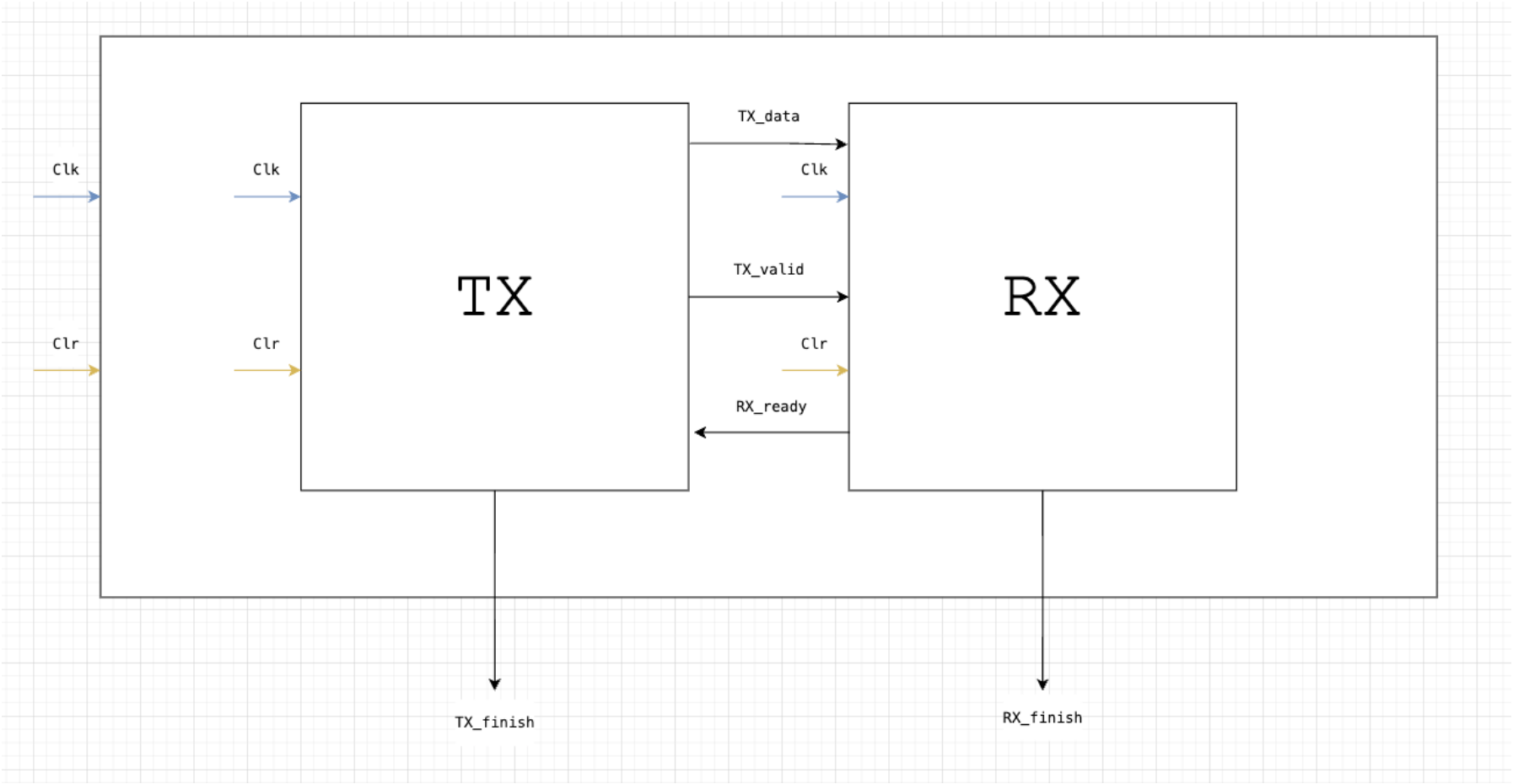
תכן מערכות ספרתיות מתקדמות

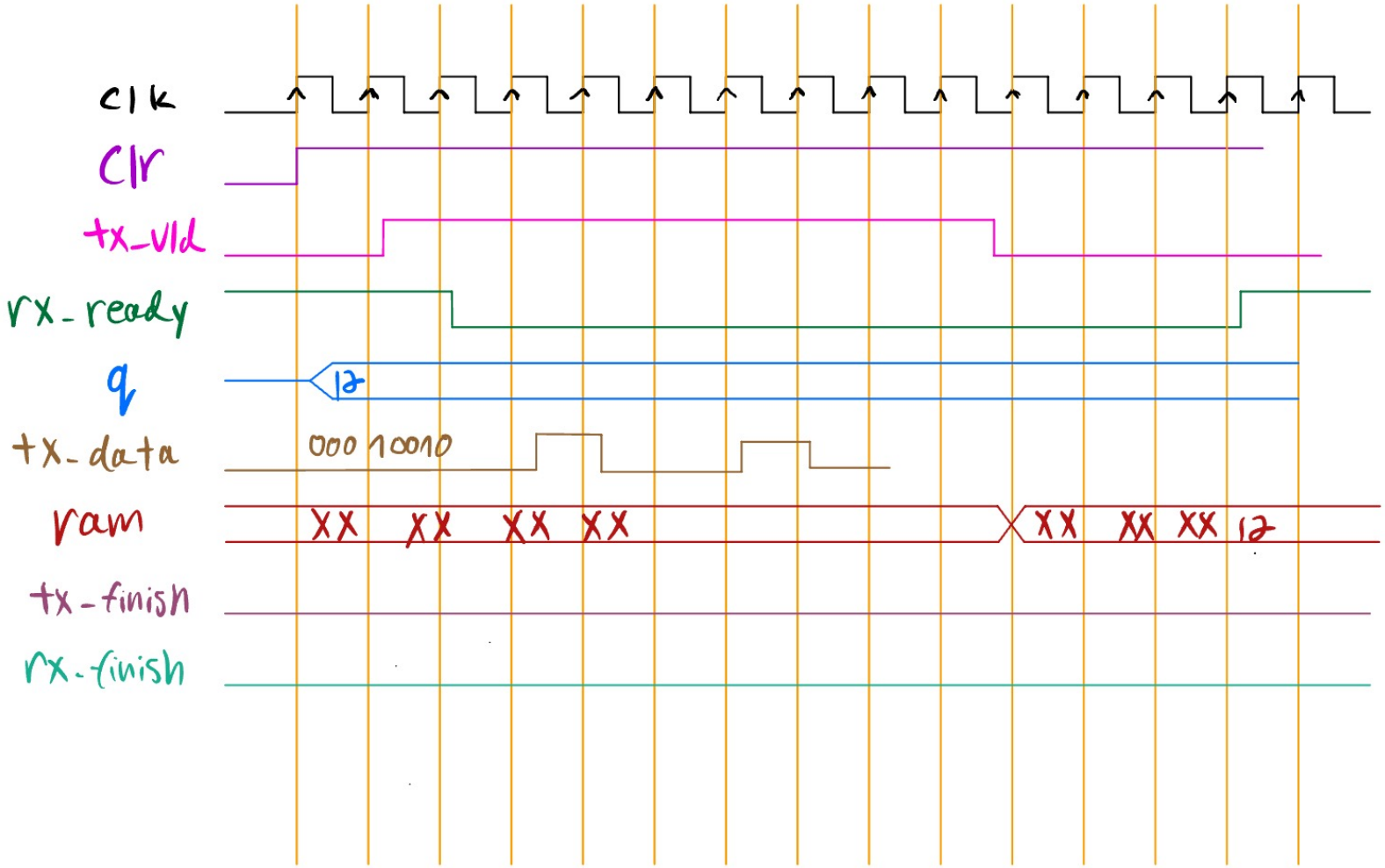
פרוייקטון סוף קורס

אבי שבל ודניאל שקד

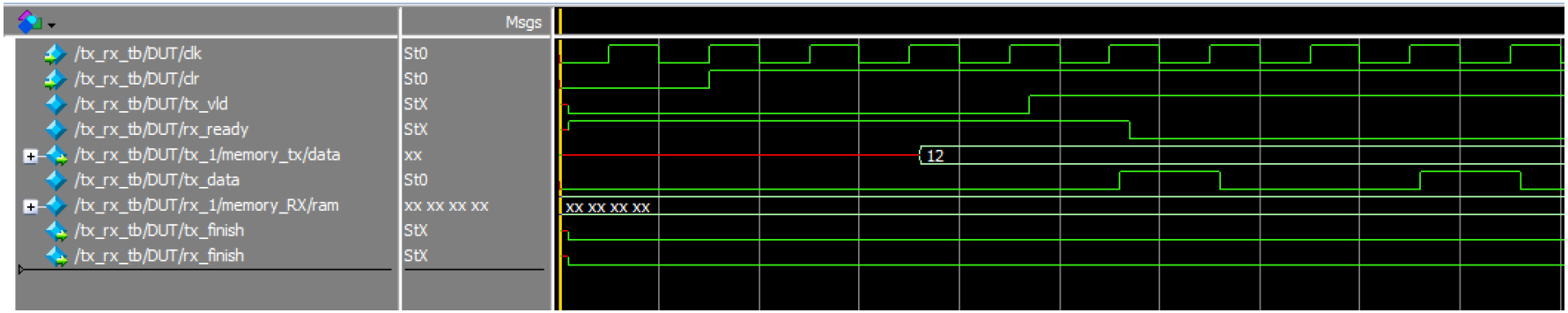
TX_RX – Top Level

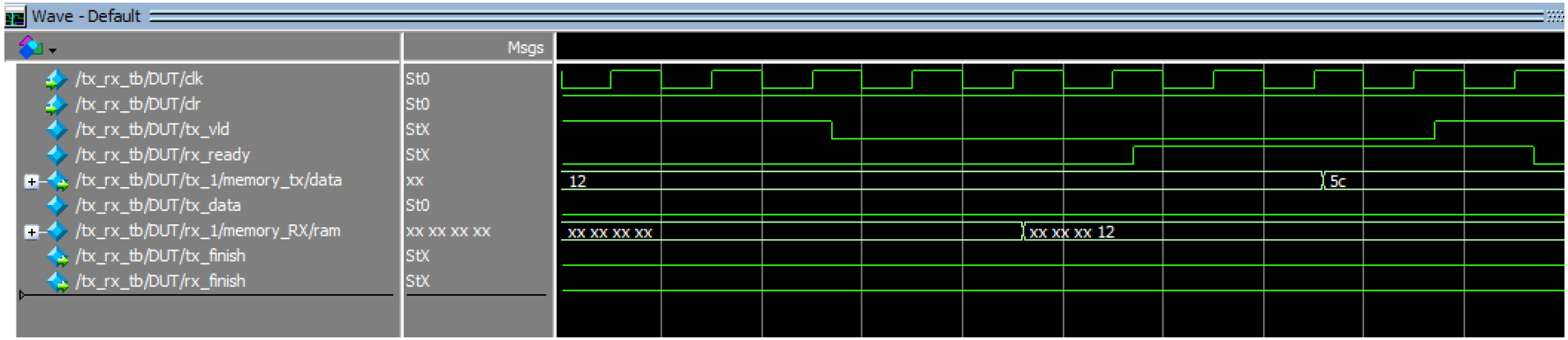


TX_RX – Top Level דיאגרמת גלים



TX_RX – Top Level דיאגרמת גלים

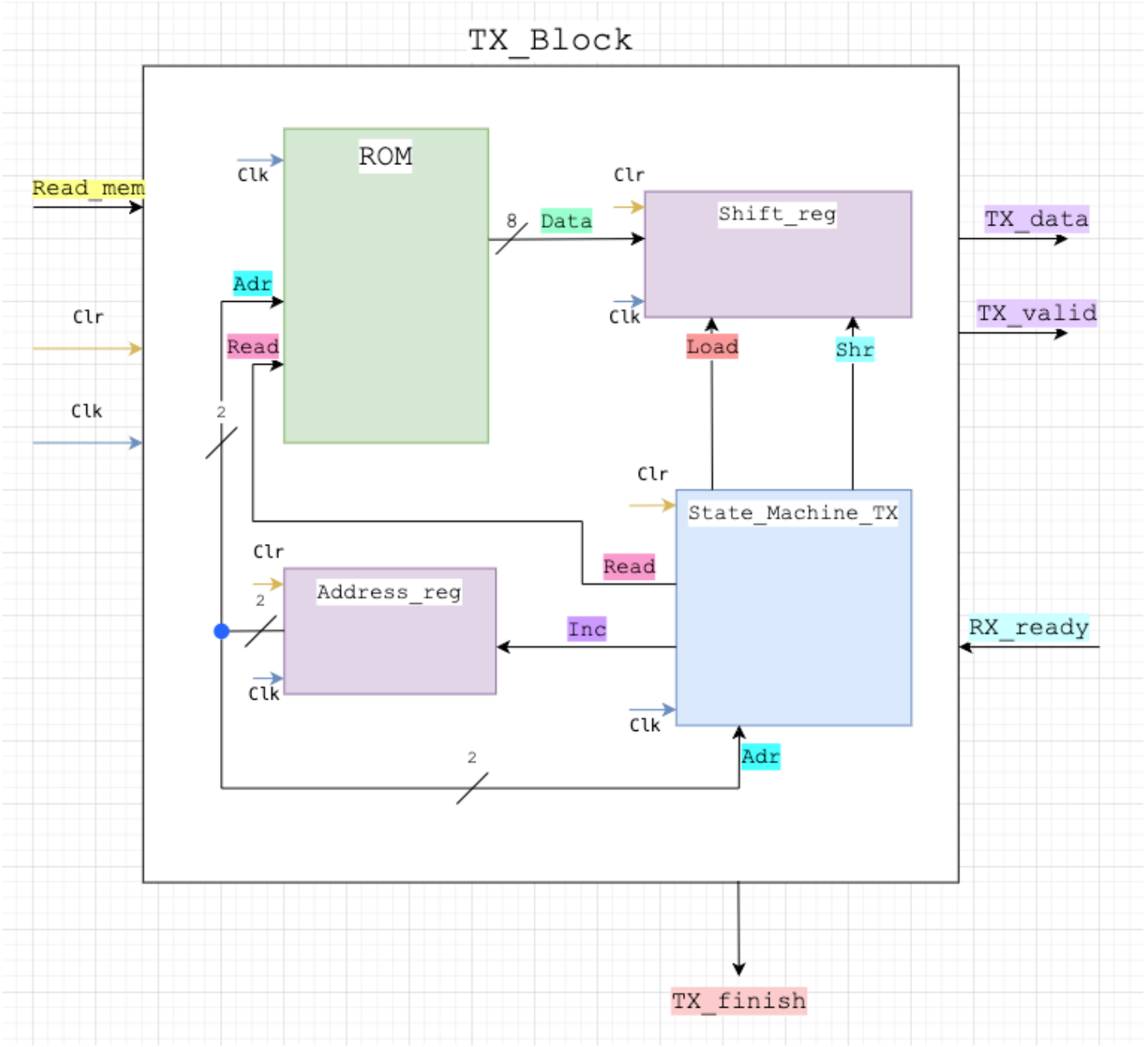




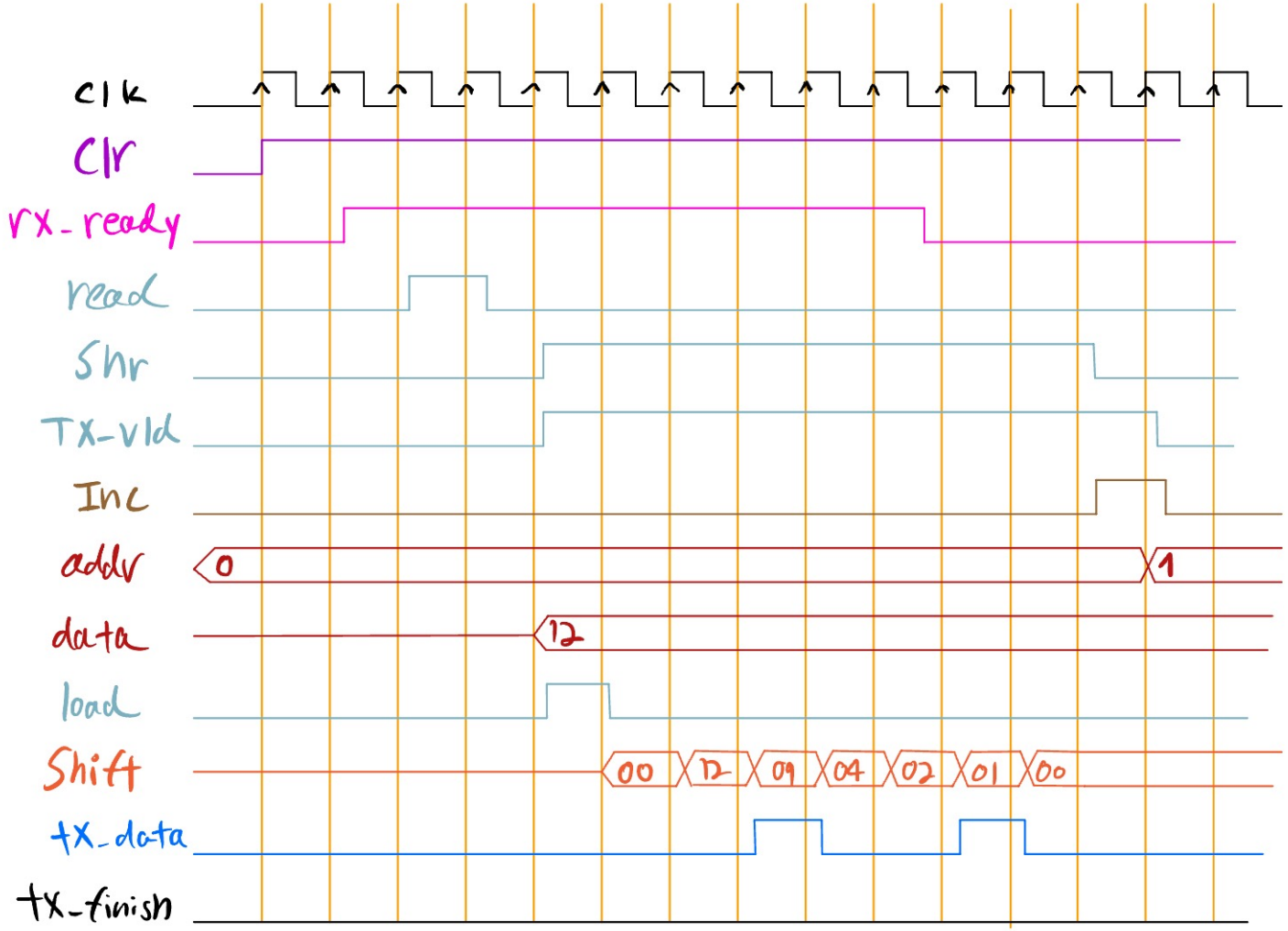
TX_RX – Top Level דיאגרמת גלים

```
1   `timescale 1ns/1ns
2
3   module tx_rx (
4
5       input wire clk,
6       input wire clr,
7       output wire tx_finish,
8       output wire rx_finish
9   );
10
11  wire tx_data;
12  wire tx_vld;
13  wire rx_ready;
14
15  //declaration of the tx module
16  tx tx_1 (.tx_data(tx_data), .tx_vld(tx_vld), .tx_finish(tx_finish), .clk(clk), .clr(clr), .rx_ready(rx_ready));
17
18  //declaration of the rx module
19  rx rx_1 (.rx_ready(rx_ready), .rx_finish(rx_finish), .tx_vld(tx_vld), .clk(clk) , .clr(clr), .tx_data(tx_data));
20
21  endmodule
22
```

