VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Engineering

LAB 2
INTRODUCTION TO SYSTEM ON CHIP
LABORATORY REPORT

Instructor: Pham Kieu Nhat Anh

Class: CC01
Name: Hoang Thuy Tram
Student ID: 2353202

Ho Chi Minh city, December 2025

Contents

1 Github

The Lab 2 project files (RTL source code, testbenches, simulation scripts) are located in the `Lab02` folder. They are available at the following repository: https://github.com/trxmhoang/SoC.git

2 Carry Look-ahead Adder (CLA)

2.1 RTL implementation

Program 1 presents the RTL implementation of a 32-bit carry Look-ahead Adder (CLA), a high-speed adder widely used in digital arithmetic circuits to efficiently compute sums of large binary numbers. The design is fully hierarchical and modular, utilizing generate and propagate logic for fast carry calculation.

- Module gp1: Computes generate (g) and propagate (p) signals for a single bit, the fundamental building block for the CLA logic.
- Module gp4: Aggregates generate and propagate signals over a 4-bit window. It determines the carry outputs for the lower three bits, as well as combined generate (gout) and propagate (pout) signals for the window.
- Module gp8: Extends the concept of gp4 to an 8-bit block by cascading two gp4 modules. It combines their generate and propagate outputs and produces intermediate carries for all bit positions in the block.
- Module cla: The top-level 32-bit carry look-ahead adder. It instantiates 32 gp1 modules to compute bitwise generate and propagate signals for all input bits of a and b. These are grouped into eight 4-bit groups, and each is managed by a gp4 module. The 4-bit groups are further grouped into higher-level sections and connected via a single gp8 module that establishes the block-level carry-ins. Carries for all bit positions are efficiently computed and used to calculate the final sum bits.

```
1  `timescale 1ns / 1ps
2
3  /**
4   * @param a first 1-bit input
5   * @param b second 1-bit input
6   * @param g whether a and b generate a carry
7   * @param p whether a and b would propagate an incoming carry
8   */
9  module gp1(input wire a, b,
10            output wire g, p);
11    assign g = a & b;
12    assign p = a | b;
13 endmodule
14
15 /**
16  * Computes aggregate generate/propagate signals over a 4-bit window.
17  * @param gin incoming generate signals
18  * @param pin incoming propagate signals
19  * @param cin the incoming carry
20  * @param gout whether these 4 bits internally would generate a carry-
      out (independent of cin)
```

```
 * @param pout whether these 4 bits internally would propagate an
    incoming carry from cin
 * @param cout the carry outs for the low-order 3 bits
 */
module gp4(input wire [3:0] gin, pin,
           input wire cin,
           output wire gout, pout,
           output wire [2:0] cout);

   // TODO: your code here
   wire [3:0] c;
   assign c[0] = cin;
   assign c[1] = gin[0] | (pin[0] & c[0]);
   assign c[2] = gin[1] | (pin[1] & gin[0]) | (pin[1] & pin[0] & c[0])
      ;
   assign c[3] = gin[2] | (pin[2] & gin[1]) | (pin[2] & pin[1] & gin
      [0]) | (pin[2] & pin[1] & pin[0] & c[0]);

   assign gout = gin[3] | (pin[3] & gin[2]) | (pin[3] & pin[2] & gin
      [1]) | (pin[3] & pin[2] & pin[1] & gin[0]);
   assign pout = &pin;
   assign cout = c[3:1];

endmodule

/** Same as gp4 but for an 8-bit window instead */
module gp8(input wire [7:0] gin, pin,
           input wire cin,
           output wire gout, pout,
           output wire [6:0] cout);

   // TODO: your code here
   wire gout_lo, pout_lo;
   wire gout_hi, pout_hi;
   wire c4;
   wire [2:0] cout_lo, cout_hi;

   gp4 gpa_lo (
      .gin(gin[3:0]),
      .pin(pin[3:0]),
      .cin(cin),
      .gout(gout_lo),
      .pout(pout_lo),
      .cout(cout_lo)
   );

   assign c4 = gout_lo | (pout_lo & cin);

   gp4 gpa_hi (
      .gin(gin[7:4]),
      .pin(pin[7:4]),
      .cin(c4),
      .gout(gout_hi),
```

```
70        .pout(pout_hi),
71        .cout(cout_hi)
72    );
73
74    assign gout = gout_hi | (pout_hi & gout_lo);
75    assign pout = pout_hi & pout_lo;
76    assign cout = {cout_hi, c4, cout_lo};
77
78 endmodule
79
80 module cla
81   (input wire [31:0]  a, b,
82    input wire         cin,
83    output wire [31:0] sum);
84
85    // TODO: your code here
86    wire [31:0] g, p;
87    genvar i;
88    generate
89       for(i = 0; i < 32; i = i + 1) begin : gp1_loop
90          gp1 u_gp1 (
91             .a(a[i]),
92             .b(b[i]),
93             .g(g[i]),
94             .p(p[i])
95          );
96       end
97    endgenerate
98
99    wire [7:0] gout, pout;
100   wire [7:0] c_in;
101   wire [23:0] c_out;
102   wire unused_gout, unused_pout;
103
104   assign c_in[0] = cin;
105   gp8 u_gp8 (
106      .gin(gout),
107      .pin(pout),
108      .cin(cin),
109      .gout(unused_gout),
110      .pout(unused_pout),
111      .cout(c_in[7:1])
112   );
113
114   generate
115      for(i = 0; i < 8; i = i + 1) begin : gp4_loop
116         gp4 u_gp4 (
117            .gin(g[4*i + 3 : 4*i]),
118            .pin(p[4*i + 3 : 4*i]),
119            .cin(c_in[i]),
120            .gout(gout[i]),
121            .pout(pout[i]),
122            .cout(c_out[3*i + 2 : 3*i])
```

```
        );
    end
endgenerate

wire [31:0] full_cout;
generate
    for(i = 0; i < 8; i = i + 1) begin : cout_loop
        assign full_cout[4*i] = c_in[i];
        assign full_cout[4*i + 3 : 4*i + 1] = c_out[3*i + 2 : 3*i];
    end
endgenerate

assign sum = a ^ b ^ full_cout;
endmodule
```

Program 1: RTL implementation of the CLA module.

