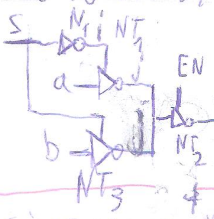
Arash Shahin
810 199 49 2

Digital Logic Design
CA2

1400/8/
ECE367

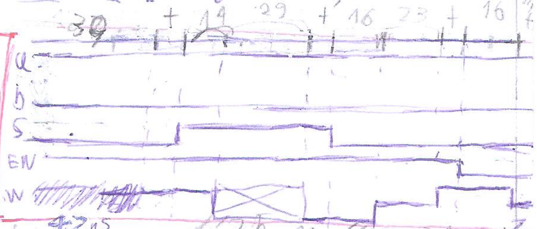
1) a)


b) NOTIF(14, 18, 16)
NOT(7, 9)

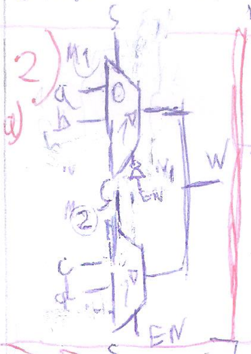
$T_o1 = N_{1(1)} + NT_{1(0)} + NT_{2(1)} = 39$ ns
$a=1$ $b=0$
$EN=1$ $S=1 \to S=0$

$T_o0 = N_{1(0)} + NT_1 + NT_2 = 43$ ns
$a=1$ $b=0$
$EN=1$ $S=0 \to S=1$

$T_o2 = NT_{2(3)} = 16$ ns
$a=1$ $b=0$
$S=1$ $EN=1 \to EN=0$



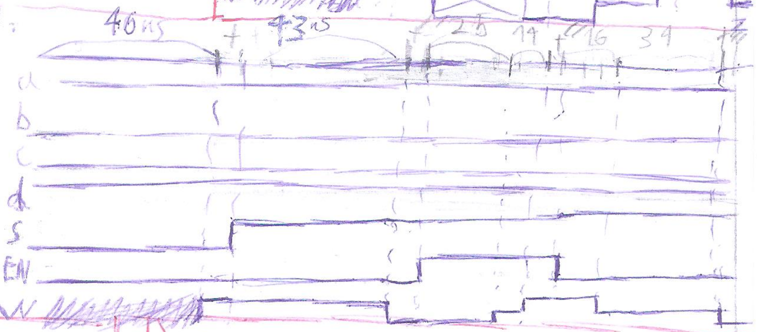
2) a) Since we are not using primitives, but our own gates, then the delay that results from changing our inputs isn't always our worst case delay, which means we'll have shorter and more realistic delays.

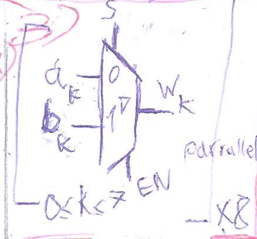
2) a)


b) Multi2(39, 43, 16)
NOT(7, 9)

$T_o1: N_{1(1)} + M_{1(1)} = 46$ ns
$a=1/b=0/c=0/d=0/S=0/EN=1 \to EN=0$

$T_o0: N_{1(1)} + M_{1(0)} = 50$ ns
$a=1/b=0/c=0/d=1/S=1/EN=1 \to EN=0$

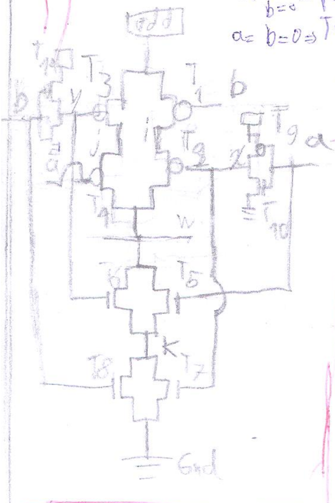


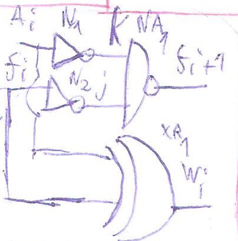
3) a)


b) Because they are all parallel to each other and so all of their delays happens simultaneously and are independent of each other.

d) Since our assign does not have a delay then it immediately changes its output. The other one takes time to change.

4) CMOS of XOR
$a=1, b=0 \Rightarrow T_o1: T_{9(2)} + T_{7(2)} + T_{5(2)} = 23$ ns
$a=b=0 \Rightarrow T_o0: T_{1(2)} + T_{2(2)} + T_{10(2)} = 25$ ns
XOR(23, 25)



a)


$f_0 = 0$
$f_{i+1} = f_i + A_i$
$f_{i+1} = \overline{\overline{f_i} \cdot \overline{A_i}}$

f_i \ A_i	0	1
0	0	1
1	1	1

f_{i+1}

f_i \ A_i	0	1
0	0	1
1	1	0

W_i

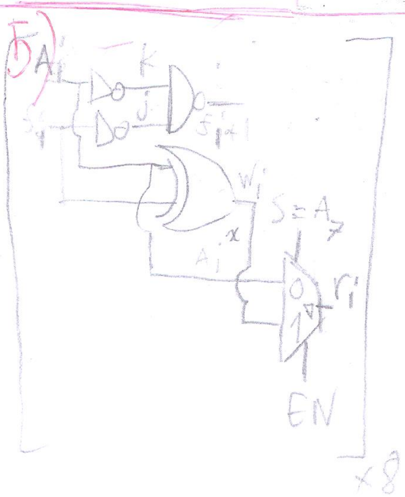
$W_i = A_i \overline{f_i} + \overline{A_i} f_i$
$W_i = A_i \oplus f_i$

We'll start from the lowest value bit, and once we reach a bit that's 1, we'll change the flag from 0 to 1 until the end.
If the flag is 0, then the bits should stay the same. Once it's 1, we should complement the bits.

b) f_{i+1}: $A_i=f_i=1 \Rightarrow T_o1: N_1^{9} + NA_1^{14} = 23$ (23, 17)
$A_i=f_i=0 \Rightarrow T_o0: N_1^{7} + NA_1^{10} = 17$
W_i: Same as XR_i: (23, 25)

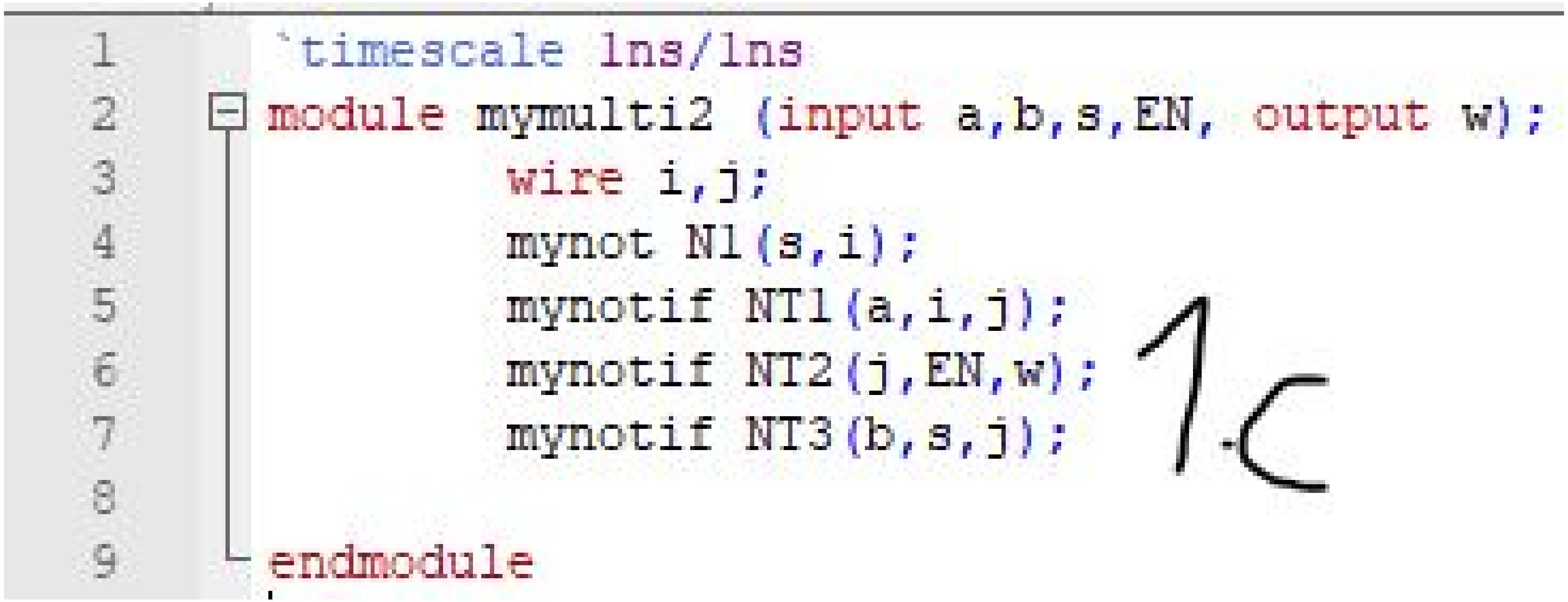
c) Worst delay: 00000001
$\underset{(1)}{23} \times 7 + \underset{(1)}{23} = 23 \times 8 = 184$ ns
From f_{i+1} gates
from the last XOR

d) Since we are using the worst case delay in our assign, that means that in most cases, the actual delay is shorter.