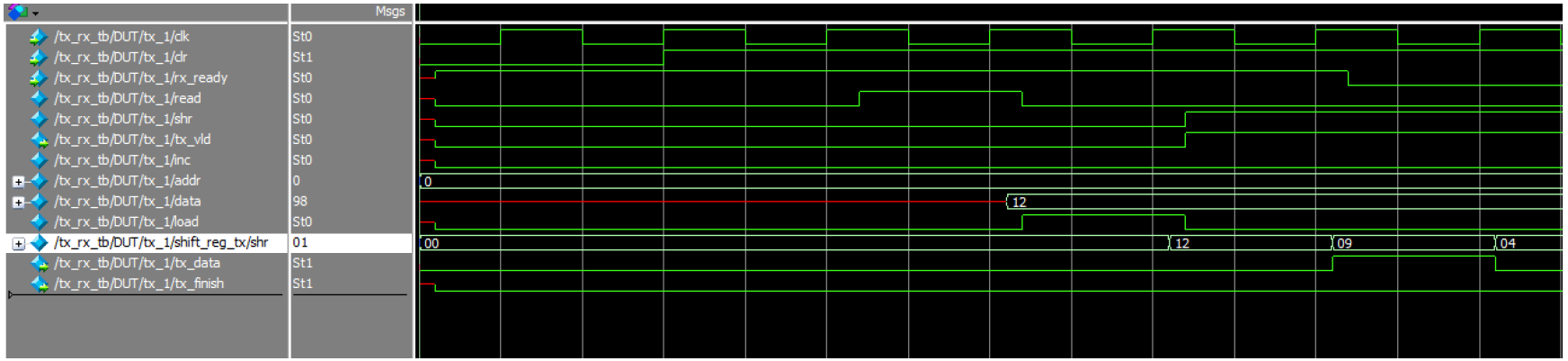
TX

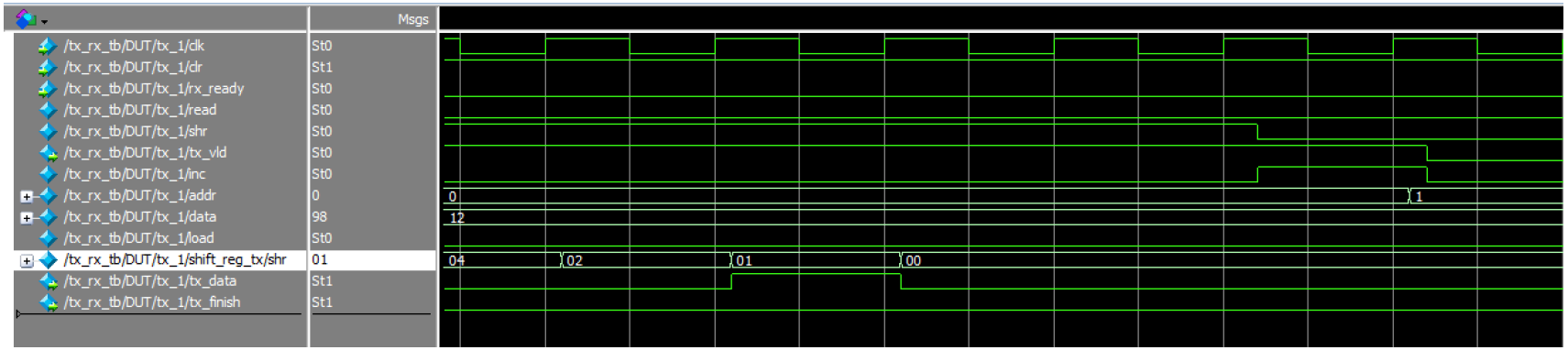

TX_Block
ROM
Clk
Adr
Read
Data
8
Shift_reg
Clr
Clk
Load
Shr
Read_mem
Clr
Clk
2
TX_data
TX_valid
RX_ready
State_Machine_TX
Clr
Clk
Read
Inc
Address_reg
Clr
2
Clk
2
Adr
TX_finish

TX_Block דיאגרמת גלים

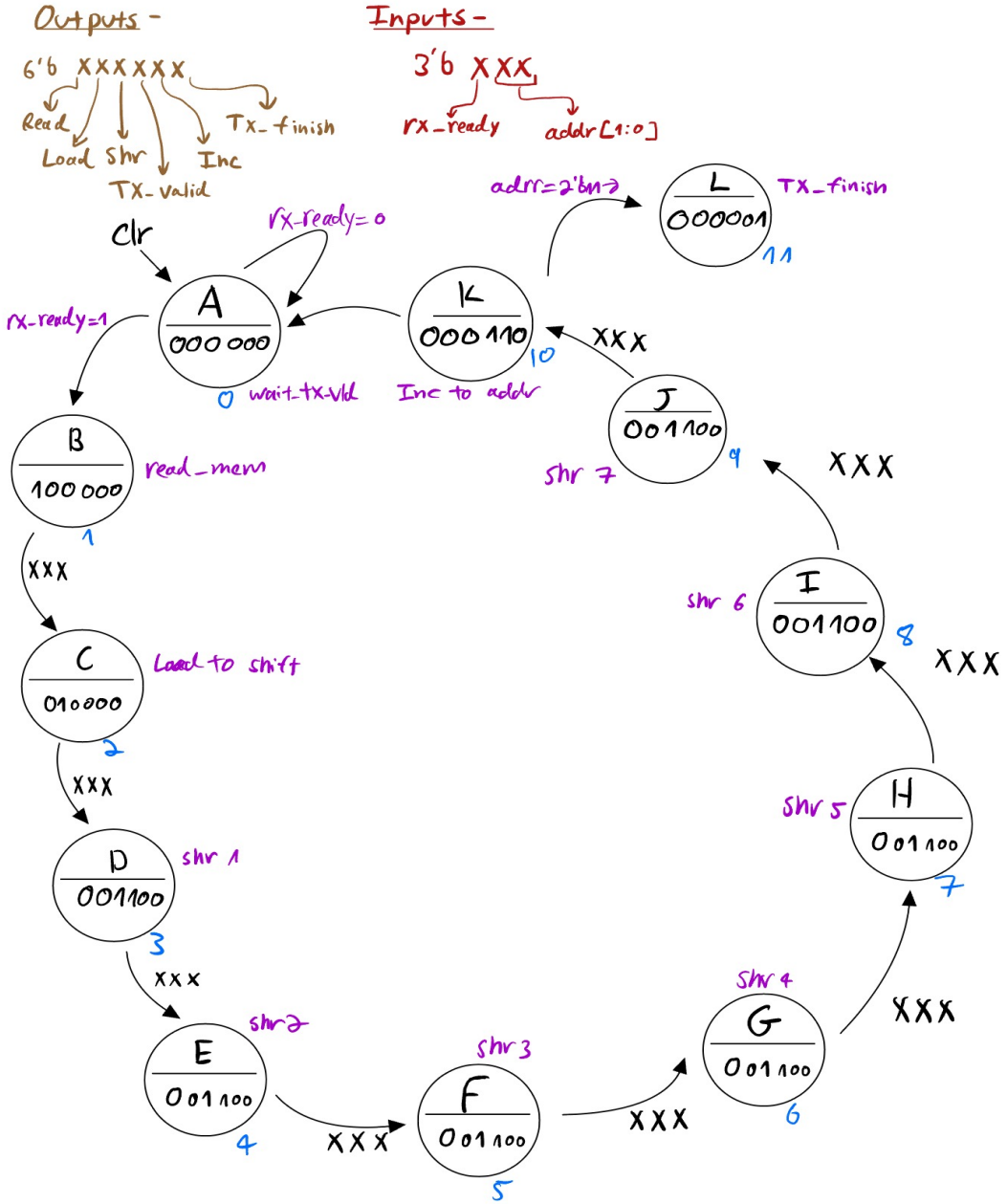


דיאגרמת גלים TX_Block





SM_TX


Outputs -
6'b XXXXXX
Read
Load
Shr
TX_valid
Inc
Tx_finish
Inputs -
3'b XXX
rx_ready
addr[1:0]
Clr
A
000000
0
wait_tx_vld
rx_ready=0
rx_ready=1
B
100000
1
read_mem
C
010000
2
Load to shift
D
001100
3
shr 1
E
001100
4
shr 2
F
001100
5
shr 3
G
001100
6
shr 4
H
001100
7
shr 5
I
001100
8
shr 6
J
001100
9
shr 7
K
000110
10
Inc to addr
L
000001
11
TX_finish
addr=2'b11
XXX