Program 2 implements the system module, which uses a 32-bit carry look-ahead adder to sum the constant value 26 with the 4-bit input `btn`. The resulted sum is then output to the 6-bit led signal, allowing the LED output to reflect the combined value of the button inputs and the fixed base value, which is 26 in this program. Once the buttons are released, meaning their values are 0, the LED output shows the bit string of value 26.

```
module system (
    input wire [3:0] btn,
    output wire [5:0] led
);

wire [31:0] sum;
cla cla1 (
    .a (32'd26),
    .b ({28'b0, btn}),
    .cin (1'b0),
    .sum (sum)
);

assign led = sum[5:0];
endmodule
```

Program 2: RTL implementation of the system module.

2.2 Testbench

2.2.1 gp4 module testbench

This testbench is used to test the gp4 module, verigying its generate, propagate and carry-out logic. The testbench applies various input patterns to the module to ensure correctness of the 4-bit carry look-ahead logic.

```
`timescale 1ns / 1ps
module tb_gp4;
reg [3:0] gin, pin;
reg cin;
```

```
wire gout, pout;
wire [2:0] cout;

integer pass, fail;

gp4 dut (
   .gin(gin),
   .pin(pin),
   .cin(cin),
   .gout(gout),
   .pout(pout),
   .cout(cout)
);

task divi;
    $display ("
       ------------------------------------------------------------
       --------------");
endtask

task msg (input [700:0] txt);
    begin
        divi();
        $display ("%0s", txt);
        divi();
    end
endtask

task test (input [3:0] tg, input [3:0] tp, input tcin);
    begin
        divi();
        $display ("[TEST]   Time = %0t | gin = 4'b%b, pin = 4'b%b, cin
            = %b", $time, tg, tp, tcin);
        divi();

        gin = tg;
        pin = tp;
        cin = tcin;
        #10;
    end
endtask

task check (input exp_gout, input exp_pout, input [2:0] exp_cout);
    begin
        $display ("[OUTPUT] Time = %0t | gout = %b, pout = %b, cout =
           3'b%b", $time, gout, pout, cout);
        $display ("[EXPECT] Time = %0t | gout = %b, pout = %b, cout =
           3'b%b", $time, exp_gout, exp_pout, exp_cout);

        if ((gout === exp_gout) && (pout === exp_pout) && (cout ===
           exp_cout)) begin
            $display ("======> PASSED");
            pass = pass + 1;
```

```
52          end else begin
53              $display ("======> FAILED");
54              fail = fail + 1;
55          end
56      end
57  endtask
58
59  initial begin
60      pass = 0;
61      fail = 0;
62      #10;
63
64      test (0, 0, 0);
65      check (0, 0, 0);
66
67      test (8, 0, 0);
68      check (1, 0, 0);
69
70      test (0, 4'b1111, 1);
71      check (0, 1, 3'b111);
72
73      test (0, 4'b1111, 0);
74      check (0, 1, 0);
75
76      test (4'b1111, 4'b1111, 1);
77      check (1, 1, 3'b111);
78
79      test (4'b1010, 4'b0101, 0);
80      check (1, 0, 3'b110);
81
82      test (4'b1100, 4'b0011, 1);
83      check (1, 0, 3'b111);
84
85      test (4'b0110, 4'b1001, 0);
86      check (1, 0, 3'b110);
87
88      test (4'b0011, 4'b1100, 1);
89      check (1, 0, 3'b111);
90
91      test (4'b1001, 4'b0110, 0);
92      check (1, 0, 3'b111);
93
94      test (4'b0101, 4'b1010, 1'b1);
95      check (1, 0, 3'b111);
96
97      test (4'b1110, 4'b0111, 0);
98      check (1, 0, 3'b110);
99
100     test (4'b0001, 4'b1110, 1);
101     check (1, 0, 3'b111);
102
103     msg ("SUMMARY");
104     $display ("TOTAL TESTS : %0d", pass + fail);
```

```
105     $display ("tOTAL PASSED: %0d", pass);
106     $display ("TOTAL FAILED: %0d", fail);
107 
108     if (fail == 0)
109         $display ("======> ALL TESTS PASSED");
110     else if (pass == 0)
111         $display ("======> ALL TESTS FAILED");
112     else
113         $display ("======> SOME TESTS FAILED");
114 
115     #100;
116     $finish;
117 end
118 endmodule
```

Program 3: Testbench implementation for the gp4 module.

