'h7F;  //"8"
        4'b1001: dis_code = 7'h6F;  //"9"
        4'b1100: dis_code = 7'h40;  //"-" (negative)
        4'b1101: dis_code = 7'h50;  //"r"
        4'b1110: dis_code = 7'h79;  //"E"
        4'b1111: dis_code = 7'h00;  //" " (blank)
        default: dis_code = 7'h00;  //" " (blank)
    endcase
endfunction


always @(posedge i_sys_clock)begin
    num1 <= 4'b1111;  //initialize to blank
    num2 <= 4'b1111;
    num3 <= 4'b1111;
    num4 <= 4'b1111;
    dis_decimal = dis_value;

    if(i_sys_reset) //display "0" during reset
        num1 = 4'b0000;
    else if (i_b_dis_result_overflow_flag || i_b_dis_hex_overflow_flag)begin //display
"Err"
        num1 = 4'b1101;
        num2 = 4'b1101;
        num3 = 4'b1110;
    end
    else begin
        if ((keycode && i_b_dis_neg_flag) || (result && i_b_dis_neg_result))  //display "-"
            num4 = 4'b1100;

        num3 = dis_decimal / 100;  //get the value for each digit
        dis_decimal %= 100;
        num2 = dis_decimal / 10;
        num1 = dis_decimal % 10;
```

```
      if(num2 == 0 && num3 == 0)begin //display blank for num2 and num3 if it is
single digit only
         num2 = 4'b1111;
         num3 = 4'b1111;
      end
      else if(num3 == 0)  //display blank for num3 if it is double digit only
         num3 = 4'b1111;
   end
end


always @(posedge i_sys_clock)begin  //call function to get the correct value to display
case(count)
   3'b000: o_b_dis_dis_code <= dis_code(num1);
   3'b001: o_b_dis_dis_code <= dis_code(num2);
   3'b010: o_b_dis_dis_code <= dis_code(num3);
   3'b011: o_b_dis_dis_code <= dis_code(num4);
   default: o_b_dis_dis_code <= dis_code(num4);
endcase

