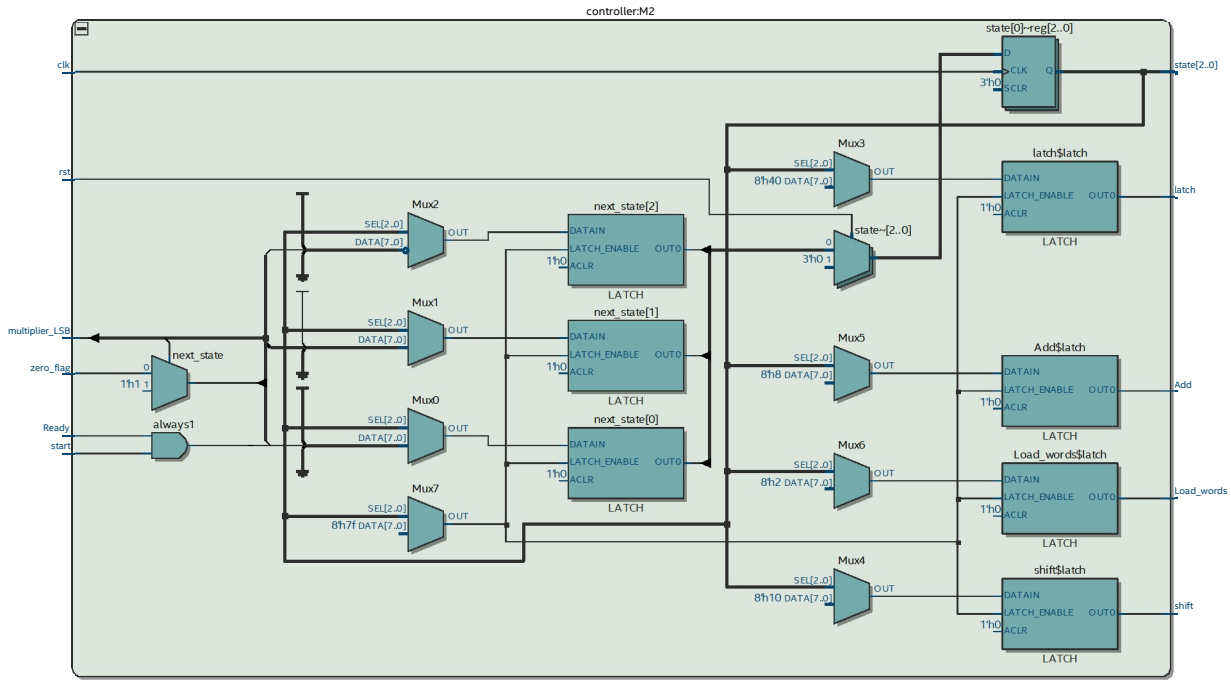
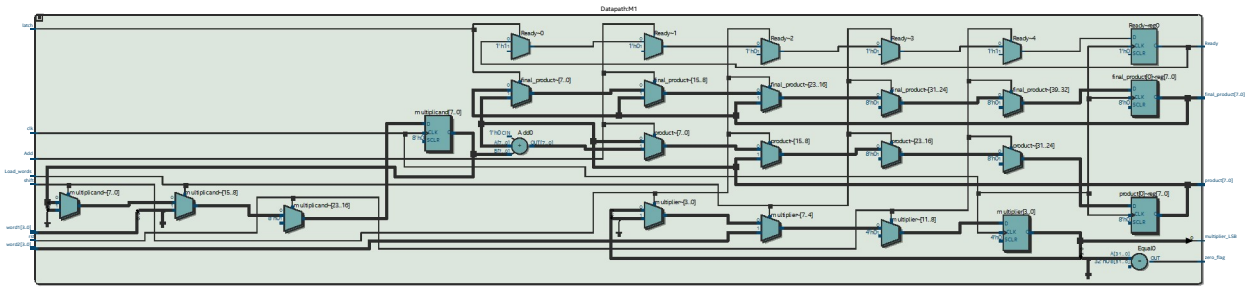
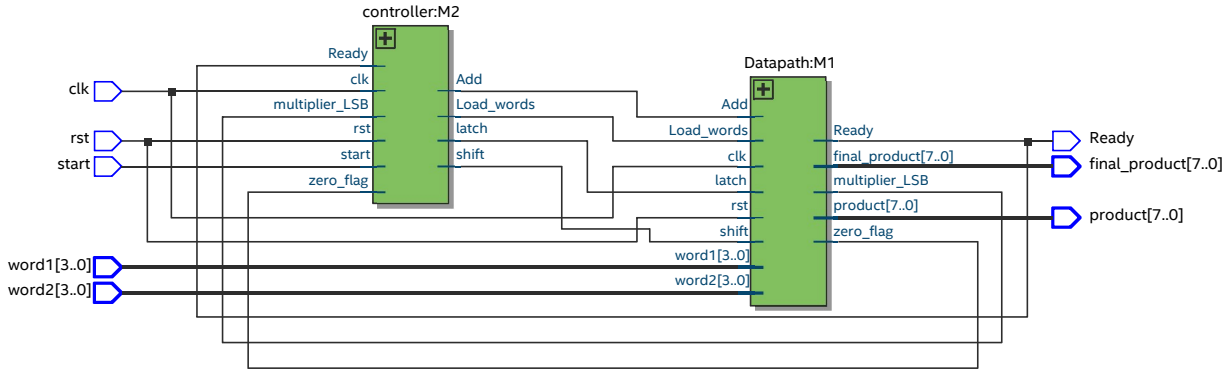
```
1   // ee417 lesson 7 Assignment 1 L7A1
2   // Name: Ron Kalin, Date: 06-25-24  Group: Kalin/Jammeh
3   // Design: Sequential_multiplier
4   // top level module raises a ready flag when ready to load new input words
5   // user should activate a start input to indicate a new multiplication operation starts
6   module Sequential_multiplier (product, final_product,
7                                 Ready, start,
8                                 word1, word2,
9                                 clk, rst);
10  parameter                   L_WORD =4;  //Datapathsize
11  output    [2*L_WORD-1: 0] product, final_product;
12  output                    Ready;
13  input     [L_WORD -1: 0]  word1, word2;
14  input                     start, clk, rst;
15
16  reg [L_WORD-1:0] mcand, mult;
17
18  wire multiplier_LSB, Load_words, shift, Add, latch, zero_flag;
19  wire [L_WORD-2:0] state;
20
21  Datapath M1   (product, final_product,
22                 Ready, multiplier_LSB, zero_flag,
23                 word1, word2,
24                 Load_words, shift, Add, latch,
25                 clk, rst);
26
27  controller M2 (Load_words, shift, Add, latch, state,
28                 Ready, multiplier_LSB, start, zero_flag,
29                 clk, rst);
30
31  endmodule
32
33  //Datapath
34  module Datapath (product, final_product,
35                   Ready, multiplier_LSB, zero_flag,
36                   word1, word2,
37                   Load_words, shift, Add, latch,
38                   clk, rst);
39
40  parameter L_WORD= 4;  //declare parameter values
41
42  //declare outputs and input
43  output reg [2*L_WORD-1:0] product, final_product;
44  output reg                Ready;
45  output                    multiplier_LSB, zero_flag;
46  input      [ L_WORD-1:0]  word1, word2;
47  input                     Load_words, shift, Add, latch;
48  input                     clk, rst;
49  //declare internal wires
50  reg    [2*L_WORD-1: 0]    multiplicand;
51  reg    [  L_WORD-1: 0]    multiplier;
52  //assign values
53  assign multiplier_LSB = multiplier[0];    //least significant bit of multiplier
54  assign zero_flag      = (multiplier == 0);//if multiplier=all zeros, zero_flag=1
55
56  //create always block
57  always @ (posedge clk)
58   begin
59         if (rst)        begin multiplier    <= 0;//if reset =1 then zero
60                               multiplicand  <= 0;
61                               product       <= 0;
62                               final_product <= 0;
63                               Ready         <= 1; end//ready high= accept input words
64     else if (Load_words) begin multiplicand  <= word1;//Load_words=1
65                               multiplier    <= word2;//then m..cand gets value of word1
66                               product       <= 0;    //mult gets value word2
67                               final_product <= 0;    //prod and final_prod=0
68                               Ready         <= 0; end//ready low means calculate
69     else if (shift)      begin multiplier    <= multiplier >> 1;   //shift right 1