TX

```
1  `timescale 1ns/1ns
2
3  module tx (
4
5      input wire clk,
6      input wire clr,
7      input wire rx_ready,
8      output wire tx_data,
9      output wire tx_vld,
10     output wire tx_finish
11 );
12
13 wire read, inc, load, shr;
14 wire [1:0] addr;
15 wire [7:0] data;
16
17 //state machine module
18 state_machine_tx state_machine_tx( .read(read), .inc(inc), .load(load), .shift(shr), .tx_vld(tx_vld),
19     .tx_finish(tx_finish), .clk(clk), .clr(clr), .rx_ready(rx_ready), .addr(addr));
20
21 //ROM module
22 ROM memory_tx( .addr(addr), .read(read), .clk(clk), .data(data));
23
24 //shift_reg module
25 shift_reg shift_reg_tx( .clk(clk), .clr(clr), .load(load), .shift(shr), .data(data), .tx_data(tx_data));
26
27 //addr_reg module
28 addr_reg addr_reg_tx( .clk(clk), .clr(clr), .inc(inc), .addr(addr));
29
30 endmodule
```

ROM

```
1  `timescale 1ns/1ns
2
3  module ROM
4  #(parameter DATA_WITDH = 8, parameter ADDR_WIDTH = 2)
5  (
6      input [(ADDR_WIDTH-1):0] addr,
7      input clk,
8      input read,
9      output reg [(DATA_WITDH-1):0] data
10 );
11
12     //declaring the ROM variable
13     reg [DATA_WITDH-1:0] rom [2**ADDR_WIDTH-1:0];
14
15     //initializing the rom with $readmemb.
16     //it will read the memory. without it this design will not compile
17
18     initial
19     begin
20         $readmemh("memory.list", rom); // reaing the memory list file
21     end
22
23     always @ (posedge clk)
24     begin
25     //the start of the data transfer - when read is on the data can get out
26     if (read)
27         data <= #1 rom[addr];
28     end
29
30 endmodule
```

Shift_Reg_TX

```
1  `timescale 1ns/1ns
2
3  module shift_reg (
4
5      input wire clk,
6      input wire clr,
7      input wire shift,
8      input wire load,
9      input wire [7:0] data,
10     output wire tx_data
11 );
12
13
14 reg [7:0] shr; //register to save the bits inside
15
16 assign tx_data = shr[0];
17
18 always @(posedge clk or negedge clr) //begins whenever there is a clock
19 begin
20     if (~clr)
21         shr <= 0; //rest
22     else if (load)
23         shr <= #1 data; //load the data
24     else if(shift)
25         shr <= #1  shr >> 1; //move one bit right, and add zero to msb
26 end
27
28 endmodule
```

Address_Reg

```
1   `timescale 1ns/1ns
2
3   module addr_reg (
4
5       input clk,
6       input clr,
7       input inc,
8       output reg [1:0] addr
9   );
10
11
12  always @(posedge clk or negedge clr)
13
14      begin //begins whenever there is a clock
15      if (~clr) //rest
16          addr <= 0;
17      else if (inc)
18          addr <= #1 addr + 1;  //moves on to the next memory cell
19      end
20
21  endmodule
```

SM_TX

```
1  `timescale 1ns/1ns
2
3  module state_machine_tx (
4
5      input wire clk,
6      input wire clr,
7      input wire rx_ready, // RX is ready to recieve data
8      input reg [1:0] addr, // 2bit for 4 addresses
9      output wire read, // allows the memory to output the data
10     output wire inc, // moves on to the next memory cell
11     output wire load, // loads the data on the next shift reg
12     output wire shift, // shitf's the reg one bit to the right
13     output wire tx_vld, // makes sure the data from the tx is valid
14     output wire tx_finish // makes sure the transmition of the data from all the memory cells is complete
15 );
16
17 reg [3:0] state;
18 reg [5:0] out;
19
20 parameter   A    = 4'h0;  //wait until rx is ready
21 parameter   B    = 4'h1;  //read memory
22 parameter   C    = 4'h2;  //load to shift
23 parameter   D    = 4'h3;  //shift 1
24 parameter   E    = 4'h4;  //shift 2
25 parameter   F    = 4'h5;  //shift 3
26 parameter   G    = 4'h6;  //shift 4
27 parameter   H    = 4'h7;  //shift 5
28 parameter   I    = 4'h8;  //shift 6
29 parameter   J    = 4'h9;  //shift 7
30 parameter   K    = 4'hA;  //inc to address
31 parameter   L    = 4'hB;  //finish tx
32
33
34 assign {read, load, shift, tx_vld, inc, tx_finish} = out;
35
```

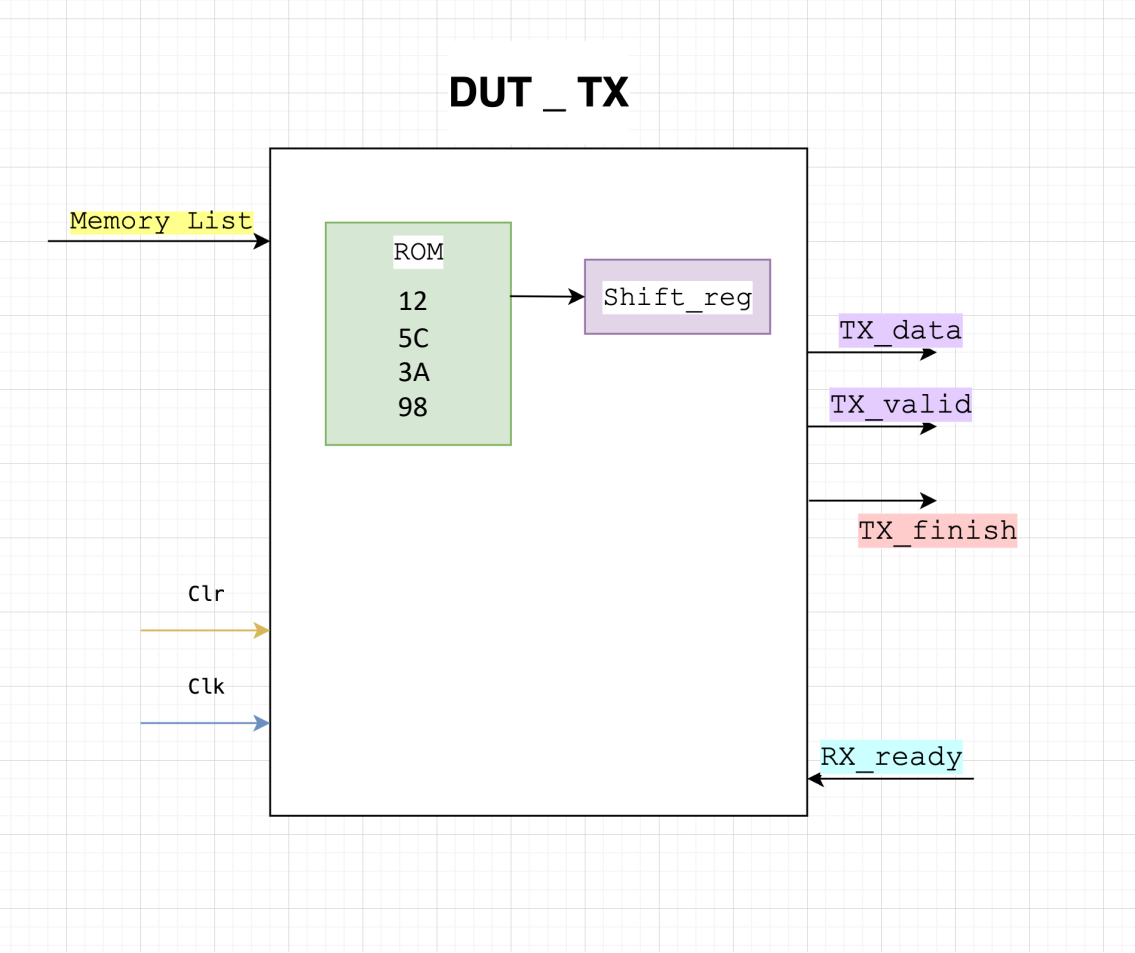
SM_TX – Transition Block

```
35
36  //state transition block
37  always @(posedge clk or negedge clr)
38      begin
39          if (~clr)
40              state <= A;
41          else
42              case (state)
43                  A:      state <= #1 (rx_ready == 1) ? B : A; //waiting for rx to be ready
44                  B:      state <= #1 C;
45                  C:      state <= #1 D;
46                  D:      state <= #1 E;
47                  E:      state <= #1 F;
48                  F:      state <= #1 G;
49                  G:      state <= #1 H;
50                  H:      state <= #1 I;
51                  I:      state <= #1 J;
52                  J:      state <= #1 K;
53                  K:      state <= #1 (addr == 2'b11) ? L : A;  //did we reach the last memort cell?
54              endcase
55      end
56
```

SM_TX – Output Logic

```
56
57
58  //output logic
59  always @(*)
60      begin
61          case (state)
62              A:  out <= #1 6'b000000;
63              B:  out <= #1 6'b100000;
64              C:  out <= #1 6'b010000;
65              D:  out <= #1 6'b001100;
66              E:  out <= #1 6'b001100;
67              F:  out <= #1 6'b001100;
68              G:  out <= #1 6'b001100;
69              H:  out <= #1 6'b001100;
70              I:  out <= #1 6'b001100;
71              J:  out <= #1 6'b001100;
72              K:  out <= #1 6'b000110;
73              L:  out <= #1 6'b000001;
74          endcase
75      end
76
77  endmodule
78
```