5) a)


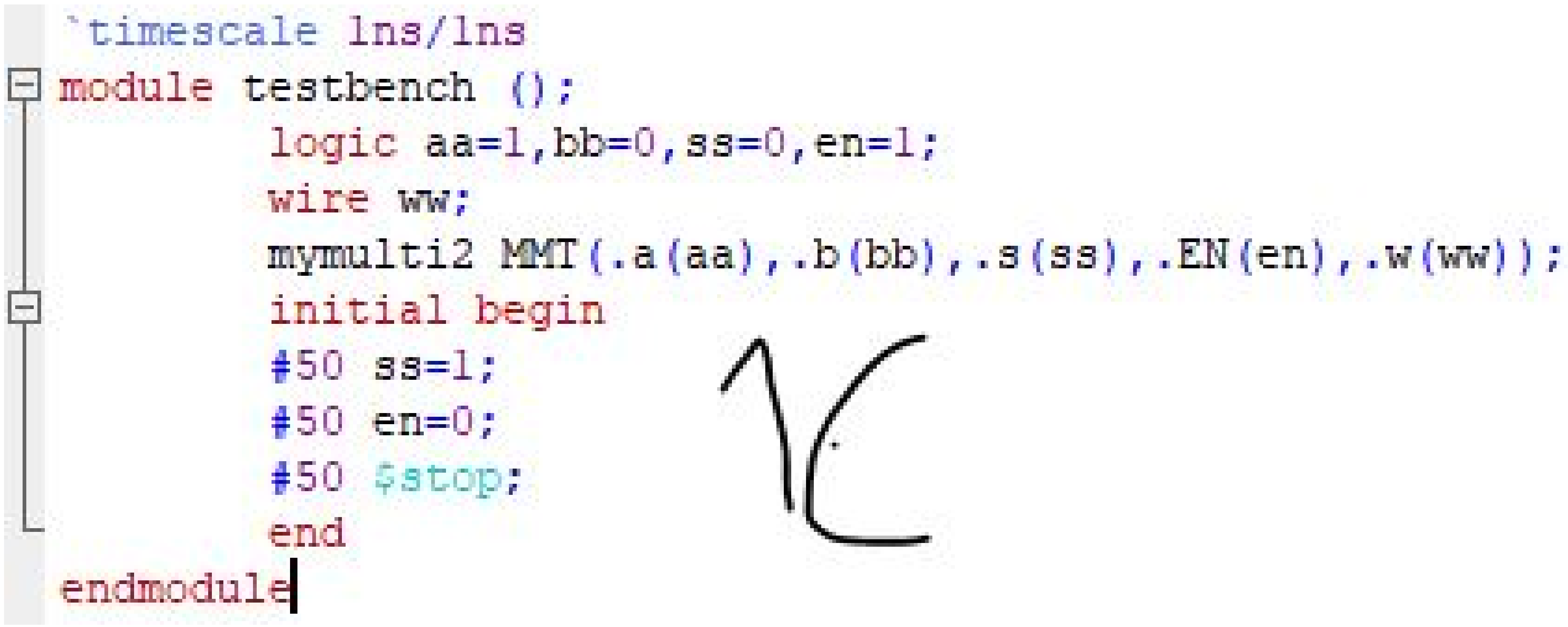
b) Worst case delay: 10000001
The last one will become known since $A_7=1$.
A negative number will give us the most delay. And in the previous problem, the worst delay was from 00000001 so the one written that the worst delay. Its delay is: 218 according to model sim

e) Yosys: 1 BUF, 12 NAND, 25 NOR, 12 NOT: 50 Gates
Mine: 8 XOR, 16 NOT, 8 NAND, 8 Multiplexer (which has 3 NOTIFs and 1 Not)
→ breaking down our multiplexer:
8 XOR, 24 Not, 8 NAND, 24 NOTIF: 64 Gates



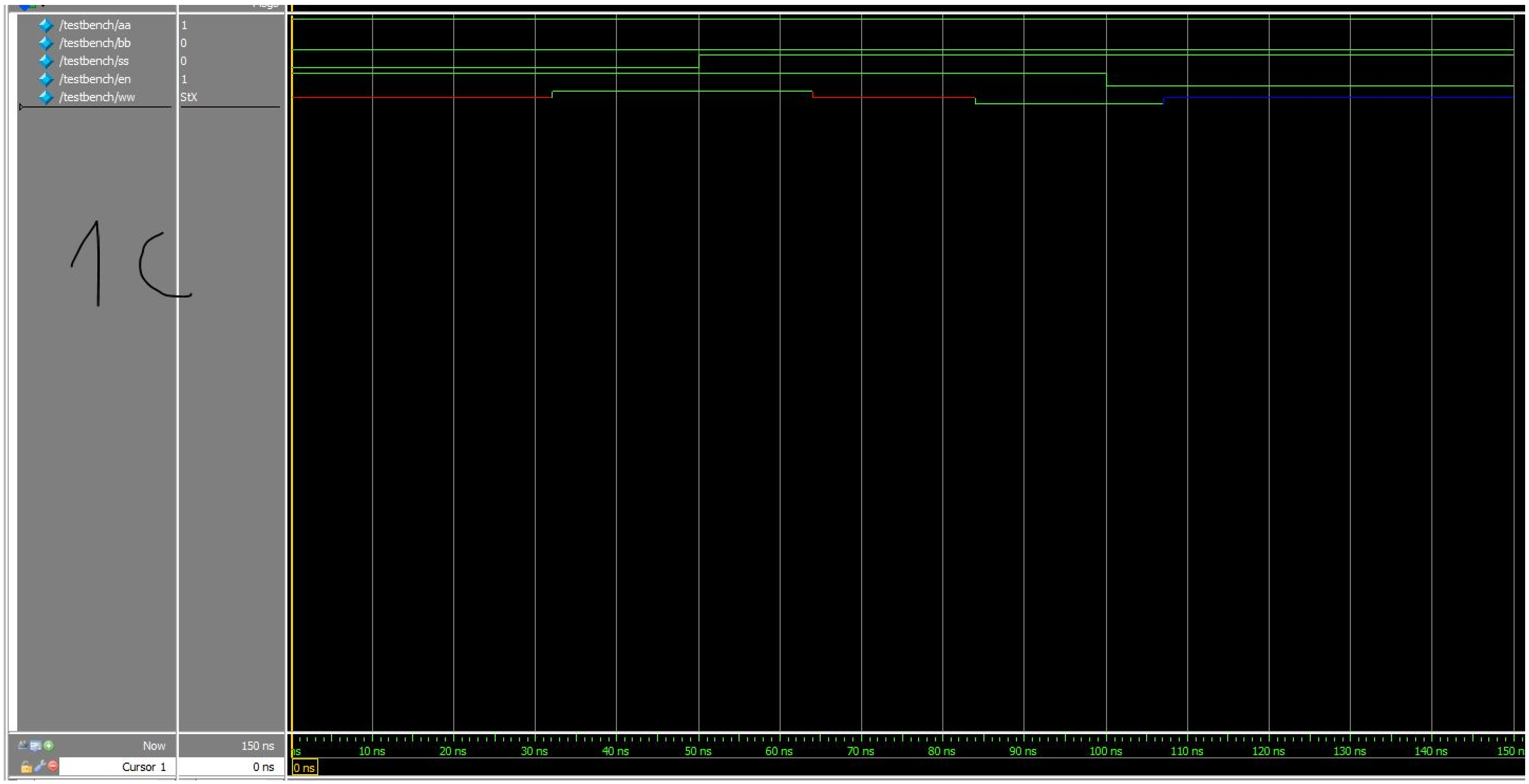
```
`timescale 1ns/1ns
module mymulti2 (input a,b,s,EN, output w);
        wire i,j;
        mynot N1(s,i);
        mynotif NT1(a,i,j);
        mynotif NT2(j,EN,w);
        mynotif NT3(b,s,j);

endmodule
```