2.2.2 cla module testbench

This testbench tests the cla module. It applies a variety of input cases, including basic additions, edge cases and large operand values to check the computed sum, and reports on test results. The goal is to verify the correctness and robustness of the 32-bit CLA adder.

```
1  `timescale 1ns / 1ps
2  module tb_cla;
3  reg [31:0] a, b;
4  reg cin;
5  wire [31:0] sum;
6 
7  integer pass, fail;
8 
9  cla dut (
10    .a(a),
11    .b(b),
12    .cin(cin),
13    .sum(sum)
14 );
15 
16 task divi;
17     $display ("
          ------------------------------------------------------------
          --------------");
18 endtask
19 
20 task msg (input [700:0] txt);
21     begin
22         divi();
23         $display ("%0s", txt);
24         divi();
25     end
26 endtask
27 
28 task test (input [31:0] ta, input [31:0] tb, input tcin);
```

```
29      begin
30          divi();
31          $display ("[TEST]   Time = %0t | a = 32'h%h, b = 32'h%h, cin =
                %b", $time, ta, tb, tcin);
32          divi();
33
34          a = ta;
35          b = tb;
36          cin = tcin;
37          #10;
38      end
39  endtask
40
41  task check (input [31:0] exp_sum);
42      begin
43          $display ("[OUTPUT] Time = %0t | sum = 32'h%h", $time, sum);
44          $display ("[EXPECT] Time = %0t | sum = 32'h%h", $time, exp_sum
               );
45
46          if (sum === exp_sum) begin
47              $display ("======> PASSED");
48              pass = pass + 1;
49          end else begin
50              $display ("======> FAILED");
51              fail = fail + 1;
52          end
53      end
54  endtask
55
56  initial begin
57      pass = 0;
58      fail = 0;
59      #10;
60
61      test (32'd15, 32'd10, 1'b0);
62      check (32'd25);
63
64      test (32'd100, 32'd200, 1'b1);
65      check (32'd301);
66
67      test (32'd0, 32'd0, 1'b0);
68      check (32'd0);
69
70      test (32'd0, 32'd0, 1'b1);
71      check (32'd1);
72
73      test (32'd1, 32'd0, 1'b0);
74      check (32'd1);
75
76      test (32'd1, 32'd1, 1'b0);
77      check (32'd2);
78
79      test (32'd1, 32'd1, 1'b1);
```

```
80      check (32'd3);
81
82      test (32'hFFFF_FFFF, 32'd0, 1'b1);
83      check (32'd0);
84
85      test (32'hFFFF_FFFF, 32'd1, 1'b0);
86      check (32'd0);
87
88      test (32'hAAAA_AAAA, 32'h5555_5555, 1'b0);
89      check (32'hFFFF_FFFF);
90
91      test (32'hAAAA_AAAA, 32'h5555_5555, 1'b1);
92      check (32'h0);
93
94      test (32'h0000_0001, 32'hFFFF_FFFF, 1'b0);
95      check (32'd0);
96
97      test (32'h0000_0001, 32'hFFFF_FFFF, 1'b1);
98      check (32'd1);
99
100     test (32'h8000_0000, 32'h8000_0000, 1'b0);
101     check (32'd0);
102
103     test (32'h7FFF_FFFF, 32'd1, 1'b0);
104     check (32'h8000_0000);
105
106     test (32'h1234_5678, 32'h8765_4321, 1'b0);
107     check (32'h9999_9999);
108
109     test (32'h1234_5678, 32'h8765_4321, 1'b1);
110     check (32'h9999_999A);
111
112     test (32'd1, 32'd1000000000, 1'b0);
113     check (32'd1000000001);
114
115     test (32'd2147483647, 32'd1, 1'b0);
116     check (32'd2147483648);
117
118     test (32'hF0F0_F0F0, 32'h0F0F_0F0F, 1'b0);
119     check (32'hFFFF_FFFF);
120
121     test (32'hF0F0_F0F0, 32'h0F0F_0F0F, 1'b1);
122     check (32'd0);
123
124     msg ("SUMMARY");
125     $display ("TOTAL TESTS : %0d", pass + fail);
126     $display ("tOTAL PASSED: %0d", pass);
127     $display ("TOTAL FAILED: %0d", fail);
128
129     if (fail == 0)
130         $display ("======> ALL TESTS PASSED");
131     else if (pass == 0)
132         $display ("======> ALL TESTS FAILED");
```

```
    else
        $display ("======> SOME TESTS FAILED");

    #100;
    $finish;
end
endmodule
```

Program 4: Testbench implementation for the cla module.

2.2.3 system module testbench

This testbench tests the system module, where it simulates all button input combinations, verifies that the LED output matches the expected sum of 26 pluses the button value. This ensures the correctness and reliability of the system module's functionality.

```
`timescale 1ns / 1ps
module tb_sys;
reg clk;
reg [3:0] btn;
wire [5:0] led;

integer i, pass, fail;

system dut (
   .btn (btn),
   .led (led)
);

task divi;
    $display ("
       ------------------------------------------------------------
       --------------");
endtask

task msg (input [700:0] txt);
    begin
        divi();
        $display ("%0s", txt);
        divi();
    end
endtask

task press (input [3:0] tbtn);
    begin
        divi();
        $display ("[TEST]    Time = %0t | btn = 4'b%b (%0d)", $time,
           tbtn, tbtn);
        divi();

        btn = tbtn;
        #10;
    end
```

```
35 endtask
36
37 task check (input [5:0] exp_led);
38     begin
39         $display ("[OUTPUT] Time = %0t | led = 6'b%b (%0d)", $time,
               led, led);
40         $display ("[EXPECT] Time = %0t | led = 6'b%b (%0d)", $time,
               exp_led, exp_led);
41
42         if (led === exp_led) begin
43             $display ("======> PASSED");
44             pass = pass + 1;
45         end else begin
46             $display ("======> FAILED");
47             fail = fail + 1;
48         end
49
50         btn = 4'b0000;
51         #10;
52     end
53 endtask
54
55 initial begin
56    clk = 0;
57    forever #5 clk = ~clk;
58 end
59
60 initial begin
61     btn = 4'b0000;
62     pass = 0;
63     fail = 0;
64     #15;
65
66     msg ("INITIAL TEST");
67     check (6'd26);
68
69     msg ("FUNCTIONAL TESTS");
70     for (i = 0; i < 16; i = i + 1) begin
71         press (i[3:0]);
72         check (6'd26 + i);
73     end
74
75     msg ("SUMMARY");
76     $display ("TOTAL TESTS : %0d", pass + fail);
77     $display ("tOTAL PASSED: %0d", pass);
78     $display ("TOTAL FAILED: %0d", fail);
79
80     if (fail == 0)
81         $display ("======> ALL TESTS PASSED");
82     else if (pass == 0)
83         $display ("======> ALL TESTS FAILED");
84     else
85         $display ("======> SOME TESTS FAILED");
```

```
86
87      #100;
88      $finish;
89  end
90  endmodule
```