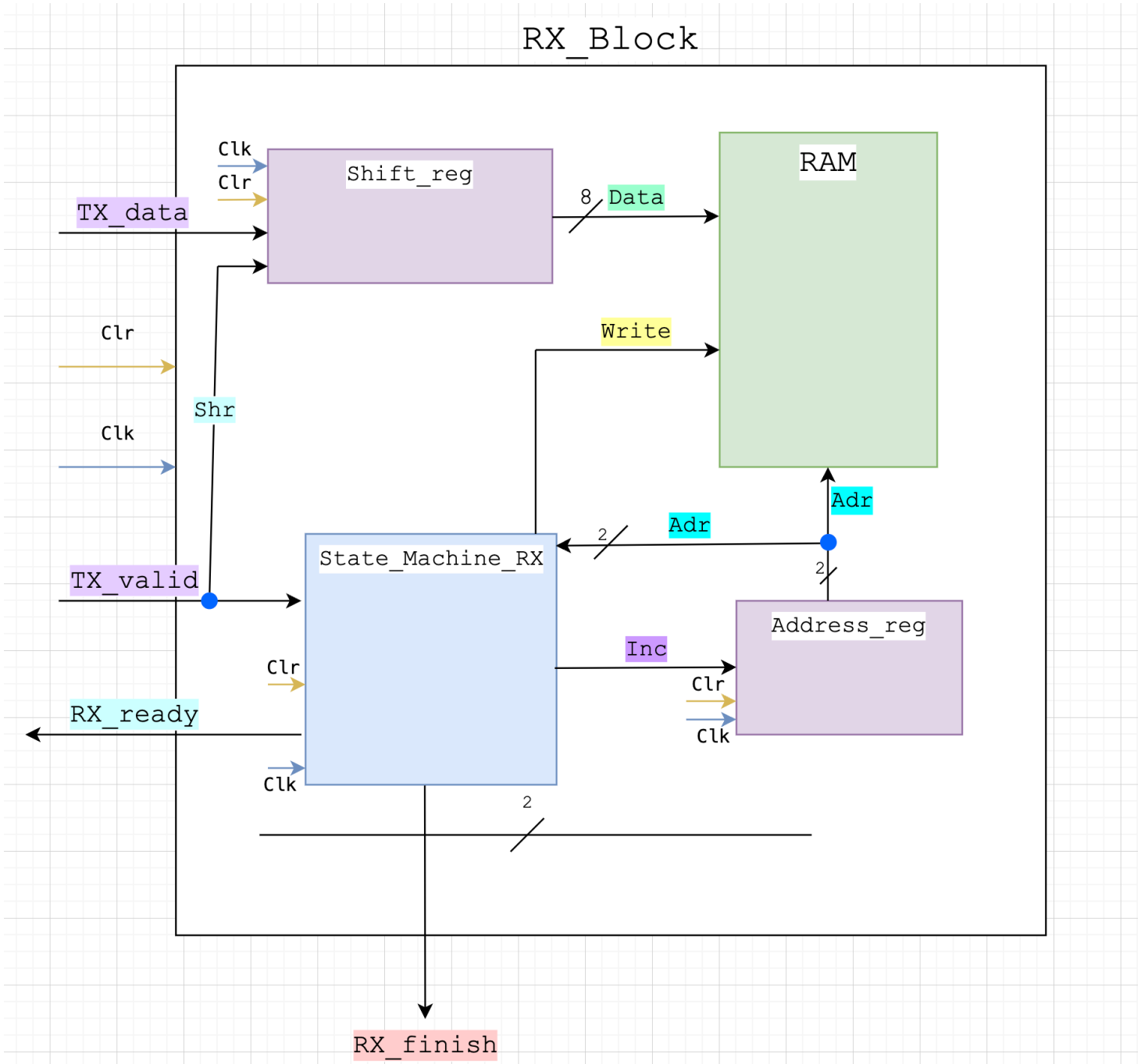
TX_TB


DUT _ TX
Memory List
ROM
12
5C
3A
98
Shift_reg
TX_data
TX_valid
TX_finish
Clr
Clk
RX_ready

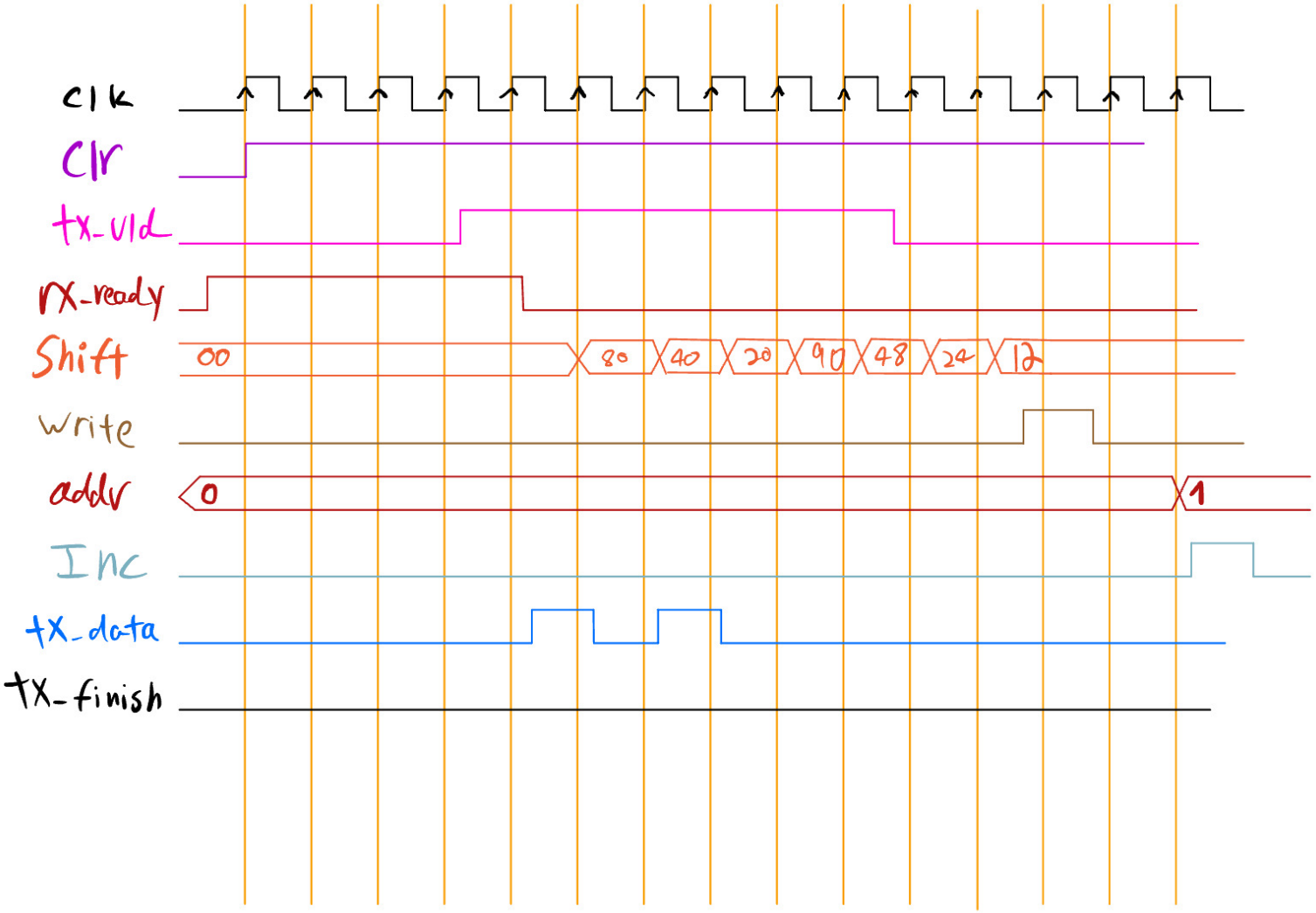
TX_TB

```
1  `timescale 1ns/1ns
2
3  module tb_tx ();
4
5  wire tx_data_tb;
6  wire tx_vld_tb;
7  wire tx_finish_tb;
8  reg clk_tb;
9  reg clr_tb;
10 reg rx_ready_tb;
11
12 //Declaring DUT (device under test)
13 tx DUT ( .clk(clk_tb), .clr(clr_tb), .rx_ready(rx_ready_tb), .tx_finish(tx_finish_tb),
14     .tx_vld(tx_vld_tb), .tx_data(tx_data_tb));
15
16
17 initial begin
18     clk_tb = 0;
19     clr_tb = 0;
20 end
21
22
23 always #5 clk_tb =~ clk_tb;
24
25
26 initial begin
27     clk_tb = 0;
28     clr_tb = 0;
29     rx_ready_tb = 0;
30
31     repeat (2) @(posedge clk_tb); // wiat 2 clks
32     clr_tb = 1;
33     repeat (2) @(posedge clk_tb); //wait 2 clks
34     rx_ready_tb = 1;
35     end
36
37 endmodule
```