1C

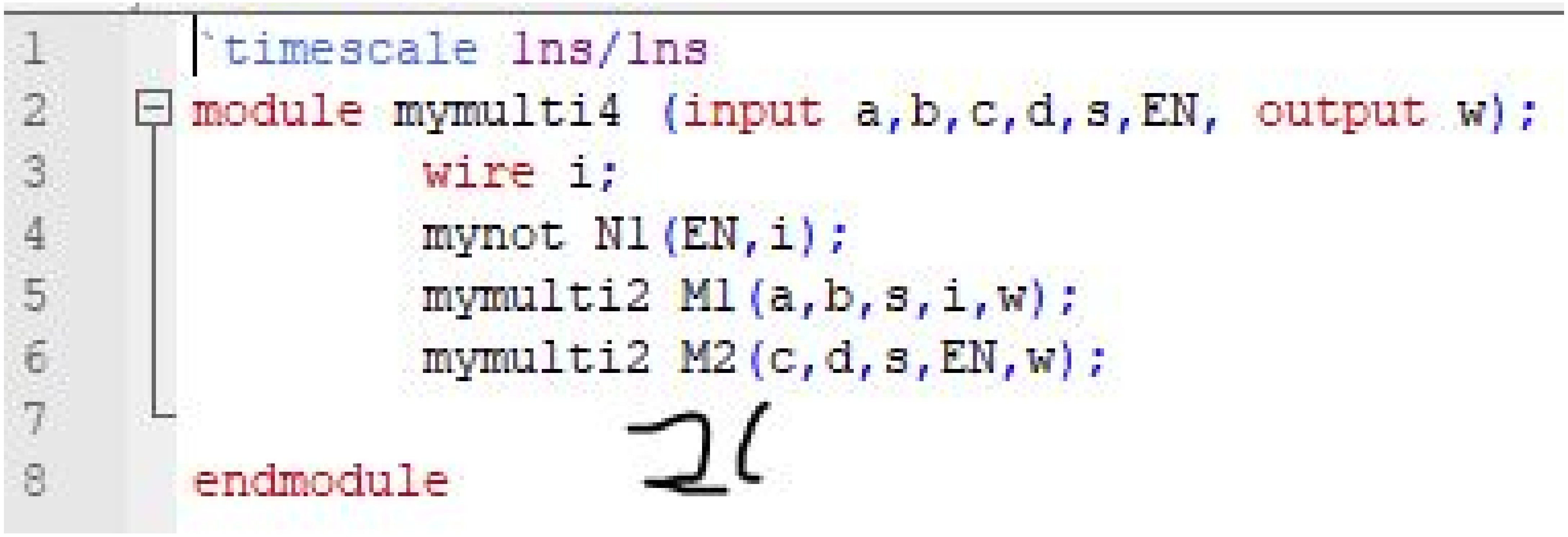


```
`timescale 1ns/1ns
module testbench ();
          logic aa=1,bb=0,ss=0,en=1;
          wire ww;
          mymulti2 MMT(.a(aa),.b(bb),.s(ss),.EN(en),.w(ww));
          initial begin
          #50 ss=1;
          #50 en=0;
          #50 $stop;
          end
endmodule
```

[illegible]

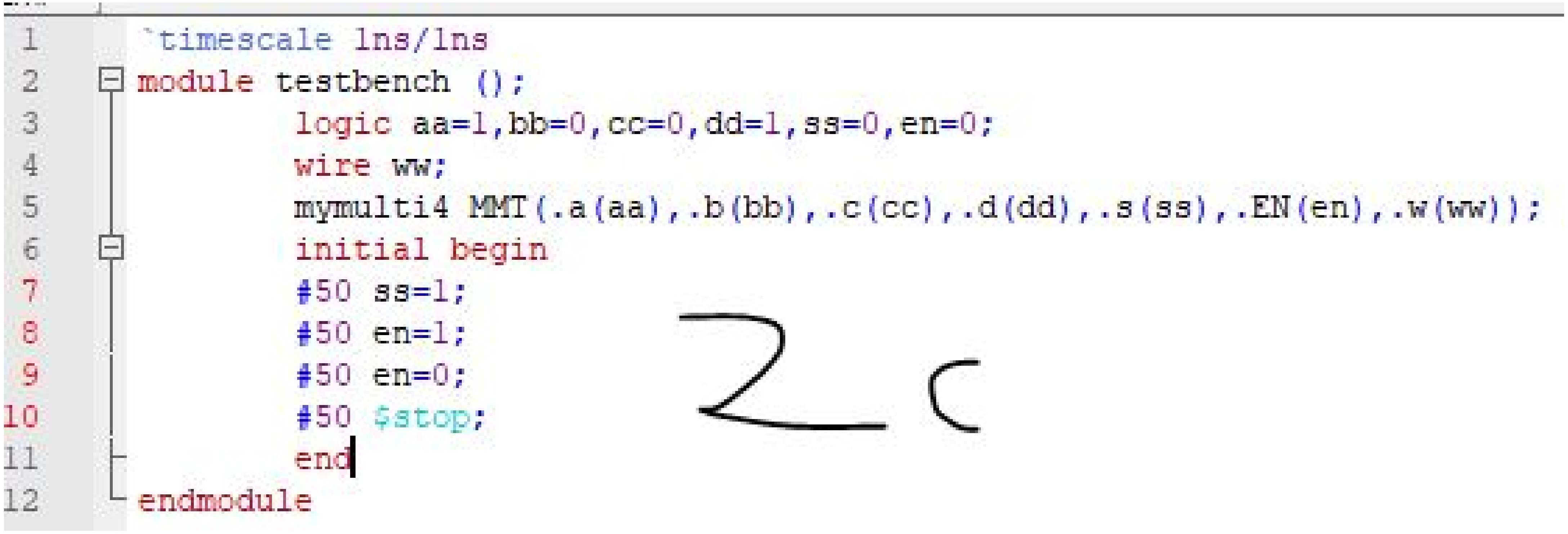


1C
/testbench/aa 1
/testbench/bb 0
/testbench/ss 0
/testbench/en 1
/testbench/ww StX
Now 150 ns
Cursor 1 0 ns
ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns
140 ns
150 n
0 ns



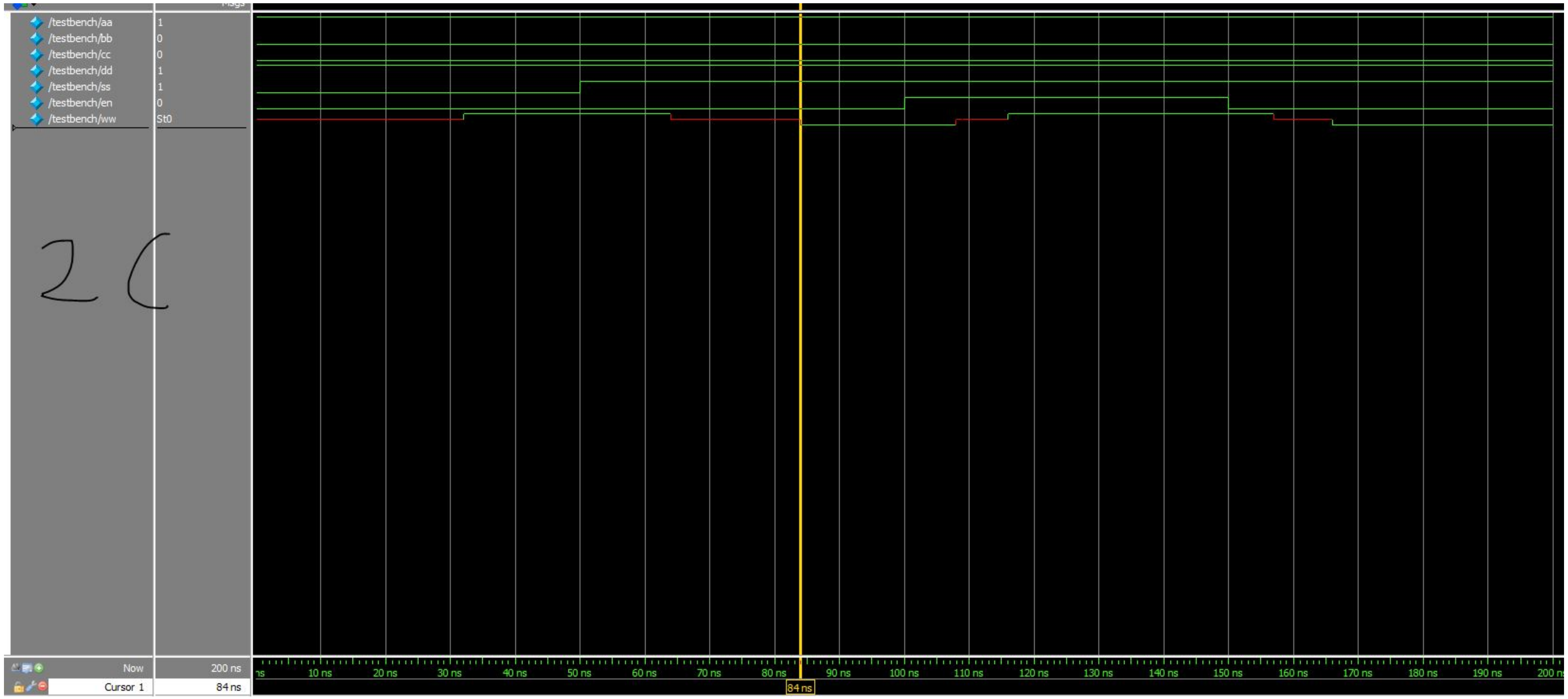
```
`timescale 1ns/1ns
module mymulti4 (input a,b,c,d,s,EN, output w);
        wire i;
        mynot N1(EN,i);
        mymulti2 M1(a,b,s,i,w);
        mymulti2 M2(c,d,s,EN,w);

endmodule
```