   count++;
   if (count ==4)  //reset to 0 for looping
   count <= 0;
end
endmodule
```

4.5.7 Test Plan

- The output will set only one 7 segment display to light up around 10ns each time (triggered by clock).
- It will look like all 7-segment display is turned on as the speed is very fast

No.	Test Case	Description of test vector Generation	Expected Output	Status
1	No input	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b0 9. Set i_b_dis_hex_keycode[7:0] = 8'b0	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 0)	PASS
2	Reset is	1. Set i_sys_reset = 1	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001	PASS

	triggered	2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b0 9. Set i_b_dis_hex_keycode[7:0] = 8'b0	o_b_dis_dis_code = 7'h3F 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h00 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 0)	
3	Result overflow	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 1 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b0 9. Set i_b_dis_hex_keycode[7:0] = 8'b0	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h50 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h79 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS
4	Hex input overflow	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 1 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b0 9. Set i_b_dis_hex_keycode[7:0] = 8'b0	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h50 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h50 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h79 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display Err)	PASS

5	Display value pressed at hex keypad	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b0 9. Set i_b_dis_hex_keycode[7:0] = 8'b01100100	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h3F 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h06 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 100)	PASS
6	Negative flag is triggered	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 1 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b0 9. Set i_b_dis_hex_keycode[7:0] = 8'b01100100	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h3F 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h3F 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h06 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display -100)	PASS
7	Display result	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 1 5. Set i_b_dis_equal = 0 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 0 8. Set i_b_dis_result[7:0] = 8'b01000010 9. Set i_b_dis_hex_keycode[7:0] = 8'b0	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h7D 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h7D 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h00 (display 66)	PASS

8	Display negative result	1. Set i_sys_reset = 0 2. Set i_b_dis_result_overflow_flag = 0 3. Set i_b_dis_hex_overflow_flag = 0 4. Set i_b_dis_op_valid_key_pressed = 0 5. Set i_b_dis_equal = 1 6. Set i_b_dis_neg_flag = 0 7. Set i_b_dis_neg_result = 1 8. Set i_b_dis_result[7:0] = 8'b01000010 9. Set i_b_dis_hex_keycode[7:0] = 8'b0	Loop output between different value continuously 1) o_b_dis_sel = 4'b0001 o_b_dis_dis_code = 7'h7D 2) o_b_dis_sel = 4'b0010 o_b_dis_dis_code = 7'h7D 3) o_b_dis_sel = 4'b0100 o_b_dis_dis_code = 7'h00 4) o_b_dis_sel = 4'b1000 o_b_dis_dis_code = 7'h40 (display - 66)	PASS

Table 4.5.5: Test plan of 7-segment display decoder

4.5.8 Testbench and Simulation result

4.5.8.1 Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Ng Jing Xuan
//
// Create Date: 20.04.2025 16:01:24
// File Name: tb_b_dis.sv
// Module Name: tb_b_dis
// Project Name: 8-bit integer calculator
// Code Type: Behavioural
// Description: Testbench for 7-segment display decoder
//
//////////////////////////////////////////////////////////////////////////////////
module tb_b_dis;

logic tb_sys_clock;
logic tb_sys_reset;
logic tb_b_dis_result_overflow_flag;
logic tb_b_dis_hex_overflow_flag;
logic tb_b_dis_op_valid_key_pressed;
logic tb_b_dis_equal;
logic tb_b_dis_neg_flag;
logic tb_b_dis_neg_result;
logic [7:0] tb_b_dis_result;
logic [7:0] tb_b_dis_hex_keycode;
wire [6:0] tb_b_dis_dis_code;
wire [3:0] tb_b_dis_sel;

b_dis dut_b_dis
(.i_sys_clock(tb_sys_clock),
.i_sys_reset(tb_sys_reset),
```

```
.i_b_dis_result_overflow_flag(tb_b_dis_result_overflow_flag),
.i_b_dis_hex_overflow_flag(tb_b_dis_hex_overflow_flag),
.i_b_dis_op_valid_key_pressed(tb_b_dis_op_valid_key_pressed),
.i_b_dis_equal(tb_b_dis_equal),
.i_b_dis_neg_flag(tb_b_dis_neg_flag),
.i_b_dis_neg_result(tb_b_dis_neg_result),
.i_b_dis_result(tb_b_dis_result),
.i_b_dis_hex_keycode(tb_b_dis_hex_keycode),
.o_b_dis_dis_code(tb_b_dis_dis_code),
.o_b_dis_sel(tb_b_dis_sel));

initial begin
tb_sys_clock = 1'b1;
forever #5 tb_sys_clock = ~tb_sys_clock;
end


//Test case 1 (No input)
initial begin
tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b0;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b0;
tb_b_dis_hex_keycode = 8'b0;


//Test case 2 (Reset is triggered)
#80tb_sys_reset = 1'b1;
tb_b_dis_result_overflow_flag = 1'b0;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b0;
tb_b_dis_hex_keycode = 8'b0;


//Test case 3 (Result overflow)
#40tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b1;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b0;
tb_b_dis_hex_keycode = 8'b0;


//Test case 4 (Hex input overflow)
#40tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b0;
```

```
tb_b_dis_hex_overflow_flag = 1'b1;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b0;
tb_b_dis_hex_keycode = 8'b0;


//Test case 5 (Display hex keypad value)
#40tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b0;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b0;
tb_b_dis_hex_keycode = 8'b01100100;


//Test case 6 (Display negative value)
#40tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b0;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b1;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b0;
tb_b_dis_hex_keycode = 8'b01100100;


//Test case 7 (Display result)
#40tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b0;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b1;
tb_b_dis_equal = 1'b0;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b0;
tb_b_dis_result = 8'b01000010;
tb_b_dis_hex_keycode = 8'b00000000;


//Test case 8 (Display negative result)
#40tb_sys_reset = 1'b0;
tb_b_dis_result_overflow_flag = 1'b0;
tb_b_dis_hex_overflow_flag = 1'b0;
tb_b_dis_op_valid_key_pressed = 1'b0;
tb_b_dis_equal = 1'b1;
tb_b_dis_neg_flag = 1'b0;
tb_b_dis_neg_result = 1'b1;
tb_b_dis_result = 8'b01000010;
tb_b_dis_hex_keycode = 8'b0;
end
```

```
initial #360 $stop;
endmodule
```