```

```
70                                  multiplicand  <= multiplicand << 1; //shift left 1
71                                  Ready         <= 0; end
72      else if (Add)         begin product       <= product + multiplicand;
73                                  Ready         <= 0; end
74      else if (latch)       begin final_product <= product; //adding done
75                                  Ready         <= 1; end   //ready for new input
76      else                  begin end
77     end
78
79   endmodule
80
81   //controller
82   module controller (Load_words, shift, Add, latch, state,
83               Ready, multiplier_LSB, start, zero_flag,
84               clk, rst);
85   //parameter L_WORD= 4;  //declare parameter values
86
87   //declare outputs/inputs, control unit only handles single bit inputs and outputs
88   output reg Load_words, shift, Add, latch;
89   output reg [2:0] state;
90   input      Ready, multiplier_LSB, start, zero_flag;
91   input      clk, rst;
92
93   reg [2:0]  next_state; //3 bits for up to 8 states
94   //build code from FSM diagram
95
96   //declare states from FSM
97   parameter idle    = 3'b000;
98   parameter loading = 3'b001;
99   parameter loaded  = 3'b010;
100  parameter add     = 3'b011;
101  parameter shft    = 3'b100;
102  parameter buff    = 3'b101;
103  parameter ltch    = 3'b110;
104
105  always @ (posedge clk)
106  if (rst) state <= idle;
107     else  state <= next_state;
108
109  always @ *
110    //assign probe <= state;
111    case (state)
112      idle: begin
113              Load_words = 1'b0;
114              latch      = 1'b0;
115              Add        = 1'b0;
116              shift      = 1'b0;
117              if (Ready && start) next_state = loading;
118                            else  next_state = idle;    end
119      loading: begin
120              Load_words = 1'b1;
121              latch      = 1'b0;
122              Add        = 1'b0;
123              shift      = 1'b0;
124              next_state = loaded;   end
125      loaded: begin  //2nd load stage is needed because input data changes
126              Load_words = 1'b0;  //1 cycle is needed to look at the change
127              latch      = 1'b0;
128              Add        = 1'b0;
129              shift      = 1'b0;
130              if (multiplier_LSB) next_state = add;
131                            else  next_state = shft;    end
132      add: begin
133              Load_words = 1'b0;
134              latch      = 1'b0;
135              Add        = 1'b1;    //output changes
136              shift      = 1'b0;
137              next_state = shft;     end  //shift always follows after an add
138      shft: begin
```

```
139               Load_words = 1'b0;
140               latch      = 1'b0;
141               Add        = 1'b0;
142               shift      = 1'b1;
143               next_state = buff;     end
144       buff: begin //buffer state needed after shift because input data changes
145               Load_words = 1'b0; //1 cycle is needed to take a look at the change
146               latch      = 1'b0;
147               Add        = 1'b0;