2(

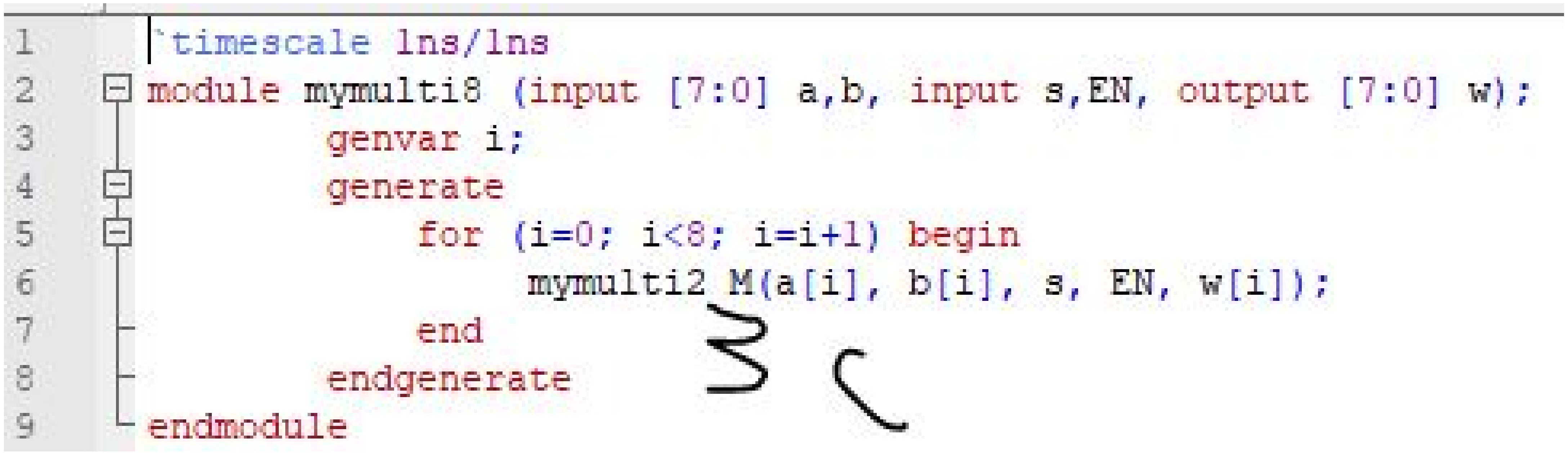


```
`timescale 1ns/1ns
module testbench ();
        logic aa=1,bb=0,cc=0,dd=1,ss=0,en=0;
        wire ww;
        mymulti4 MMT(.a(aa),.b(bb),.c(cc),.d(dd),.s(ss),.EN(en),.w(ww));
        initial begin
        #50 ss=1;
        #50 en=1;
        #50 en=0;
        #50 $stop;
        end
endmodule
```

2c

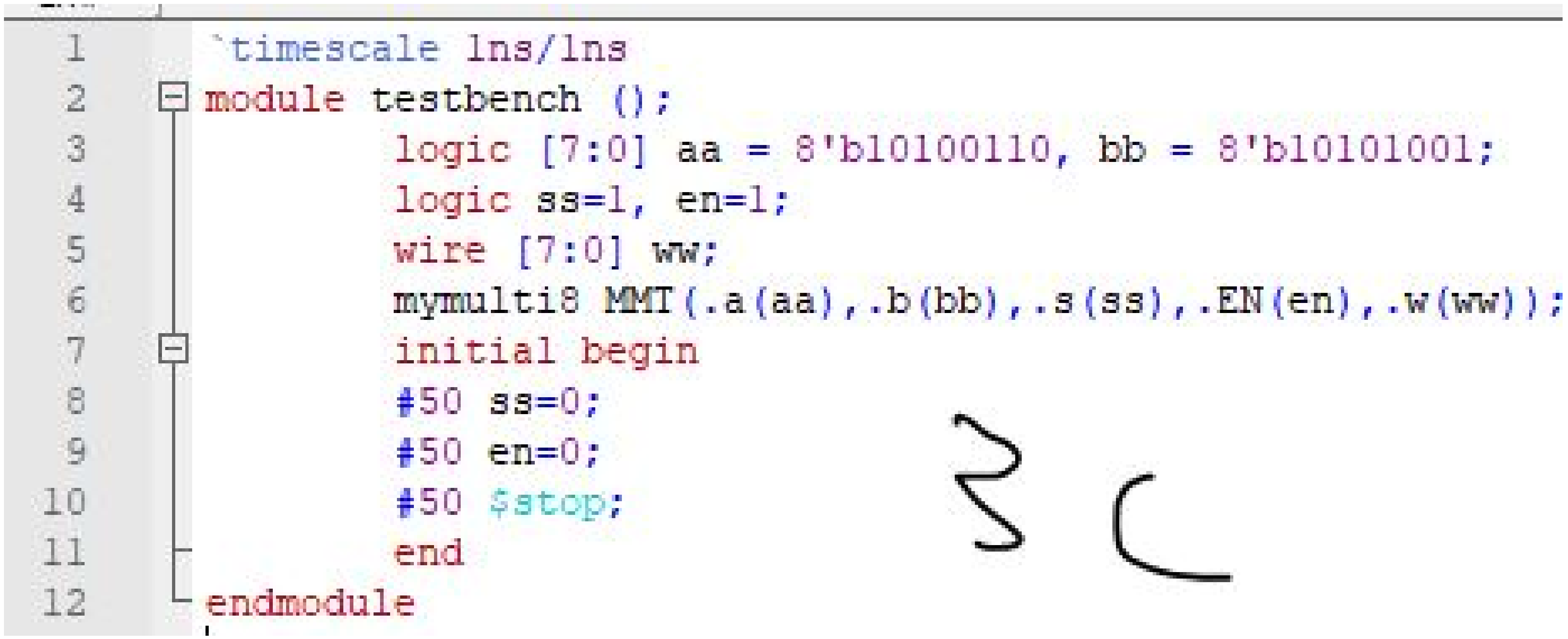

/testbench/aa 1
/testbench/bb 0
/testbench/cc 0
/testbench/dd 1
/testbench/ss 1
/testbench/en 0
/testbench/ww St0
Now 200 ns
Cursor 1 84 ns
ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120 ns 130 ns 140 ns 150 ns 160 ns 170 ns 180 ns 190 ns 200 ns
84 ns

2C



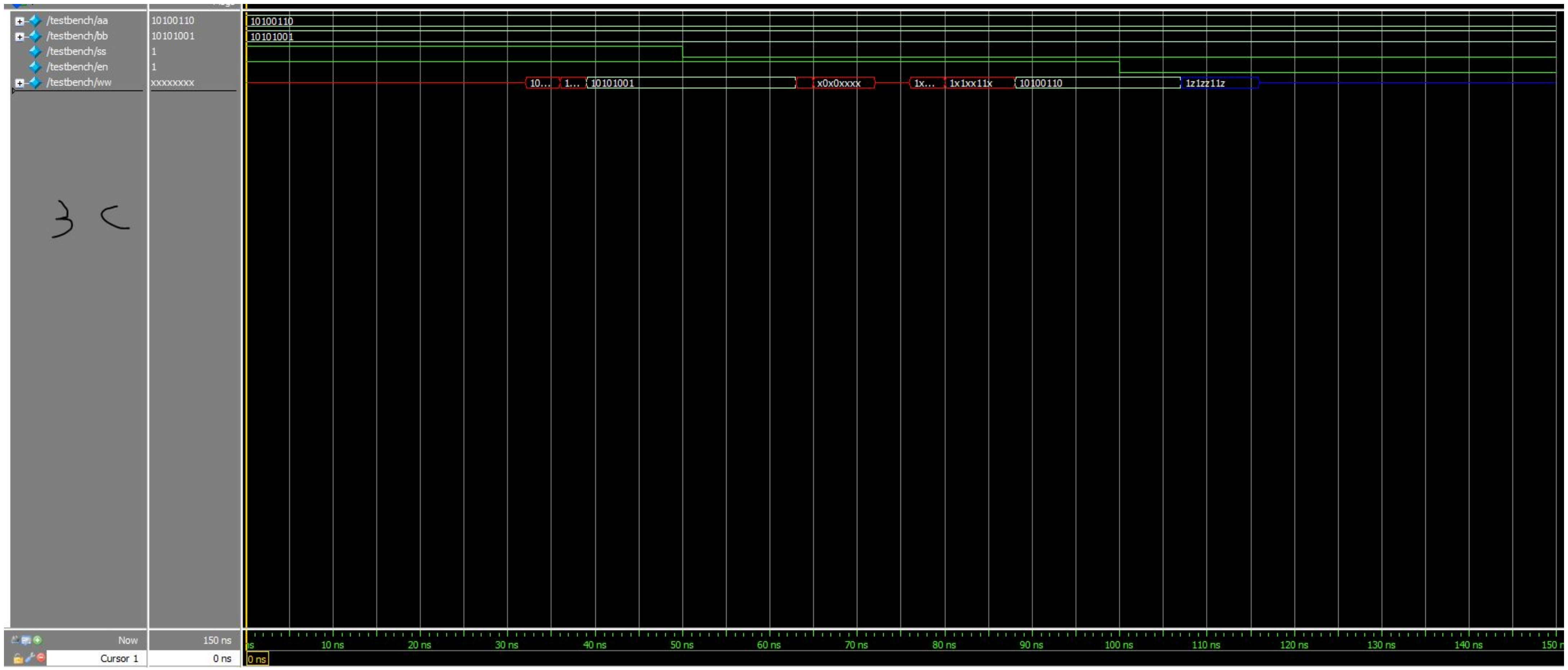
```
`timescale 1ns/1ns
module mymulti8 (input [7:0] a,b, input s,EN, output [7:0] w);
        genvar i;
        generate
            for (i=0; i<8; i=i+1) begin
                mymulti2 M(a[i], b[i], s, EN, w[i]);
            end
        endgenerate
endmodule
```