Program 5: Testbench implementation for the system module

2.3 Testbench results

All test cases executed successfully for all tested modules, with each marked as `PASSED`. This outcome confirms that the test modules behave as intended and that the implemented RTL designs meets the expected functionality.

2.3.1 gp4 module testbench results

```
1  # ------------------------------------------------------------------
2  # [TEST]   Time = 10000 | gin = 4'b0000, pin = 4'b0000, cin = 0
3  # ------------------------------------------------------------------
4  # [OUTPUT] Time = 20000 | gout = 0, pout = 0, cout = 3'b000
5  # [EXPECT] Time = 20000 | gout = 0, pout = 0, cout = 3'b000
6  # ======> PASSED
7  # ------------------------------------------------------------------
8  # [TEST]   Time = 20000 | gin = 4'b1000, pin = 4'b0000, cin = 0
9  # ------------------------------------------------------------------
10 # [OUTPUT] Time = 30000 | gout = 1, pout = 0, cout = 3'b000
11 # [EXPECT] Time = 30000 | gout = 1, pout = 0, cout = 3'b000
12 # ======> PASSED
13 # ------------------------------------------------------------------
14 # [TEST]   Time = 30000 | gin = 4'b0000, pin = 4'b1111, cin = 1
15 # ------------------------------------------------------------------
16 # [OUTPUT] Time = 40000 | gout = 0, pout = 1, cout = 3'b111
17 # [EXPECT] Time = 40000 | gout = 0, pout = 1, cout = 3'b111
18 # ======> PASSED
19 # ------------------------------------------------------------------
20 # [TEST]   Time = 40000 | gin = 4'b0000, pin = 4'b1111, cin = 0
21 # ------------------------------------------------------------------
22 # [OUTPUT] Time = 50000 | gout = 0, pout = 1, cout = 3'b000
23 # [EXPECT] Time = 50000 | gout = 0, pout = 1, cout = 3'b000
24 # ======> PASSED
25 # ------------------------------------------------------------------
26 # [TEST]   Time = 50000 | gin = 4'b1111, pin = 4'b1111, cin = 1
27 # ------------------------------------------------------------------
28 # [OUTPUT] Time = 60000 | gout = 1, pout = 1, cout = 3'b111
29 # [EXPECT] Time = 60000 | gout = 1, pout = 1, cout = 3'b111
30 # ======> PASSED
31 # ------------------------------------------------------------------
32 # [TEST]   Time = 60000 | gin = 4'b1010, pin = 4'b0101, cin = 0
33 # ------------------------------------------------------------------
34 # [OUTPUT] Time = 70000 | gout = 1, pout = 0, cout = 3'b110
35 # [EXPECT] Time = 70000 | gout = 1, pout = 0, cout = 3'b110
36 # ======> PASSED
37 # ------------------------------------------------------------------
```

```
# [TEST]   Time = 70000 | gin = 4'b1100, pin = 4'b0011, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 80000 | gout = 1, pout = 0, cout = 3'b111
# [EXPECT] Time = 80000 | gout = 1, pout = 0, cout = 3'b111
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 80000 | gin = 4'b0110, pin = 4'b1001, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 90000 | gout = 1, pout = 0, cout = 3'b110
# [EXPECT] Time = 90000 | gout = 1, pout = 0, cout = 3'b110
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 90000 | gin = 4'b0011, pin = 4'b1100, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 100000 | gout = 1, pout = 0, cout = 3'b111
# [EXPECT] Time = 100000 | gout = 1, pout = 0, cout = 3'b111
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 100000 | gin = 4'b1001, pin = 4'b0110, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 110000 | gout = 1, pout = 0, cout = 3'b111
# [EXPECT] Time = 110000 | gout = 1, pout = 0, cout = 3'b111
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 110000 | gin = 4'b0101, pin = 4'b1010, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 120000 | gout = 1, pout = 0, cout = 3'b111
# [EXPECT] Time = 120000 | gout = 1, pout = 0, cout = 3'b111
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 120000 | gin = 4'b1110, pin = 4'b0111, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 130000 | gout = 1, pout = 0, cout = 3'b110
# [EXPECT] Time = 130000 | gout = 1, pout = 0, cout = 3'b110
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 130000 | gin = 4'b0001, pin = 4'b1110, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 140000 | gout = 1, pout = 0, cout = 3'b111
# [EXPECT] Time = 140000 | gout = 1, pout = 0, cout = 3'b111
# ======> PASSED
# ------------------------------------------------------------------
# SUMMARY
# ------------------------------------------------------------------
# TOTAL TESTS : 13
# tOTAL PASSED: 13
# TOTAL FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../../tb/tb_gp4.v(116)
#    Time: 240 ns  Iteration: 0  Instance: /tb_gp4
```

Simulation 1: Testbench results of the gp4 module.