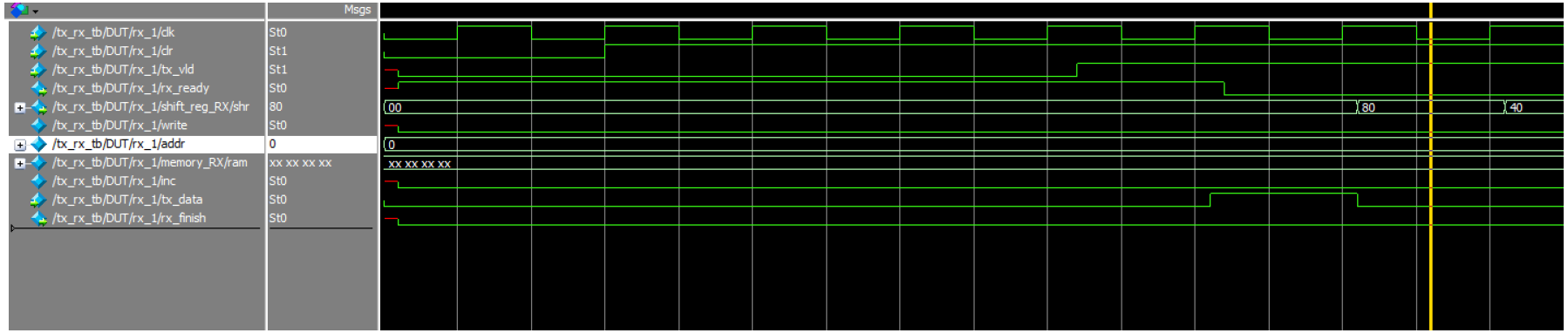
RX

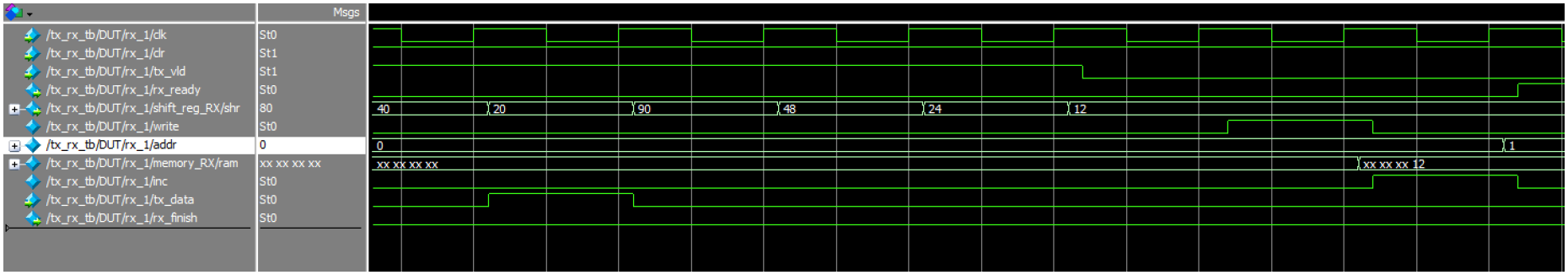

RX_Block
Clk
Clr
Shift_reg
TX_data
8
Data
RAM
Clr
Write
Shr
Clk
Adr
Adr
2
State_Machine_RX
2
TX_valid
Address_reg
Inc
Clr
Clr
RX_ready
Clk
Clk
2
RX_finish

RX_Block דיאגרמת גלים

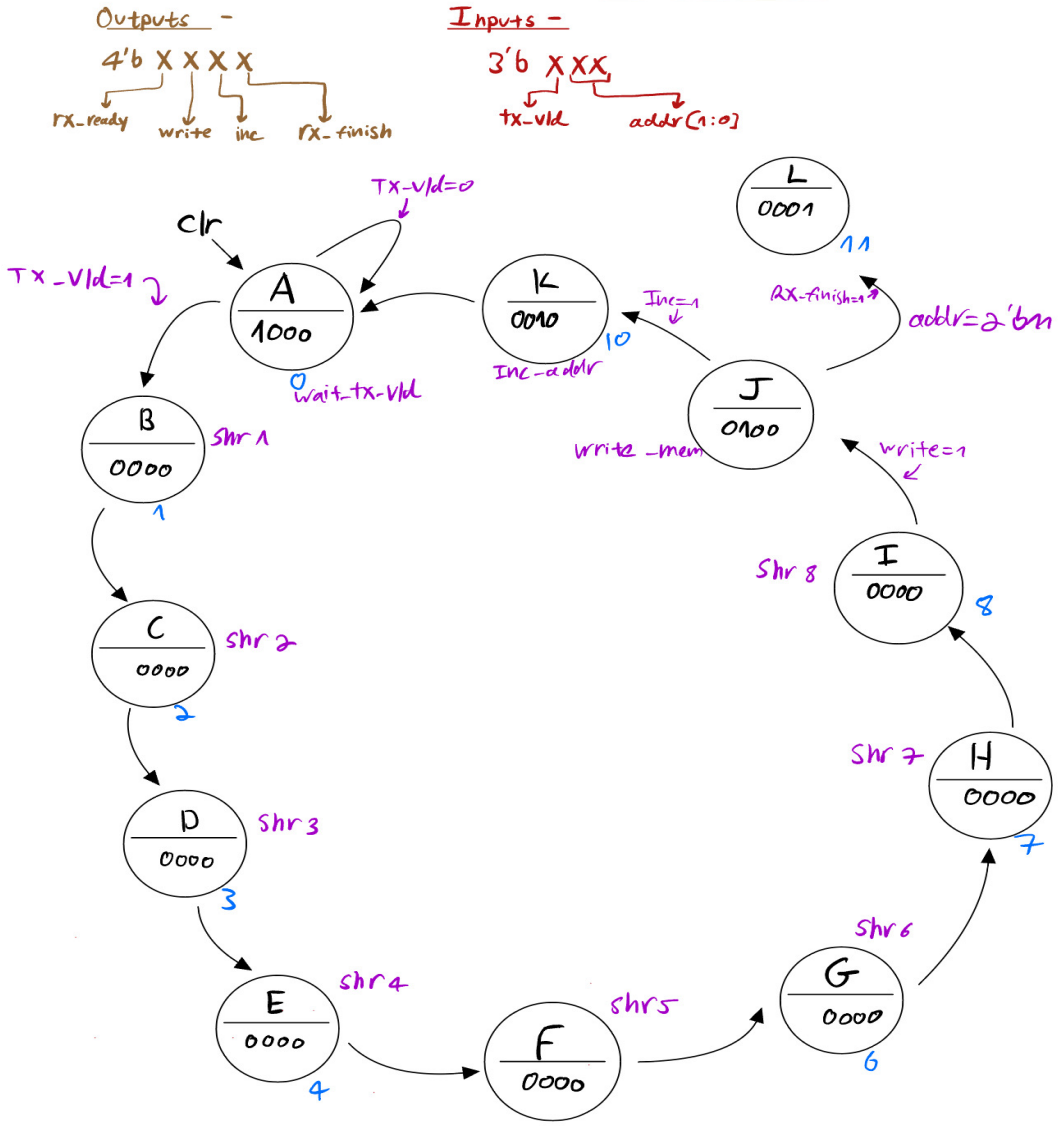


RX_Block דיאגרמת גלים





SM_RX


Outputs -
4'b X X X X
rx_ready
write
inc
rx_finish
Inputs -
3'b X X X
tx_vld
addr[1:0]
clr
A
1000
0
wait_tx_vld
Tx_vld=0
Tx_vld=1
B
0000
1
shr 1
C
0000
2
shr 2
D
0000
3
shr 3
E
0000
4
shr 4
F
0000
shr 5
G
0000
6
shr 6
H
0000
7
shr 7
I
0000
8
shr 8
write=1
J
0100
write_mem
Inc=1
K
0010
10
Inc_addr
RX-finish=1
addr=2'b11
L
0001
11

RX

```
`timescale 1ns/1ns

module rx (

    input wire clk,
    input wire clr,
    input wire tx_vld,
    input wire tx_data,
    output wire rx_ready,
    output wire rx_finish
);

wire write;
wire inc;
wire [1:0] addr;
wire [7:0] data;
wire [7:0] q;

//state machine module
state_machine_rx state_machine_RX(.clk(clk), .clr(clr), .tx_vld(tx_vld),
    .addr(addr), .rx_ready(rx_ready), .write(write), .inc(inc), .rx_finish(rx_finish));

//RAM module
RAM memory_RX(.addr(addr), .data(data), .write(write), .clk(clk), .q(q));

//shitf reg module
shift_reg_rx shift_reg_RX(.clk(clk), .clr(clr), .shift(tx_vld), .tx_data(tx_data), .shr(data));

//address reg module
addr_reg addr_reg_RX( .clk(clk), .clr(clr), .inc(inc), .addr(addr));

endmodule
```

RAM

```
1  `timescale 1ns/1ns
2
3  module RAM
4  #(parameter DATA_WIDTH = 8, parameter ADDR_WIDTH = 2)
5  (
6      input [(DATA_WIDTH-1):0] data,
7      input [(ADDR_WIDTH-1):0] addr,
8      input clk,
9      input write,
10     output [(DATA_WIDTH-1):0] q
11 );
12
13
14     //declaring the RAM variables
15     reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
16
17     //defeining variables to hold the registered addresses
18     reg [ADDR_WIDTH-1:0] addr_reg;
19
20     always @(posedge clk)
21     begin
22     //writing to the RAM
23         if(write)    //write loop
24             ram [addr] <= #1 data;
25
26         addr_reg <= #1 addr;
27     end
28
29     assign q = ram [addr_reg]; //assining the new address into the RAM
30
31 endmodule
```

Shift_Reg_RX

```
1  `timescale 1ns/1ns
2
3  module shift_reg_rx (
4
5      input wire clk,
6      input wire clr,
7      input wire shift,
8      input wire tx_data,
9      output reg [7:0] shr
10
11 );
12
13 always @(posedge clk or negedge clr)  //start when the clk is moving
14     begin
15         if(~clr) //rest
16             shr <= 0;
17         else if (shift)
18             shr <= #1 {tx_data, shr [7:1]}; //move one bit right, and save every bit that is inserted
19     end
20
21 endmodule
22
```