3 C



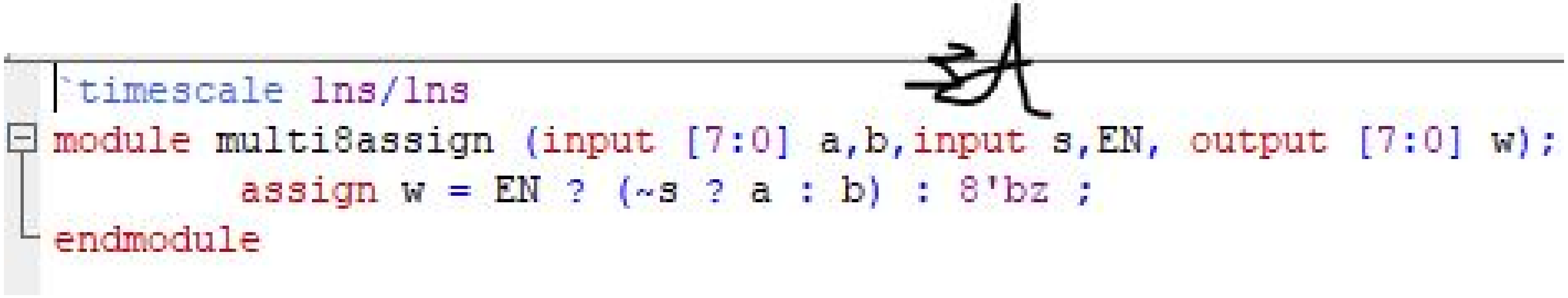
```
`timescale 1ns/1ns
module testbench ();
        logic [7:0] aa = 8'b10100110, bb = 8'b10101001;
        logic ss=1, en=1;
        wire [7:0] ww;
        mymulti8 MMT(.a(aa),.b(bb),.s(ss),.EN(en),.w(ww));
        initial begin
        #50 ss=0;
        #50 en=0;
        #50 $stop;
        end
endmodule
```

3 C

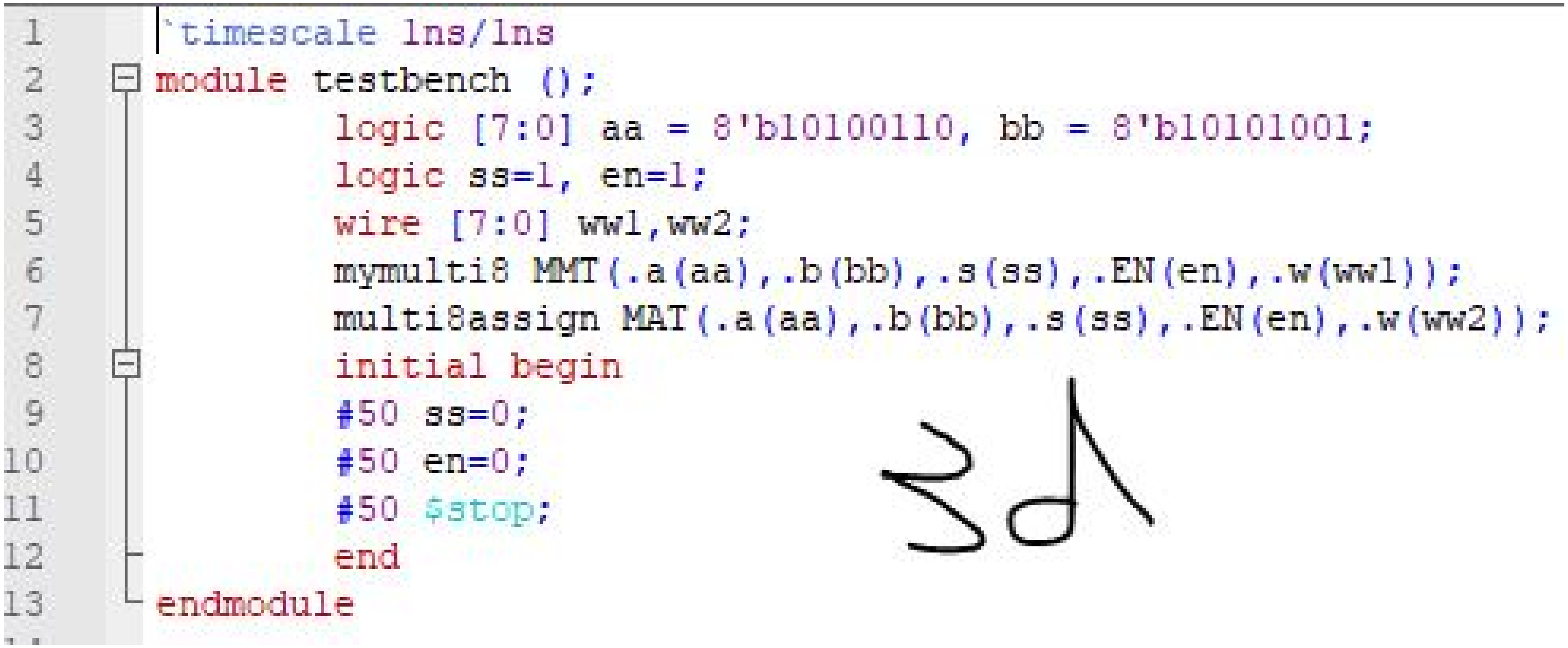


/testbench/aa
/testbench/bb
/testbench/ss
/testbench/en
/testbench/ww
10100110
10101001
1
1
xxxxxxxx
10100110
10101001
10...
1...
10101001
x0x0xxxx
1x...
1x1xx11x
10100110
1z1zz11z
Now
150 ns
Cursor 1
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns
140 ns
0 ns

3 C

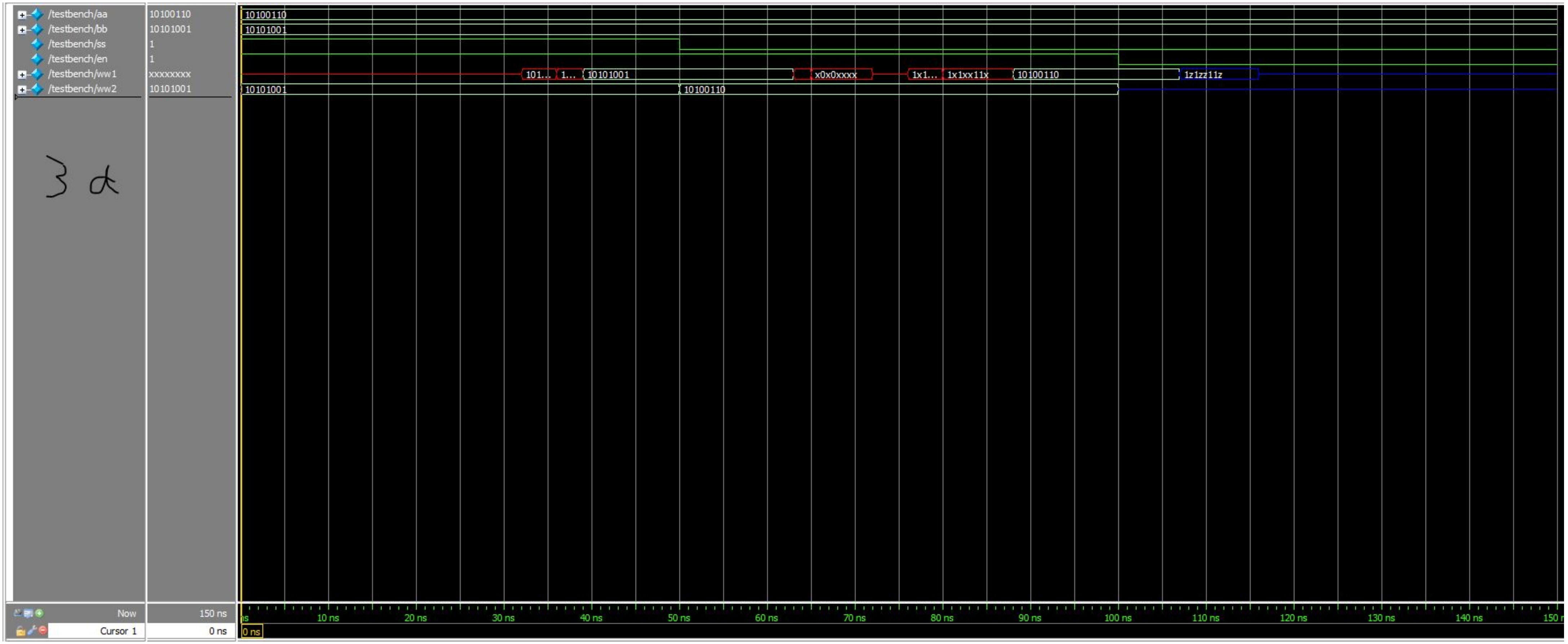


```
`timescale 1ns/1ns
module multi8assign (input [7:0] a,b,input s,EN, output [7:0] w);
          assign w = EN ? (~s ? a : b) : 8'bz ;
endmodule
```



```
`timescale 1ns/1ns
module testbench ();
        logic [7:0] aa = 8'b10100110, bb = 8'b10101001;
        logic ss=1, en=1;
        wire [7:0] ww1,ww2;
        mymulti8 MMT(.a(aa),.b(bb),.s(ss),.EN(en),.w(ww1));
        multi8assign MAT(.a(aa),.b(bb),.s(ss),.EN(en),.w(ww2));
        initial begin
        #50 ss=0;
        #50 en=0;
        #50 $stop;
        end
endmodule
```