2.3.2 cla module testbench results

```
# ------------------------------------------------------------------
# [TEST]   Time = 10000 | a = 32'h0000000f, b = 32'h0000000a, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 20000 | sum = 32'h00000019
# [EXPECT] Time = 20000 | sum = 32'h00000019
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 20000 | a = 32'h00000064, b = 32'h000000c8, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 30000 | sum = 32'h0000012d
# [EXPECT] Time = 30000 | sum = 32'h0000012d
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 30000 | a = 32'h00000000, b = 32'h00000000, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 40000 | sum = 32'h00000000
# [EXPECT] Time = 40000 | sum = 32'h00000000
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 40000 | a = 32'h00000000, b = 32'h00000000, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 50000 | sum = 32'h00000001
# [EXPECT] Time = 50000 | sum = 32'h00000001
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 50000 | a = 32'h00000001, b = 32'h00000000, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 60000 | sum = 32'h00000001
# [EXPECT] Time = 60000 | sum = 32'h00000001
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 60000 | a = 32'h00000001, b = 32'h00000001, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 70000 | sum = 32'h00000002
# [EXPECT] Time = 70000 | sum = 32'h00000002
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 70000 | a = 32'h00000001, b = 32'h00000001, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 80000 | sum = 32'h00000003
# [EXPECT] Time = 80000 | sum = 32'h00000003
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 80000 | a = 32'hffffffff, b = 32'h00000000, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 90000 | sum = 32'h00000000
# [EXPECT] Time = 90000 | sum = 32'h00000000
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 90000 | a = 32'hffffffff, b = 32'h00000001, cin = 0
# ------------------------------------------------------------------
```

```
# [OUTPUT] Time = 100000 | sum = 32'h00000000
# [EXPECT] Time = 100000 | sum = 32'h00000000
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 100000 | a = 32'haaaaaaaa, b = 32'h55555555, cin = 0
# ---------------------------------------------------------------------
# [OUTPUT] Time = 110000 | sum = 32'hffffffff
# [EXPECT] Time = 110000 | sum = 32'hffffffff
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 110000 | a = 32'haaaaaaaa, b = 32'h55555555, cin = 1
# ---------------------------------------------------------------------
# [OUTPUT] Time = 120000 | sum = 32'h00000000
# [EXPECT] Time = 120000 | sum = 32'h00000000
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 120000 | a = 32'h00000001, b = 32'hffffffff, cin = 0
# ---------------------------------------------------------------------
# [OUTPUT] Time = 130000 | sum = 32'h00000000
# [EXPECT] Time = 130000 | sum = 32'h00000000
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 130000 | a = 32'h00000001, b = 32'hffffffff, cin = 1
# ---------------------------------------------------------------------
# [OUTPUT] Time = 140000 | sum = 32'h00000001
# [EXPECT] Time = 140000 | sum = 32'h00000001
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 140000 | a = 32'h80000000, b = 32'h80000000, cin = 0
# ---------------------------------------------------------------------
# [OUTPUT] Time = 150000 | sum = 32'h00000000
# [EXPECT] Time = 150000 | sum = 32'h00000000
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 150000 | a = 32'h7fffffff, b = 32'h00000001, cin = 0
# ---------------------------------------------------------------------
# [OUTPUT] Time = 160000 | sum = 32'h80000000
# [EXPECT] Time = 160000 | sum = 32'h80000000
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 160000 | a = 32'h12345678, b = 32'h87654321, cin = 0
# ---------------------------------------------------------------------
# [OUTPUT] Time = 170000 | sum = 32'h99999999
# [EXPECT] Time = 170000 | sum = 32'h99999999
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 170000 | a = 32'h12345678, b = 32'h87654321, cin = 1
# ---------------------------------------------------------------------
# [OUTPUT] Time = 180000 | sum = 32'h9999999a
# [EXPECT] Time = 180000 | sum = 32'h9999999a
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 180000 | a = 32'h00000001, b = 32'h3b9aca00, cin = 0
```

```
# ------------------------------------------------------------------
# [OUTPUT] Time = 190000 | sum = 32'h3b9aca01
# [EXPECT] Time = 190000 | sum = 32'h3b9aca01
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 190000 | a = 32'h7fffffff, b = 32'h00000001, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 200000 | sum = 32'h80000000
# [EXPECT] Time = 200000 | sum = 32'h80000000
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 200000 | a = 32'hf0f0f0f0, b = 32'h0f0f0f0f, cin = 0
# ------------------------------------------------------------------
# [OUTPUT] Time = 210000 | sum = 32'hffffffff
# [EXPECT] Time = 210000 | sum = 32'hffffffff
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 210000 | a = 32'hf0f0f0f0, b = 32'h0f0f0f0f, cin = 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 220000 | sum = 32'h00000000
# [EXPECT] Time = 220000 | sum = 32'h00000000
# ======> PASSED
# ------------------------------------------------------------------
# SUMMARY
# ------------------------------------------------------------------
# TOTAL TESTS : 21
# tOTAL PASSED: 21
# TOTAL FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../../tb/tb_cla.v(137)
#    Time: 320 ns  Iteration: 0  Instance: /tb_cla
```

Simulation 2: Testbench results of the cla module.