148               shift      = 1'b0;
149               if (multiplier_LSB) next_state = add;
150               else if (zero_flag) next_state = ltch;
151                                   else   next_state = shft;
    end
152       ltch: begin
153               Load_words = 1'b0;
154               Add        = 1'b0;
155               shift      = 1'b0;
156               latch      = 1'b1;
157               next_state = loading;    end  //cycle back to loading stage
158
159   //below is initial pseudocode
160   //if rst=1 then load_words=0, shift=add=latch=0
161   //if rst=0 and start=1 and ready=1 then load_words=1, shift=0, add=0, latch=0
162   //if rst=0 and start=1 and ready=0 then
163     //if multiplier_LSB=0 then load_words=0, shift=1, add=0, latch=0
164     //
165     //else if multiplier_LSB=1 then load_words=0, shift=0, add=1, latch=0
166     //
167     //else if zero_flag=1      then load_words=0, shift=0, Add=0, latch=1, ready=1 ...done
168     endcase
169   endmodule
170
171
```

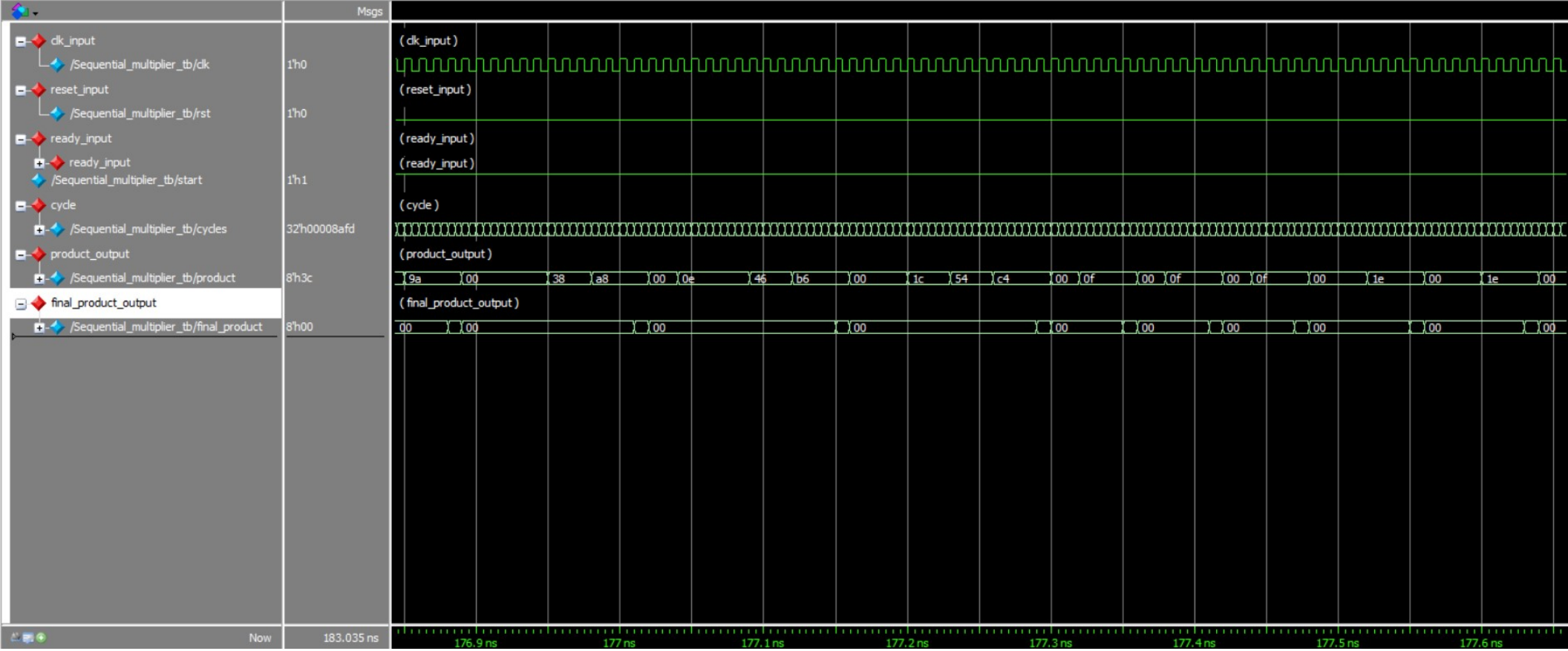

controller:M2
clk
rst
multiplier_LSB
zero_flag
Ready
start
next_state
always1
Mux0
Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
Mux7
next_state[2]
next_state[1]
next_state[0]
state~[2..0]
state[0]~reg[2..0]
latch$latch
Add$latch
Load_words$latch
shift$latch
LATCH
state[2..0]
latch
Add
Load_words
shift


Datapath:M1
multiplicand[7..0]
multiplicand[15..8]
multiplicand[23..16]
multiplier[3..0]
multiplier[7..4]
multiplier[11..8]
multiplier[3..0]
final_product[7..0]
final_product[15..8]
final_product[23..16]
final_product[31..24]
final_product[39..32]
product[7..0]
product[15..8]
product[23..16]
product[31..24]
Ready~0
Ready~1
Ready~2
Ready~3
Ready~4
Add0
Equal0


controller:M2
Ready
clk
multiplier_LSB
rst
start
zero_flag
Add
Load_words
latch
shift
Datapath:M1
Add
Load_words
clk
latch
rst
shift
word1[3..0]
word2[3..0]
Ready
final_product[7..0]
multiplier_LSB
product[7..0]
zero_flag
clk
rst
start
word1[3..0]
word2[3..0]
Ready
final_product[7..0]
product[7..0]

```
1   /*-------------------------------------------------------------------------------
2   Name Ron Kalin
3   CLass: EE417 Summer 2024
4   Lesson 07 HW Question 01
5   Group: Ron Kalin/ Lamin Jammeh
6   Project Description: test-bench for sequential multiplier
7   -------------------------------------------------------------------------------*/
8   module Sequential_multiplier_tb ();
9
10  //set the parameters wires and registers
11  parameter      word_size = 4;  //bit length of word inputs
12  parameter      half_cycle = 5; //half cycle time of clock
13  parameter      full_cycle = 10;//full cycle time of clock
14  parameter      cycle_time = 160;//number of cycles before next cycle
15  /*top level module under test original declaration
16  module Sequential_multiplier (product, final_product,
17                                Ready, start,
18                               word1, word2,
19                               clk, rst);*/
20  //define outputs as wires, inputs as registers
21  wire [2*word_size-1: 0] product, final_product;
22  wire                    Ready;