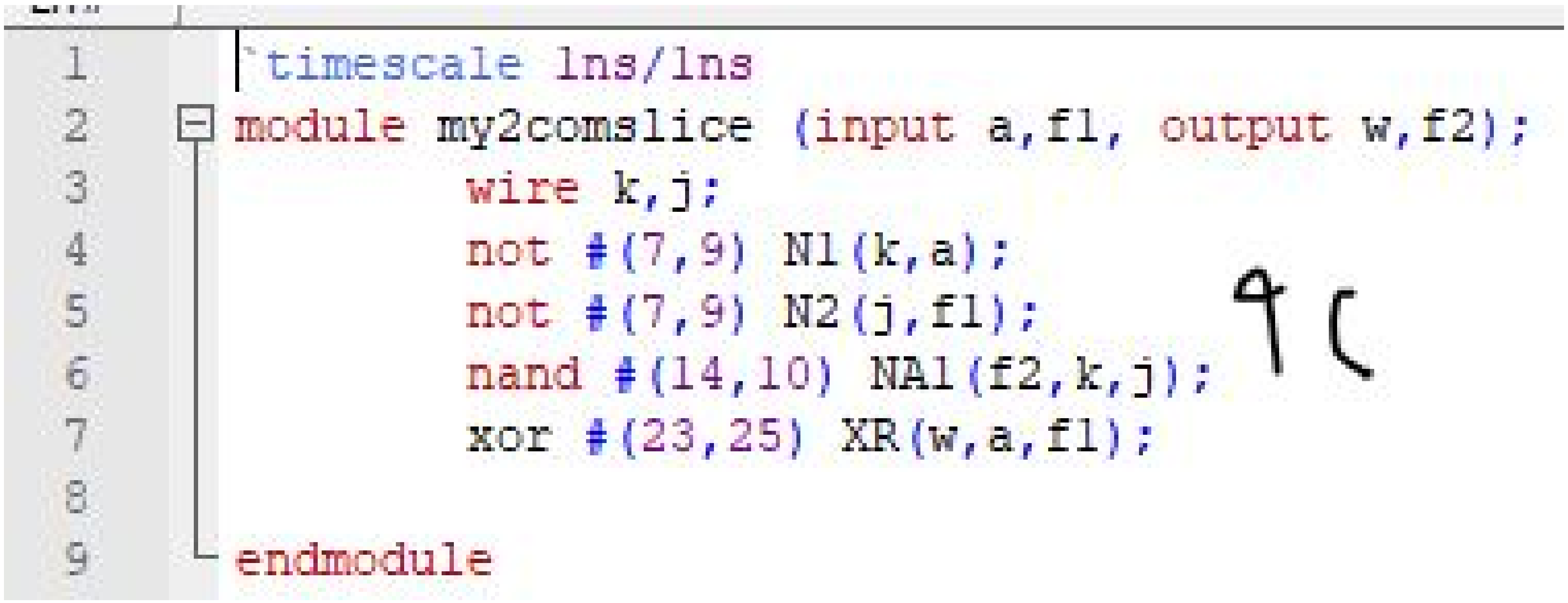
3d



```
/testbench/aa    10100110
/testbench/bb    10101001
/testbench/ss    1
/testbench/en    1
/testbench/ww1   xxxxxxxx
/testbench/ww2   10101001

Now       150 ns
Cursor 1  0 ns
```

```
aa:  10100110
bb:  10101001
ww1: 101...  1...  10101001  x0x0xxxx  1x1...  1x1xx11x  10100110  1z1zz11z
ww2: 10101001  10100110
```

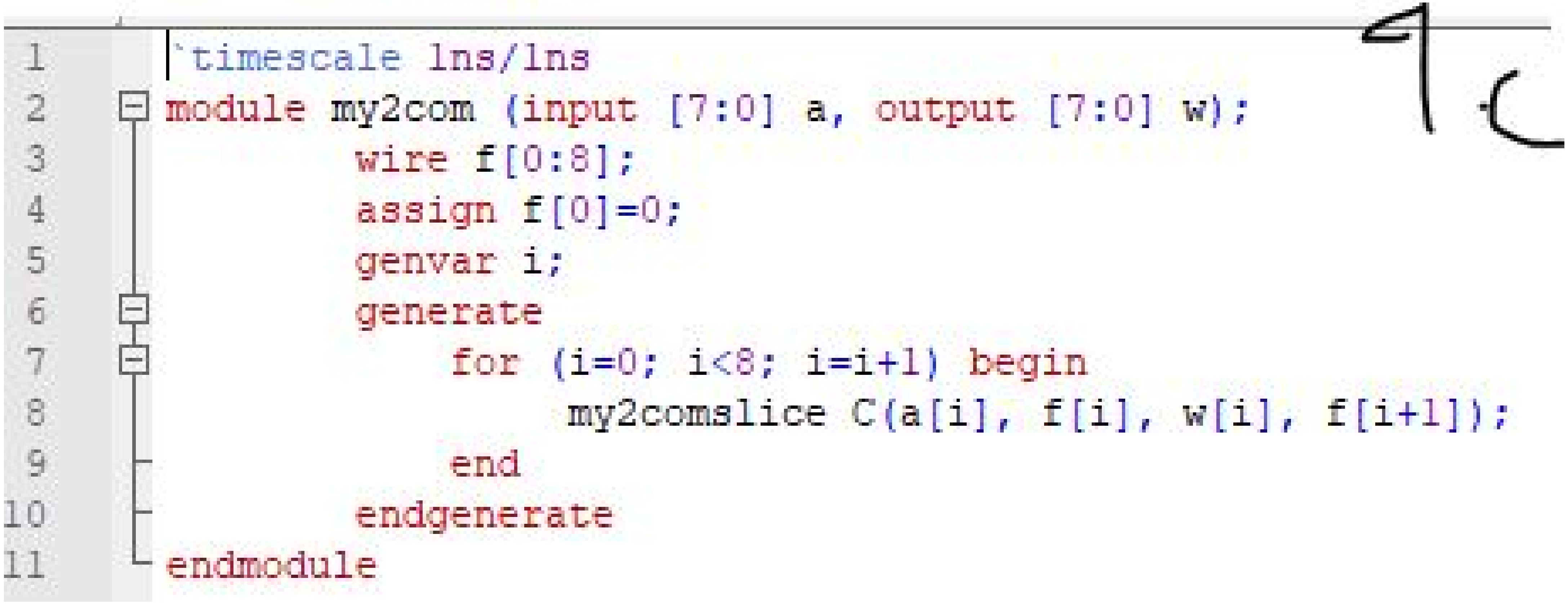
3 d



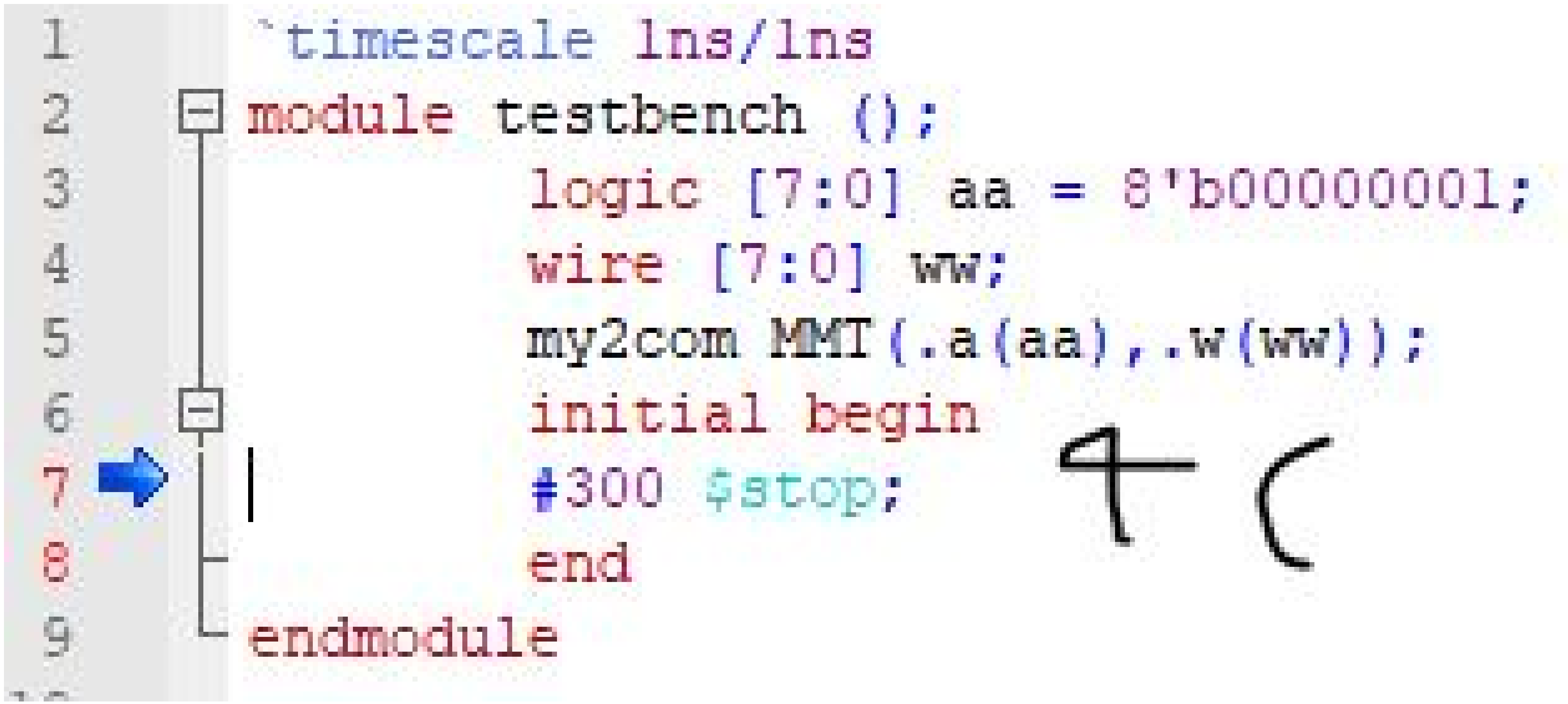
```
`timescale 1ns/1ns
module my2comslice (input a,f1, output w,f2);
            wire k,j;
            not #(7,9) N1(k,a);
            not #(7,9) N2(j,f1);
            nand #(14,10) NA1(f2,k,j);
            xor #(23,25) XR(w,a,f1);

endmodule
```