2.3.3 system module testbench results

```
# ------------------------------------------------------------------
# INITIAL TEST
# ------------------------------------------------------------------
# [OUTPUT] Time = 15000 | led = 6'b011010 (26)
# [EXPECT] Time = 15000 | led = 6'b011010 (26)
# ======> PASSED
# ------------------------------------------------------------------
# FUNCTIONAL TESTS
# ------------------------------------------------------------------
# ------------------------------------------------------------------
# [TEST]   Time = 25000 | btn = 4'b0000 (0)
# ------------------------------------------------------------------
# [OUTPUT] Time = 35000 | led = 6'b011010 (26)
# [EXPECT] Time = 35000 | led = 6'b011010 (26)
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 45000 | btn = 4'b0001 (1)
# ------------------------------------------------------------------
```

```
# [OUTPUT] Time = 55000 | led = 6'b011011 (27)
# [EXPECT] Time = 55000 | led = 6'b011011 (27)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 65000 | btn = 4'b0010 (2)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 75000 | led = 6'b011100 (28)
# [EXPECT] Time = 75000 | led = 6'b011100 (28)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 85000 | btn = 4'b0011 (3)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 95000 | led = 6'b011101 (29)
# [EXPECT] Time = 95000 | led = 6'b011101 (29)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 105000 | btn = 4'b0100 (4)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 115000 | led = 6'b011110 (30)
# [EXPECT] Time = 115000 | led = 6'b011110 (30)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 125000 | btn = 4'b0101 (5)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 135000 | led = 6'b011111 (31)
# [EXPECT] Time = 135000 | led = 6'b011111 (31)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 145000 | btn = 4'b0110 (6)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 155000 | led = 6'b100000 (32)
# [EXPECT] Time = 155000 | led = 6'b100000 (32)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 165000 | btn = 4'b0111 (7)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 175000 | led = 6'b100001 (33)
# [EXPECT] Time = 175000 | led = 6'b100001 (33)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 185000 | btn = 4'b1000 (8)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 195000 | led = 6'b100010 (34)
# [EXPECT] Time = 195000 | led = 6'b100010 (34)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 205000 | btn = 4'b1001 (9)
# ---------------------------------------------------------------------
# [OUTPUT] Time = 215000 | led = 6'b100011 (35)
# [EXPECT] Time = 215000 | led = 6'b100011 (35)
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 225000 | btn = 4'b1010 (10)
```

```
# ------------------------------------------------------------------
# [OUTPUT] Time = 235000 | led = 6'b100100 (36)
# [EXPECT] Time = 235000 | led = 6'b100100 (36)
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 245000 | btn = 4'b1011 (11)
# ------------------------------------------------------------------
# [OUTPUT] Time = 255000 | led = 6'b100101 (37)
# [EXPECT] Time = 255000 | led = 6'b100101 (37)
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 265000 | btn = 4'b1100 (12)
# ------------------------------------------------------------------
# [OUTPUT] Time = 275000 | led = 6'b100110 (38)
# [EXPECT] Time = 275000 | led = 6'b100110 (38)
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 285000 | btn = 4'b1101 (13)
# ------------------------------------------------------------------
# [OUTPUT] Time = 295000 | led = 6'b100111 (39)
# [EXPECT] Time = 295000 | led = 6'b100111 (39)
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 305000 | btn = 4'b1110 (14)
# ------------------------------------------------------------------
# [OUTPUT] Time = 315000 | led = 6'b101000 (40)
# [EXPECT] Time = 315000 | led = 6'b101000 (40)
# ======> PASSED
# ------------------------------------------------------------------
# [TEST]   Time = 325000 | btn = 4'b1111 (15)
# ------------------------------------------------------------------
# [OUTPUT] Time = 335000 | led = 6'b101001 (41)
# [EXPECT] Time = 335000 | led = 6'b101001 (41)
# ======> PASSED
# ------------------------------------------------------------------
# SUMMARY
# ------------------------------------------------------------------
# TOTAL TESTS : 17
# tOTAL PASSED: 17
# TOTAL FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../../tb/tb_sys.v(88)
#    Time: 445 ns  Iteration: 0  Instance: /tb_sys
```

Simulation 3: Testbench results of the system module.

2.4 FPGA results

This section is part of the FPGA implementation. The input value for `a` is set to 26, while the input for `b` is determined by button presses, which correspond to a 4-bit binary value.

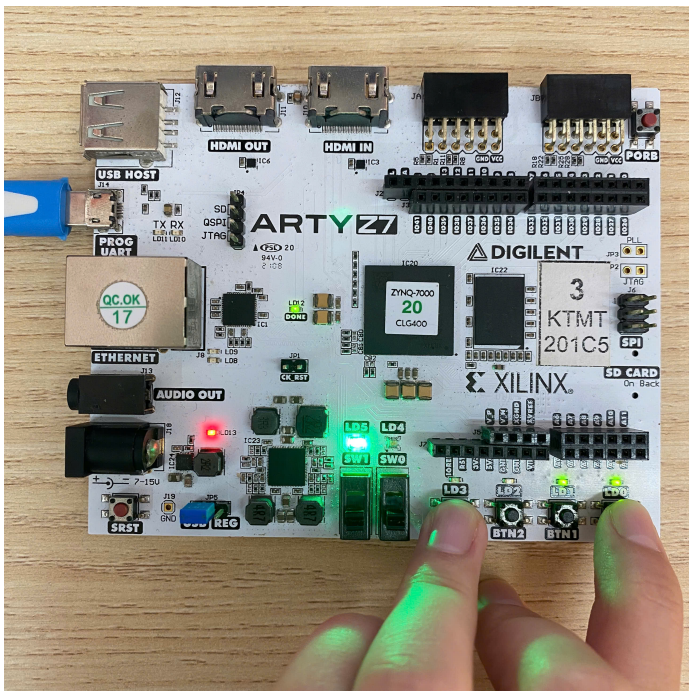

Figure 1.1

Figure 1.1 shows the input for `b` as 4'b1001, which corresponds to the decimal value 9. The resulting output is 6'b100011, equivalent to the decimal value 35. This confirms that the computation is correct.