Address_Reg

```
1   `timescale 1ns/1ns
2
3   module addr_reg (
4
5       input clk,
6       input clr,
7       input inc,
8       output reg [1:0] addr
9   );
10
11
12  always @(posedge clk or negedge clr)
13
14      begin //begins whenever there is a clock
15      if (~clr) //rest
16          addr <= 0;
17      else if (inc)
18          addr <= #1 addr + 1;  //moves on to the next memory cell
19      end
20
21  endmodule
```

SM_RX

```
1  `timescale 1ns/1ns
2
3  module state_machine_rx (
4      input wire clk,
5      input wire clr,
6      input wire tx_vld,
7      input wire [1:0] addr,
8      output wire rx_ready,
9      output wire write,
10     output wire inc,
11     output wire rx_finish
12 );
13
14 reg [3:0] state;
15 reg [3:0] out;
16
17 // State declarations
18 parameter A = 4'h0;  // wait until tx is valid
19 parameter B = 4'h1;  // shift 1
20 parameter C = 4'h2;  // shift 2
21 parameter D = 4'h3;  // shift 3
22 parameter E = 4'h4;  // shift 4
23 parameter F = 4'h5;  // shift 5
24 parameter G = 4'h6;  // shift 6
25 parameter H = 4'h7;  // shift 7
26 parameter I = 4'h8;  // shift 8
27 parameter J = 4'h9;  // write the memory
28 parameter K = 4'hA;  // inc to address
29 parameter L = 4'hB;  // finish rx
30
31 assign {rx_ready, write, inc, rx_finish} = out;
32
```

SM_RX – Transition Block

```
33  // State transition block
34  always @(posedge clk or negedge clr) begin
35      if (~clr)
36          state <= A;
37      else
38          case (state)
39              A: state <= #1 (tx_vld == 1) ? B : A;  // waiting for valid data to flow from the tx
40              B: state <= #1 C;
41              C: state <= #1 D;
42              D: state <= #1 E;
43              E: state <= #1 F;
44              F: state <= #1 G;
45              G: state <= #1 H;
46              H: state <= #1 I;
47              I: state <= #1 J;
48              J: state <= #1 (addr == 2'b11) ? L : K; //did the data reach the last memory cell?
49              K: state <= #1 A;
50          endcase
51  end
```

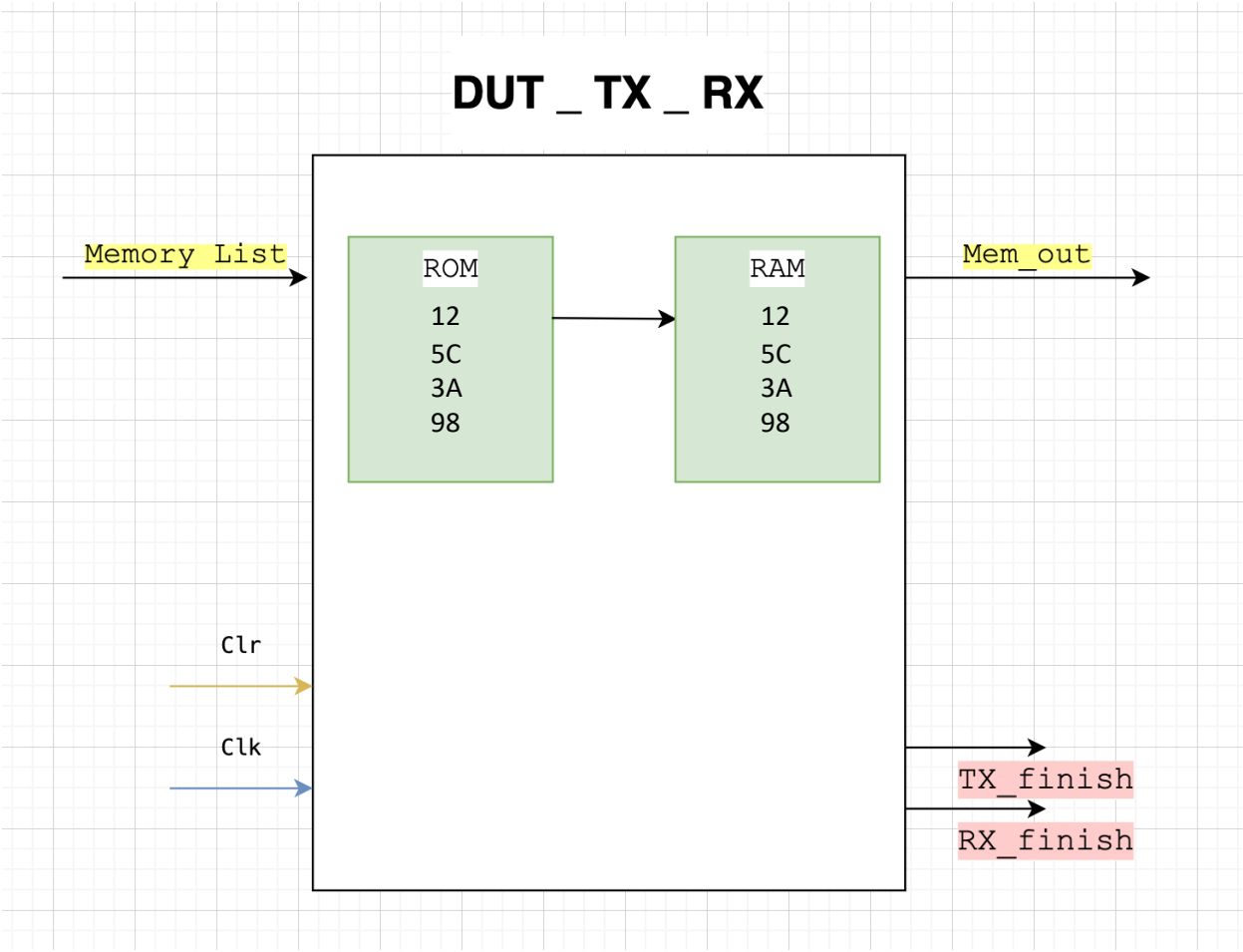
SM_RX – Output Logic

```
52
53  // Output logic - rx_ready, write, inc, rx_finish
54  always @(*) begin
55      case (state)
56          A: out <= #1 4'b1000;
57          B: out <= #1 4'b0000;
58          C: out <= #1 4'b0000;
59          D: out <= #1 4'b0000;
60          E: out <= #1 4'b0000;
61          F: out <= #1 4'b0000;
62          G: out <= #1 4'b0000;
63          H: out <= #1 4'b0000;
64          I: out <= #1 4'b0000;
65          J: out <= #1 4'b0100;
66          K: out <= #1 4'b0010;
67          L: out <= #1 4'b0001;
68      endcase
69  end
70
71  endmodule
```

TB

סביבת בדיקה של שני הבלוקים יחד

TX_RX_TB


DUT _ TX _ RX
Memory List
ROM
12
5C
3A
98
RAM
12
5C
3A
98
Mem_out
Clr
Clk
TX_finish
RX_finish

TX_RX_TB

```
1  `timescale 1ns/1ns
2
3  module tx_rx_tb ();
4
5  wire tx_finish_tb;
6  wire rx_finish_tb;
7  reg clk_tb;
8  reg clr_tb;
9
10 integer file; //defining the file descriptor for mem_out
11 integer i;
12
13 //declaration of the device under test
14 tx_rx DUT (.rx_finish(rx_finish_tb), .tx_finish(tx_finish_tb), .clk(clk_tb), .clr(clr_tb));
15
16 initial begin
17     clk_tb = 0;
18     clr_tb = 0;
19 end
20
21 //clk block
22 always #5 clk_tb =~ clk_tb;
23
24 initial begin
25 clk_tb = 0;
26 clr_tb = 0;
27
28 //resting module
29 repeat (2) @(posedge clk_tb); //wait 2 clks
30 clr_tb = 1;
31 end
```

TX_RX_TB

```
32
33  always @(posedge clk_tb)
34  begin
35          //display when rx is ready
36      if (DUT.rx_1.rx_ready)
37          $display ("time:    %d      rx_ready:   %h", $time, DUT.rx_1.rx_ready);
38
39          //display when the data from the tx is valid
40      if (DUT.tx_1.tx_vld)
41          $display ("time:    %d      tx_vld: %h      tx_data:    %h", $time, DUT.tx_1.tx_vld, DUT.tx_1.tx_data);
42
43          //display memory cell
44      if (DUT.rx_1.rx_finish)
45          begin
46              $display("time: %d      RAM_address:    %h", $time, DUT.rx_1.addr);
47          end
48
49          //after we recived all the data, display the data as a list it the file 'mem_out.list'
50      if (DUT.rx_1.rx_finish)
51          begin
52              $display("time: %d          RAM_address:    %h", $time, DUT.rx_1.addr);
53              $display("time: %d          rx_finish:  %h", $time, DUT.rx_1.rx_finish);
54              file = $fopen("mem_out.list", "w");
55              for (i = 0; i < 4; i = i + 1)
56                  begin
57                          //display each memroy word to the file
58                      $fdisplay(file, "%h", DUT.rx_1.memory_RX.ram[i]);
59                  end
60              #20;
61              $fclose(file);
62          end
63
64  end
65
66  endmodule
```