9 (

9c

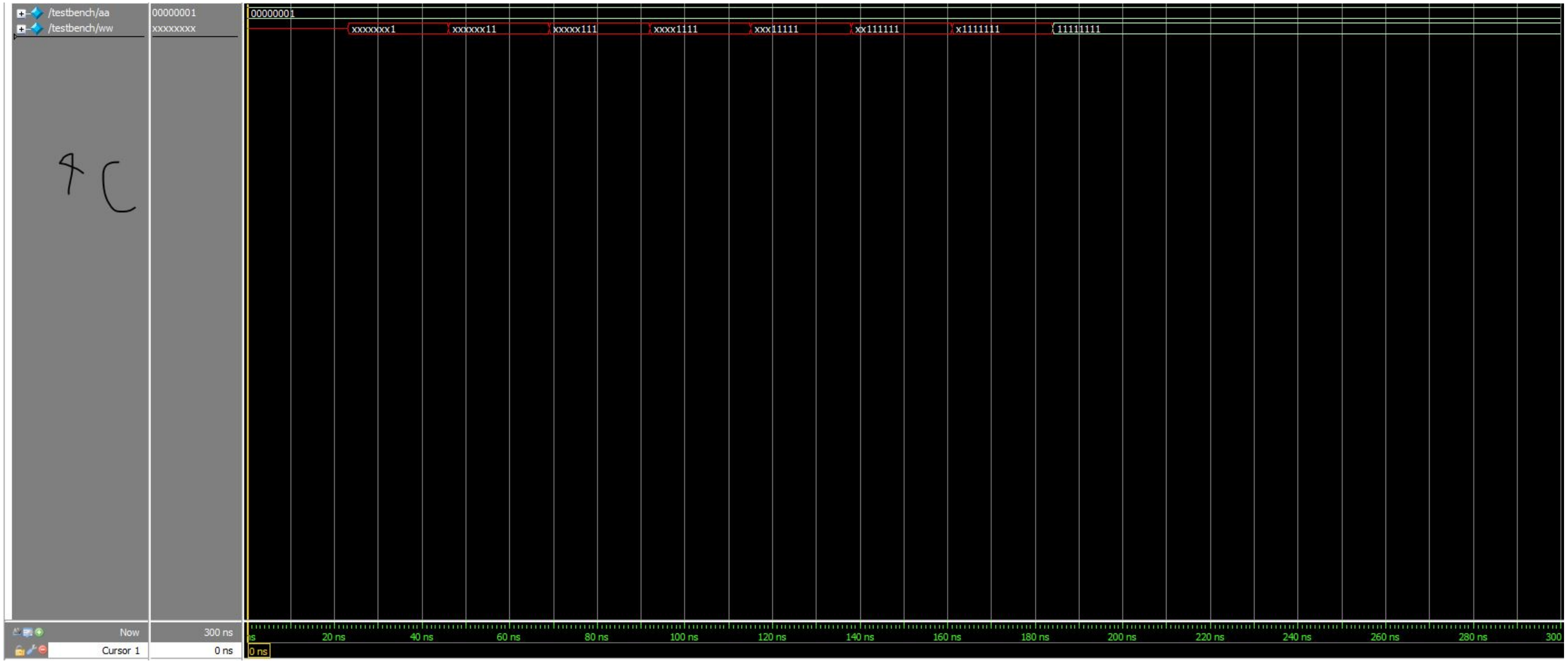


```
`timescale 1ns/1ns
module my2com (input [7:0] a, output [7:0] w);
        wire f[0:8];
        assign f[0]=0;
        genvar i;
        generate
            for (i=0; i<8; i=i+1) begin
                my2comslice C(a[i], f[i], w[i], f[i+1]);
            end
        endgenerate
endmodule
```

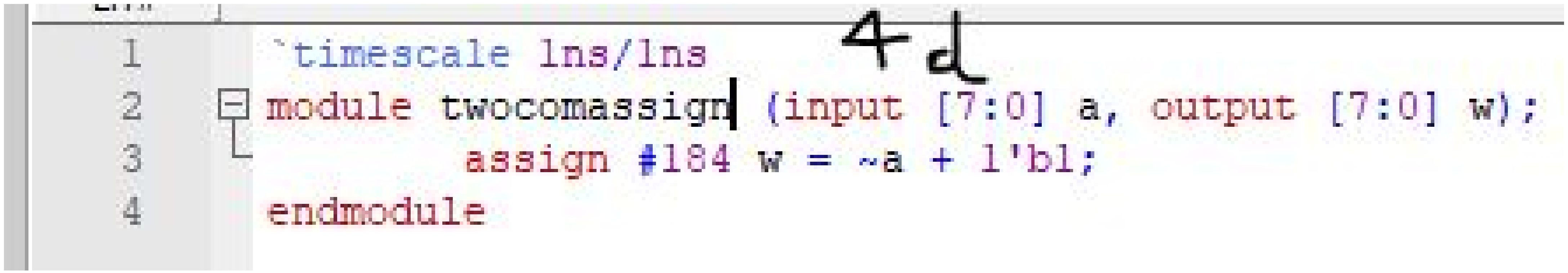


```
`timescale 1ns/1ns
module testbench ();
        logic [7:0] aa = 8'b00000001;
        wire [7:0] ww;
        my2com MMT(.a(aa),.w(ww));
        initial begin
        #300 $stop;
        end
endmodule
```

4 (

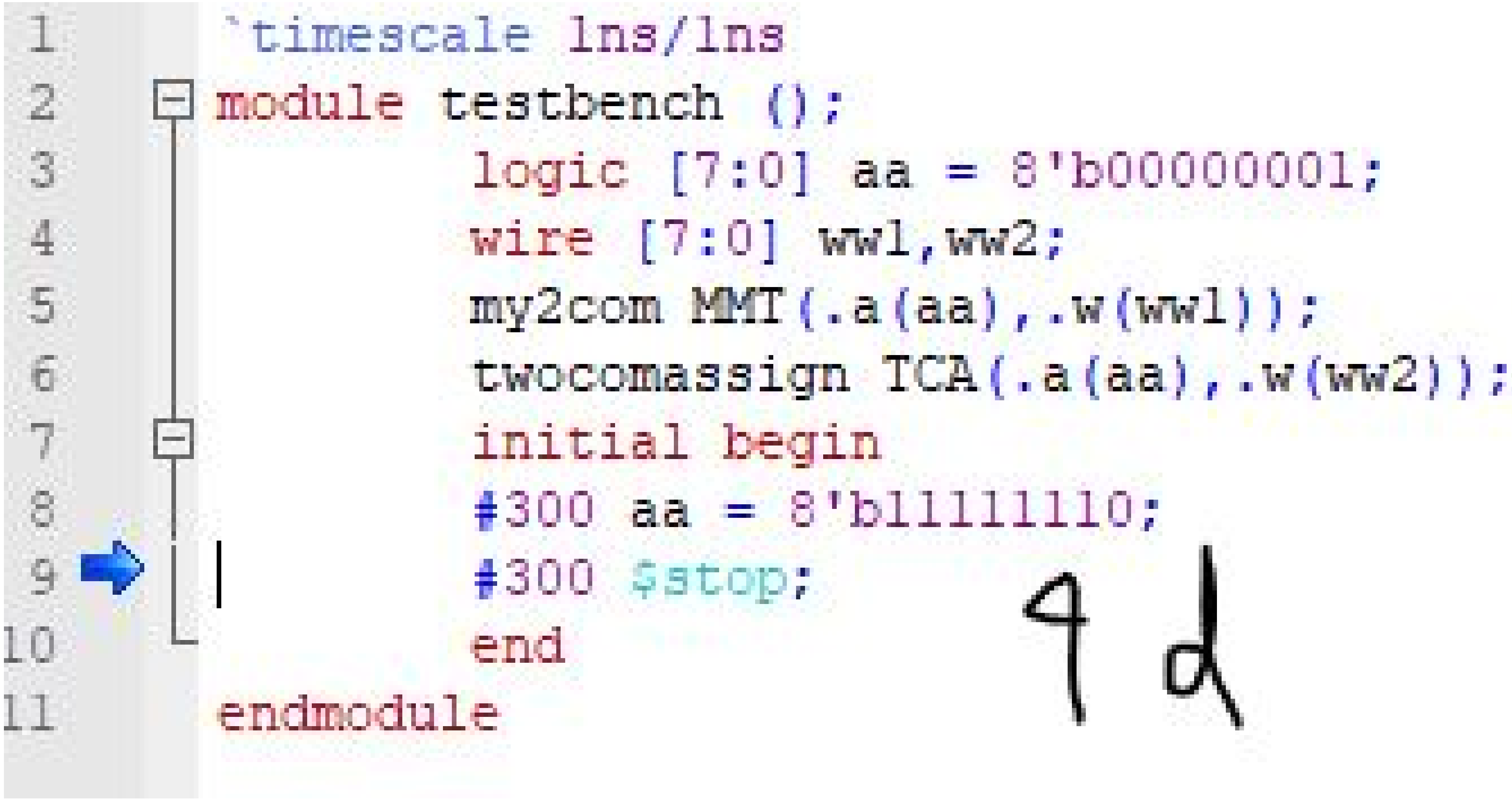


```
/testbench/aa   00000001   00000001
/testbench/ww   xxxxxxxx   xxxxxxx1 | xxxxxx11 | xxxxx111 | xxxx1111 | xxx11111 | xx111111 | x1111111 | 11111111

Now       300 ns
Cursor 1  0 ns     0 ns
```