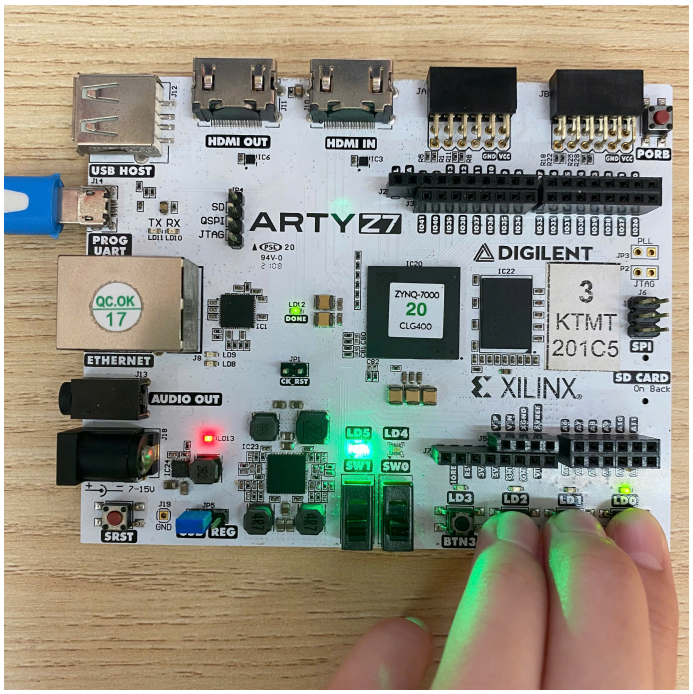

Figure 1.2

Figure 1.2 shows the input for `b` as 4'b0111, which corresponds to the decimal value 7. The resulting output is 6'b100001, equivalent to the decimal value 33. This confirms that the computation is correct.

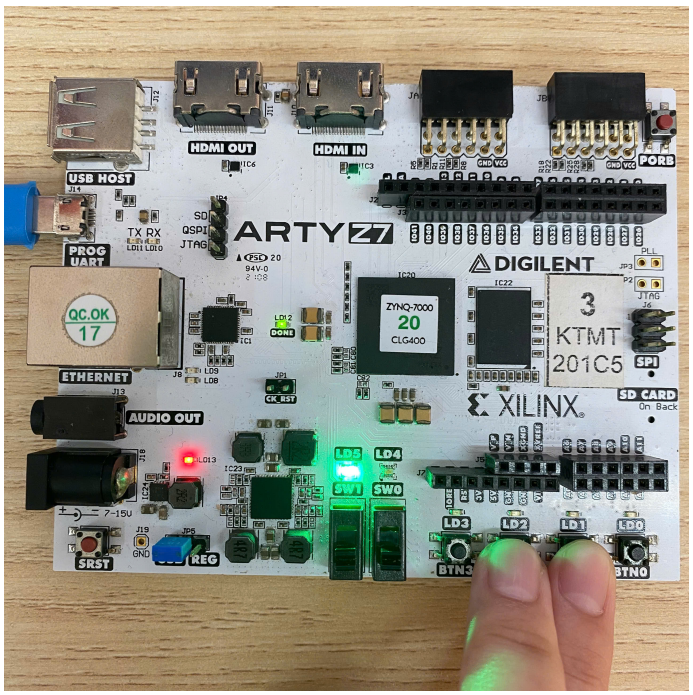

Figure 1.3

Figure 1.3 shows the input for `b` as 4'b0110, which corresponds to the decimal value 6. The resulting output is 6'b100000, equivalent to the decimal value 32. This confirms that the computation is correct.

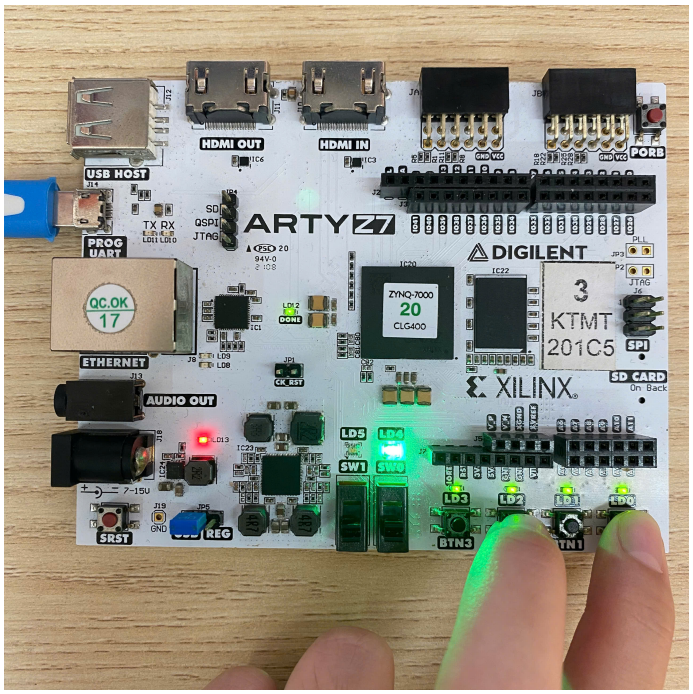

Figure 1.4

Figure 1.4 shows the input for `b` as 4'b0101, which corresponds to the decimal value 5. The resulting output is 6'b011111, equivalent to the decimal value 31. This confirms that the computation is correct.