4.5.7.2 Simulation Result

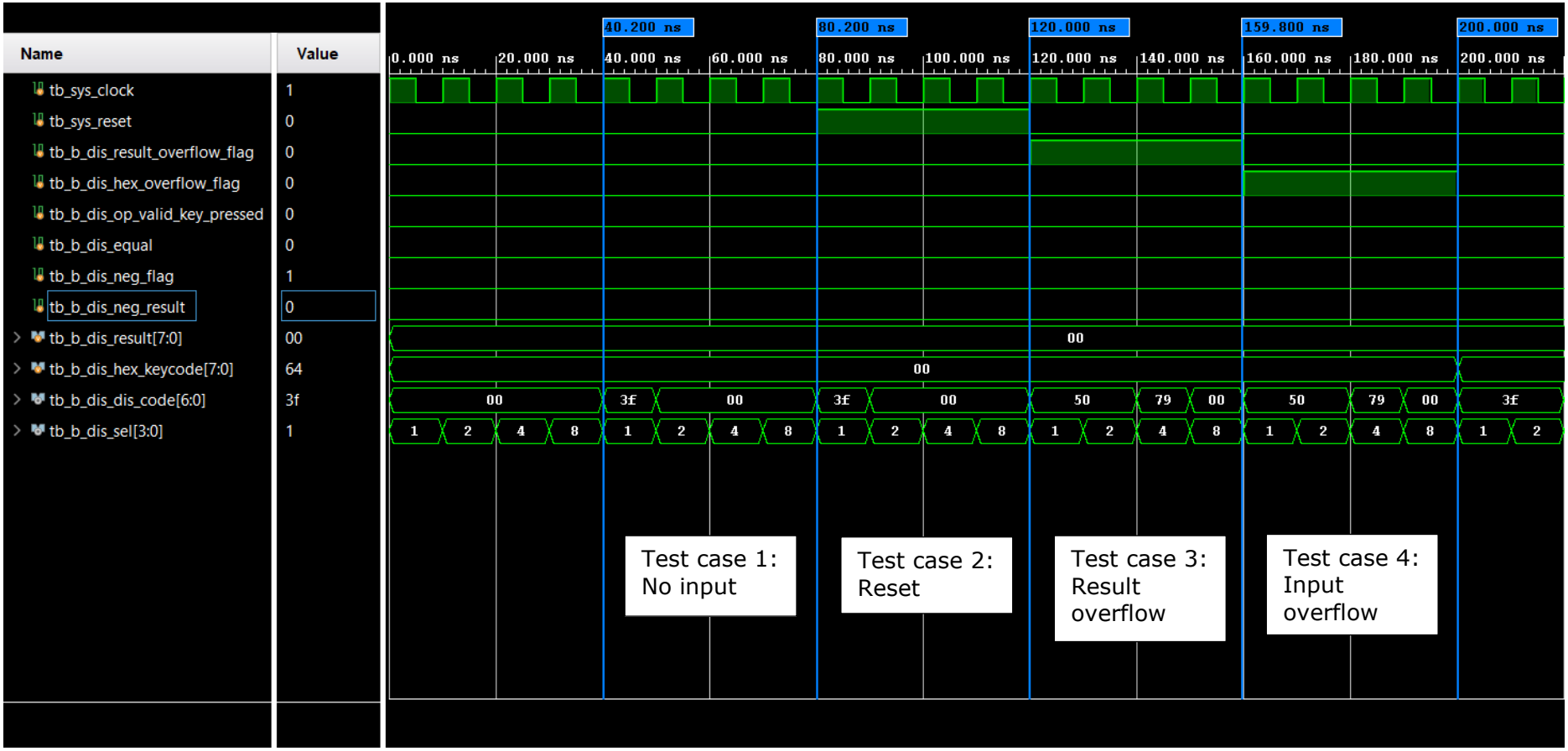

Figure 4.5.8.2.1 Test case 1,2,3,4

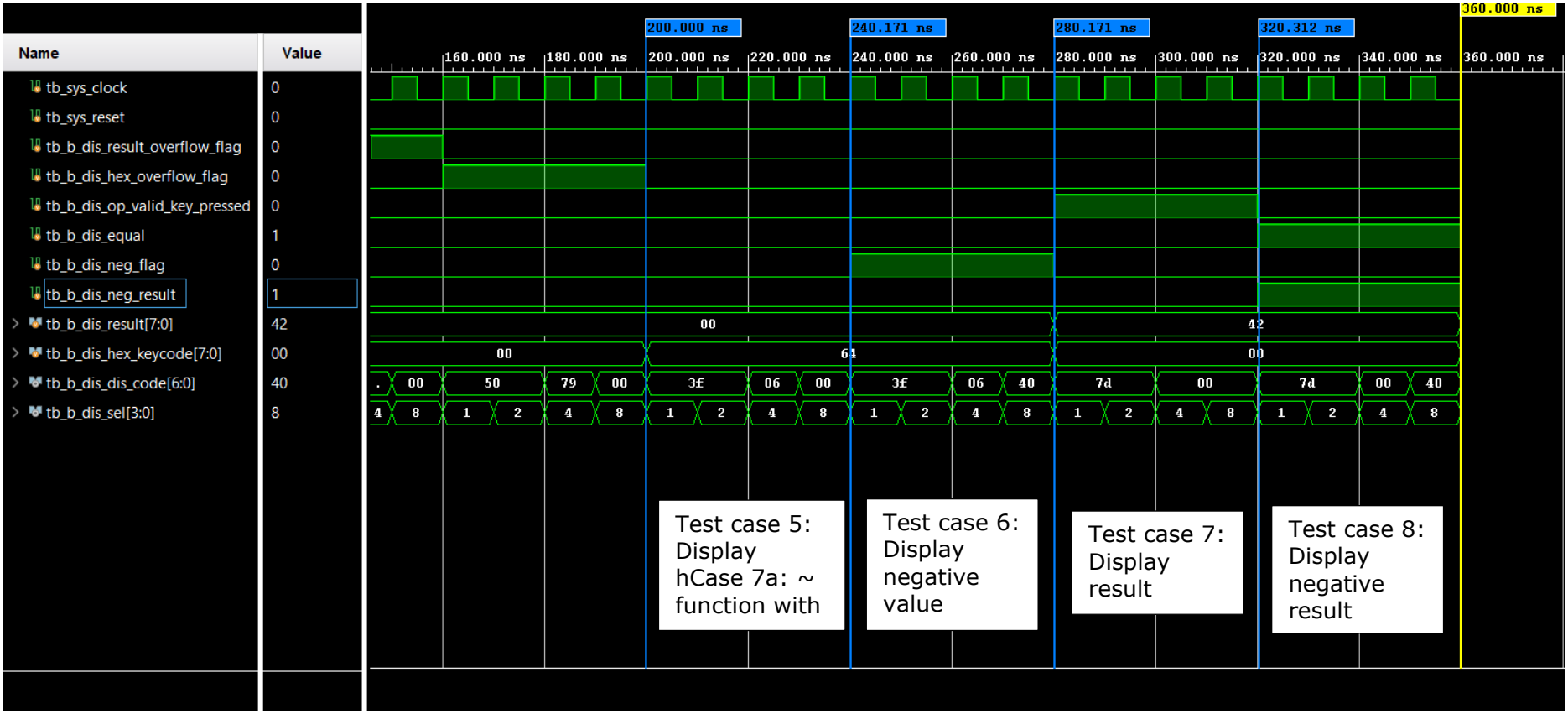

Figure 4.5.8.2.2 Test case 5,6,7,8

Reference
tb_b_dis_dis_code = 3F; display "0" on 7-segment display
tb_b_dis_dis_code = 06; display "1" on 7-segment display
tb_b_dis_dis_code = 5B; display "2" on 7-segment display
tb_b_dis_dis_code = 4F; display "3" on 7-segment display
tb_b_dis_dis_code = 66; display "4" on 7-segment display
tb_b_dis_dis_code = 6D; display "5" on 7-segment display
tb_b_dis_dis_code = 7D; display "6" on 7-segment display
tb_b_dis_dis_code = 07; display "7" on 7-segment display
tb_b_dis_dis_code = 7F; display "8" on 7-segment display
tb_b_dis_dis_code = 6F; display "9" on 7-segment display
tb_b_dis_dis_code = 40; display "-" (negative) on 7-segment display
tb_b_dis_dis_code = 50; display "r" on 7-segment display
tb_b_dis_dis_code = 79; display "E" on 7-segment display
tb_b_dis_dis_code = 00; display " " (blank) on 7-segment display