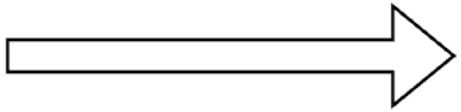
Monitor Output Log

```
# time:                5  rx_ready: 1
# time:               15  rx_ready: 1
# time:               25  rx_ready: 1
# time:               35  rx_ready: 1
# time:               45  rx_ready: 1
# time:               55  rx_ready: 1
# time:               55  tx_vld: 1        tx_data: 0
# time:               65  tx_vld: 1        tx_data: 1
# time:               75  tx_vld: 1        tx_data: 0
# time:               85  tx_vld: 1        tx_data: 0
# time:               95  tx_vld: 1        tx_data: 1
# time:              105  tx_vld: 1        tx_data: 0
# time:              115  tx_vld: 1        tx_data: 0
# time:              125  tx_vld: 1        tx_data: 0
# time:              165  rx_ready: 1
# time:              175  rx_ready: 1
# time:              185  rx_ready: 1
# time:              195  rx_ready: 1
# time:              195  tx_vld: 1        tx_data: 0
# time:              205  tx_vld: 1        tx_data: 0
# time:              215  tx_vld: 1        tx_data: 1
# time:              225  tx_vld: 1        tx_data: 1
# time:              235  tx_vld: 1        tx_data: 1
# time:              245  tx_vld: 1        tx_data: 0
# time:              255  tx_vld: 1        tx_data: 1
# time:              265  tx_vld: 1        tx_data: 0
```

```
# time:              305  rx_ready: 1
# time:              315  rx_ready: 1
# time:              325  rx_ready: 1
# time:              335  rx_ready: 1
# time:              335  tx_vld: 1        tx_data: 0
# time:              345  tx_vld: 1        tx_data: 1
# time:              355  tx_vld: 1        tx_data: 0
# time:              365  tx_vld: 1        tx_data: 1
# time:              375  tx_vld: 1        tx_data: 1
# time:              385  tx_vld: 1        tx_data: 1
# time:              395  tx_vld: 1        tx_data: 0
# time:              405  tx_vld: 1        tx_data: 0
# time:              445  rx_ready: 1
# time:              455  rx_ready: 1
# time:              465  rx_ready: 1
# time:              475  rx_ready: 1
# time:              475  tx_vld: 1        tx_data: 0
# time:              485  tx_vld: 1        tx_data: 0
# time:              495  tx_vld: 1        tx_data: 0
# time:              505  tx_vld: 1        tx_data: 1
# time:              515  tx_vld: 1        tx_data: 1
# time:              525  tx_vld: 1        tx_data: 0
# time:              535  tx_vld: 1        tx_data: 0
# time:              545  tx_vld: 1        tx_data: 1
```

Memory.list – TX_ROM

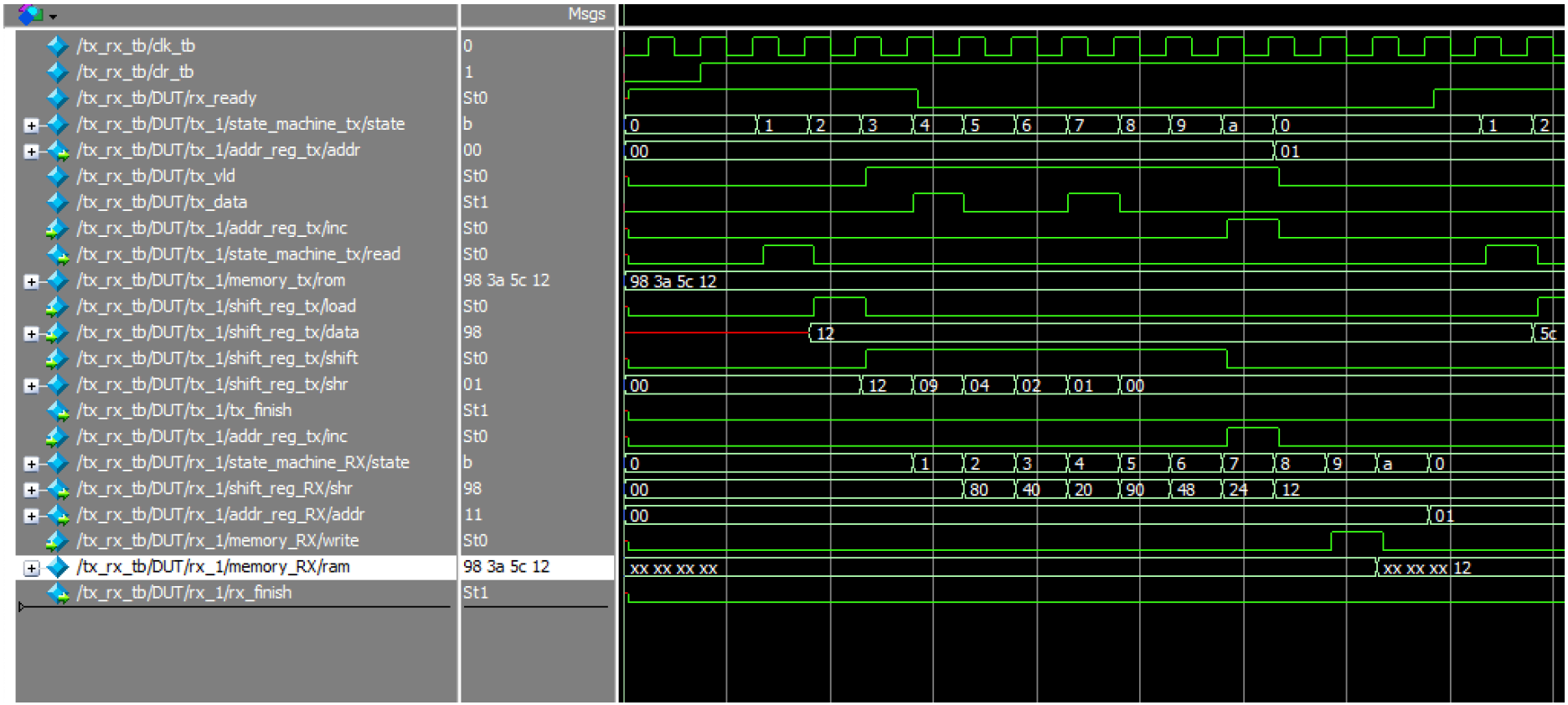
```
1  12
2  5C
3  3A
4  98
```



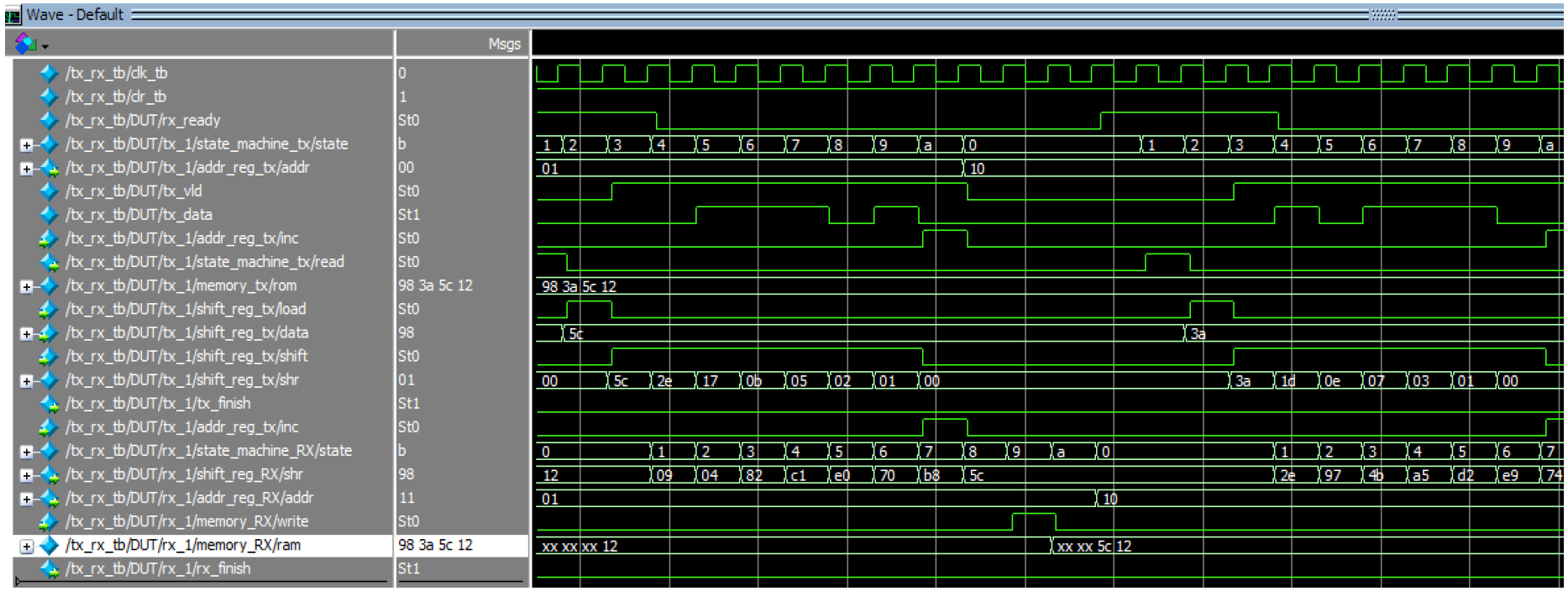
Mem_out – RX_RAM

```
1  12
2  5c
3  3a
4  98
```

דיאגרמת גלים TX_RX_TB



דיאגרמת גלים TX_RX_TB



דיאגרמת גלים TX_RX_TB