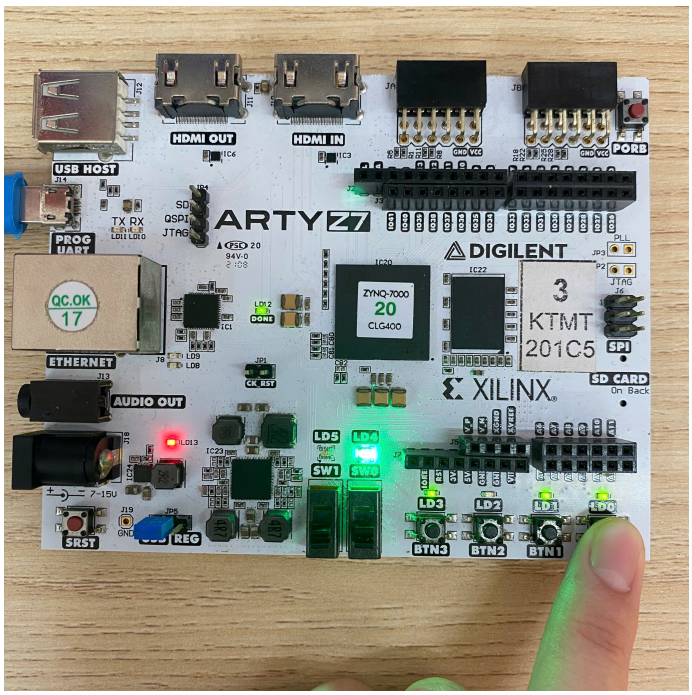

Figure 1.5

Figure 1.5 shows the input for `b` as 4'b0001, which corresponds to the decimal value 1. The resulting output is 6'b011011, equivalent to the decimal value 27. This confirms that the computation is correct.

3 Uppercase

3.1 Assembly implementation

Program 6 provides a RISC-V assembly routine that converts all lowercase ASCII letters in a string, such as "Hello world!" in this case, to uppercase, modifying the string in-place. The program iterates through each character in the string, checks if it is a lowercase letter, from 'a' to 'z', and if so, converts it to uppercase.

```
        .section .data
input_string:   .asciz "Hello world!"

        # tohost/fromhost are data locations used by Spike
        .align 8
tohost: .byte 0
        .align 8
fromhost: .byte 0

        .section .text
        .globl _start
_start:
        # Load the address of the input string into a0
        la a0, input_string
        # Your code here!
        li t1, 97
        li t2, 122
loop:
        lb t0, 0(a0)
        beqz t0, end_program
        blt t0, t1, next_char
        bgt t0, t2, next_char
        addi t0, t0, -32
        sb t0, 0(a0)

next_char:
        addi a0, a0, 1
        j loop

end_program:
        # jump here when you're done
        #j end_program
        li a1, 10
        ecall
```

Program 6: RISCV implementation of the uppercase exercise.

3.2 Simulation results

Address	Binary	Value
0x00200000 - 0x00200003	input_string 48 65 6C 6C	H, e, l, l
0x00200004 - 0x00200007	6F 20 77 6F	o, , w, o
0x00200008 - 0x0020000B	72 6C 64 21	r, l, d, !
0x0020000C - 0x0020000F	00 00 00 00	0
0x00200010 - 0x00200013	00 00 00 00	

Figure 2.1

Address	Binary	Value
0x00200000 - 0x00200003	54 68 65 20	T, h, e,
0x00200004 - 0x00200007	70 61 73 73	p, a, s, s
0x00200008 - 0x0020000B	77 6F 72 64	w, o, r, d
0x0020000C - 0x0020000F	20 69 73 3A	, i, s, :
0x00200010 - 0x00200013	20 31 32 33	, 1, 2, 3
0x00200014 - 0x00200017	34 68 4F 6D	4, h, O, m
0x00200018 - 0x0020001B	65 2E 00 00	e, ., 0

Figure 2.2

Address	Binary	Value
0x00200000 - 0x00200003	input_string 33 35 36 34	3, 5, 6, 4
0x00200004 - 0x00200007	48 6A 30 70	H, j, 0, p
0x00200008 - 0x0020000B	69 39 39 61	i, 9, 9, a
0x0020000C - 0x0020000F	66 00 00 00	f, 0

Figure 2.3

Figure 2.1 to 2.3 illustrate the ASCII input test strings used to verify the functionality of the `uppercase.S` file. The program was implemented on CreatorSim, a web-based RISC-V emulator:

- String 1: “Hello world!”
- String 2: “The password is: 1234hOme.”
- String 3: “3564Hj0pi99af”

Address	Binary	Value
0x00200000 - 0x00200003	input_string 48 45 4C 4C	H, E, L, L
0x00200004 - 0x00200007	4F 20 57 4F	O, , W, O
0x00200008 - 0x0020000B	52 4C 44 21	R, L, D, !
0x0020000C - 0x0020000F	00 00 00 00	0
0x00200010 - 0x00200013	00 00 00 00	

Figure 2.4

Address	Binary	Value
0x00200000 - 0x00200003	input_string 54 48 45 20	T, H, E,
0x00200004 - 0x00200007	50 41 53 53	P, A, S, S
0x00200008 - 0x0020000B	57 4F 52 44	W, O, R, D
0x0020000C - 0x0020000F	20 49 53 3A	, I, S, :
0x00200010 - 0x00200013	20 31 32 33	, 1, 2, 3
0x00200014 - 0x00200017	34 48 4F 4D	4, H, O, M
0x00200018 - 0x0020001B	45 2E 00 00	E, ., 0

Figure 2.5

Address	Binary	Value
0x00200000 - 0x00200003	input_string 33 35 36 34	3, 5, 6, 4
0x00200004 - 0x00200007	48 4A 30 50	H, J, 0, P
0x00200008 - 0x0020000B	49 39 39 41	I, 9, 9, A
0x0020000C - 0x0020000F	46 00 00 00	F, 0

Figure 2.6

Figure 2.4 to 2.6 demonstrate the outputs after executing the `uppercase.S` file. The results confirm that the code correctly converts all lowercase characters into their corresponding uppercase characters.