23  wire [word_size-2:0]    stateProbe2; //internal probe wire for troubleshooting
24  wire [word_size-1:0]    multiplierProbe;
25  wire [2*word_size-1: 0] multiplicandProbe;
26  reg  [word_size -1: 0]  multiplier2;
27  reg  [2*word_size-1: 0] multiplicand1;
28  reg                     start, clk, rst;
29  integer cycles;
30  //define the unit under test UUT
31  Sequential_multiplier UUT (product, final_product,
32                             Ready, start,
33                             multiplicand1, multiplier2,
34                             clk, rst);
35  //internal probes to track logic and troubleshoot
36  //assign stateProbe = UUT.state;
37  assign stateProbe2 = UUT.M2.state; //UUT=top module, M2 =submodule instance name, state is
    register
38  assign multiplierProbe   = UUT.M1.multiplier;
39  assign multiplicandProbe = UUT.M1.multiplicand;
40
41  //instantiate clock
42  initial
43      begin: clock_loop
44        clk = 1'b1;
45        cycles = 0;
46        forever begin
47           #half_cycle    clk = ~clk;
48           cycles = cycles + 1;
49        end
50        if (cycles == cycle_time) disable clock_loop;
51      end
52
53  //define input words and observe the outputs
54  initial begin
55    start = 1;  //start to high means start process
56    rst = 1;    //reset high will set everything to zero and ready to high
57    multiplicand1 = 8'b0000_0101; //initialize both words random test values
58    multiplier2   = 4'b0101;
59    #10 rst = 0;//reset low
60    #cycle_time
61    rst =1;
62    multiplicand1 = 8'b0000_1111; //initialize both words random test values
63    multiplier2   = 4'b1101;
64    #10 rst = 0;
65    #cycle_time //disable clock_loop;
66  //end
67    multiplier2 = 4'b0000;
68    multiplicand1 = 8'b0000_0000;
```

```
69      forever begin: input_loop
70       multiplier2 = multiplier2 + 1'b1;  //increment multiplier
71       if (multiplier2 == 4'b1111) begin      //if multiplier reaches max value
72           multiplier2 = 4'b0001;               //reset multiplier to one
73           multiplicand1 = multiplicand1 + 1'b1;//incr multiplicand
74       end
75       if (multiplicand1 == 8'b1111_1111 && multiplier2 == 4'b1111) begin //both reach max value
76          disable input_loop; disable clock_loop; end
77       #cycle_time;
78      end
79    end
80    ////monitor and display the output
81    initial begin
82    $monitor("multiplicand1 = %b: multiplier2 = %b: stateProbe2=%d: product=%d:
      final_product=%d:",multiplicand1, multiplier2,
83       stateProbe2,product, final_product);
84       end
85    endmodule
```


Msgs
clk_input
/Sequential_multiplier_tb/clk
1'h0
reset_input
/Sequential_multiplier_tb/rst
1'h0
ready_input
ready_input
/Sequential_multiplier_tb/start
1'h1
cycle
/Sequential_multiplier_tb/cycles
32'h00008afd
product_output
/Sequential_multiplier_tb/product
8'h3c
final_product_output
/Sequential_multiplier_tb/final_product
8'h00
Now
183.035 ns
(clk_input)
(reset_input)
(ready_input)
(ready_input)
(cycle)
(product_output)
9a 00 38 a8 00 0e 46 b6 00 1c 54 c4 00 0f 00 0f 00 0f 00 1e 00 1e 00
(final_product_output)
00 00 00 00 00 00 00 00 00 00
176.9 ns
177 ns
177.1 ns
177.2 ns
177.3 ns
177.4 ns
177.5 ns
177.6 ns

```
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=5: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=3: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=4: product= 44: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=5: product= 44: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=6: product= 44: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=1: product= 44: final_product= 44:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=2: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0100: stateProbe2=4: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=5: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=4: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=5: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=3: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=4: product= 44: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=5: product= 44: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=6: product= 44: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=1: product= 44: final_product= 44:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=2: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=3: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=4: product= 11: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=5: product= 11: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=4: product= 11: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=5: product= 11: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=3: product= 11: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0101: stateProbe2=4: product= 55: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=5: product= 55: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=6: product= 55: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=1: product= 55: final_product= 55:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=2: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=4: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=5: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=3: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=4: product= 22: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=5: product= 22: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=3: product= 22: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=4: product= 66: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=5: product= 66: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=6: product= 66: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=1: product= 66: final_product= 66:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=2: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0110: stateProbe2=4: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0111: stateProbe2=5: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0111: stateProbe2=3: product=  0: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0111: stateProbe2=4: product= 22: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0111: stateProbe2=5: product= 22: final_product=  0:
# multiplicand1 = 00111011: multiplier2 = 0111: stateProbe2=3: product= 22: final_product=  0:
```