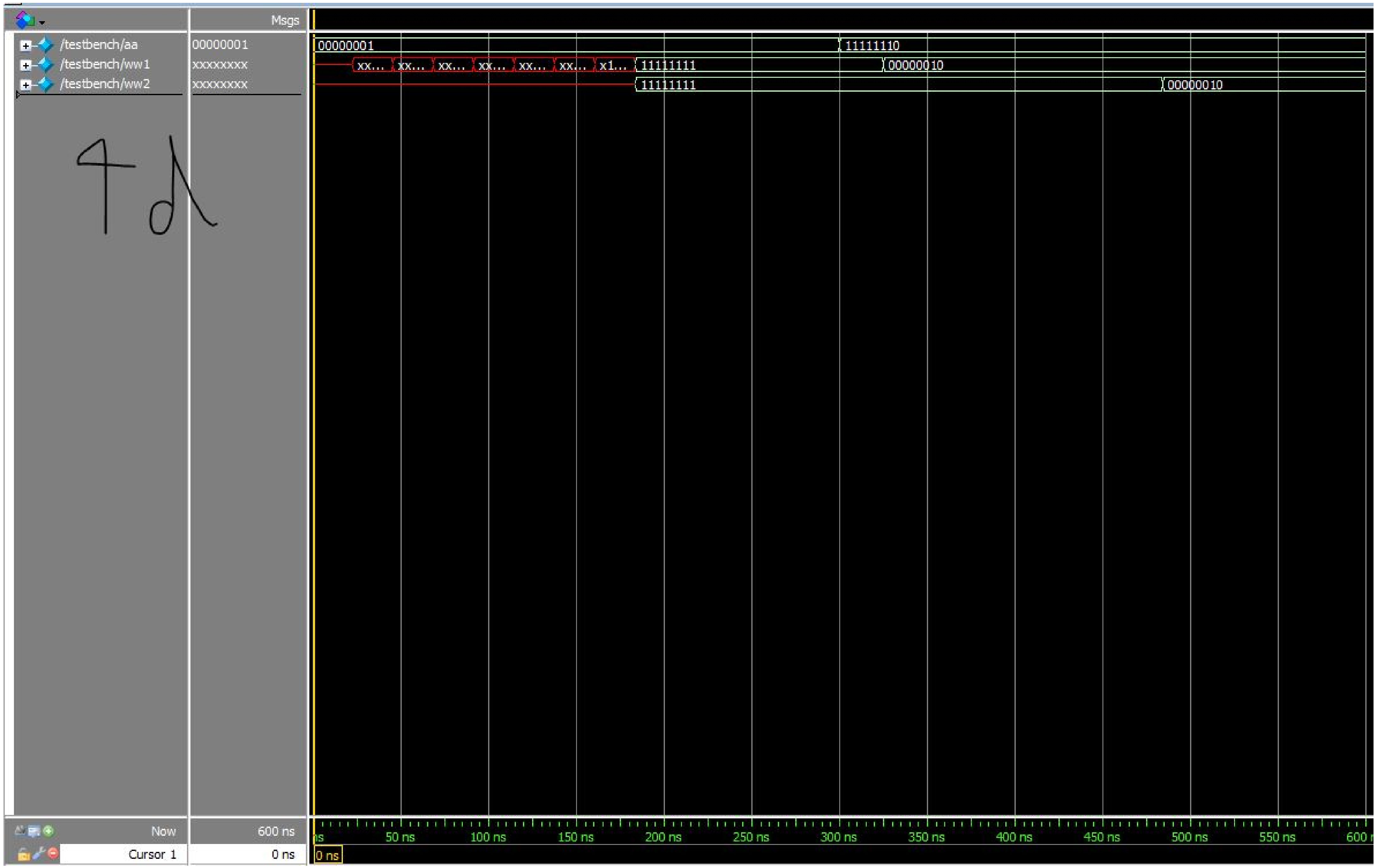
4d

```
`timescale 1ns/1ns
module twocomassign (input [7:0] a, output [7:0] w);
        assign #184 w = ~a + 1'b1;
endmodule
```



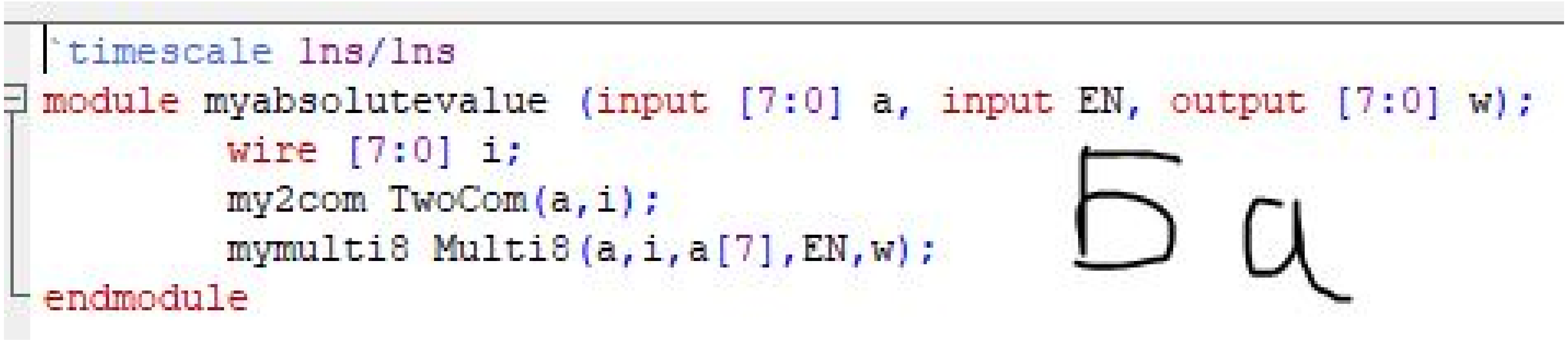
```
`timescale 1ns/1ns
module testbench ();
        logic [7:0] aa = 8'b00000001;
        wire [7:0] ww1,ww2;
        my2com MMT(.a(aa),.w(ww1));
        twocomassign TCA(.a(aa),.w(ww2));
        initial begin
        #300 aa = 8'b11111110;
        #300 $stop;
        end
endmodule
```

9d



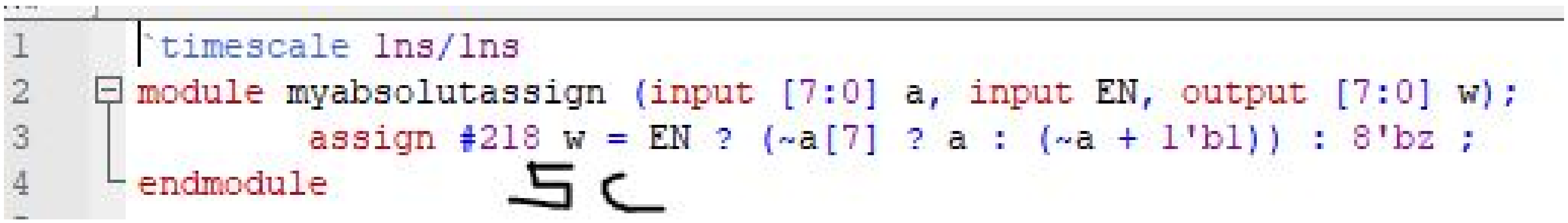
```
                   Msgs
/testbench/aa      00000001   00000001                                              11111110
/testbench/ww1     xxxxxxxx   xx... xx... xx... xx... xx... xx... x1... 11111111          00000010
/testbench/ww2     xxxxxxxx                                              11111111                             00000010

Now                600 ns     ns  50 ns  100 ns  150 ns  200 ns  250 ns  300 ns  350 ns  400 ns  450 ns  500 ns  550 ns  600 n
Cursor 1           0 ns       0 ns
```

4d



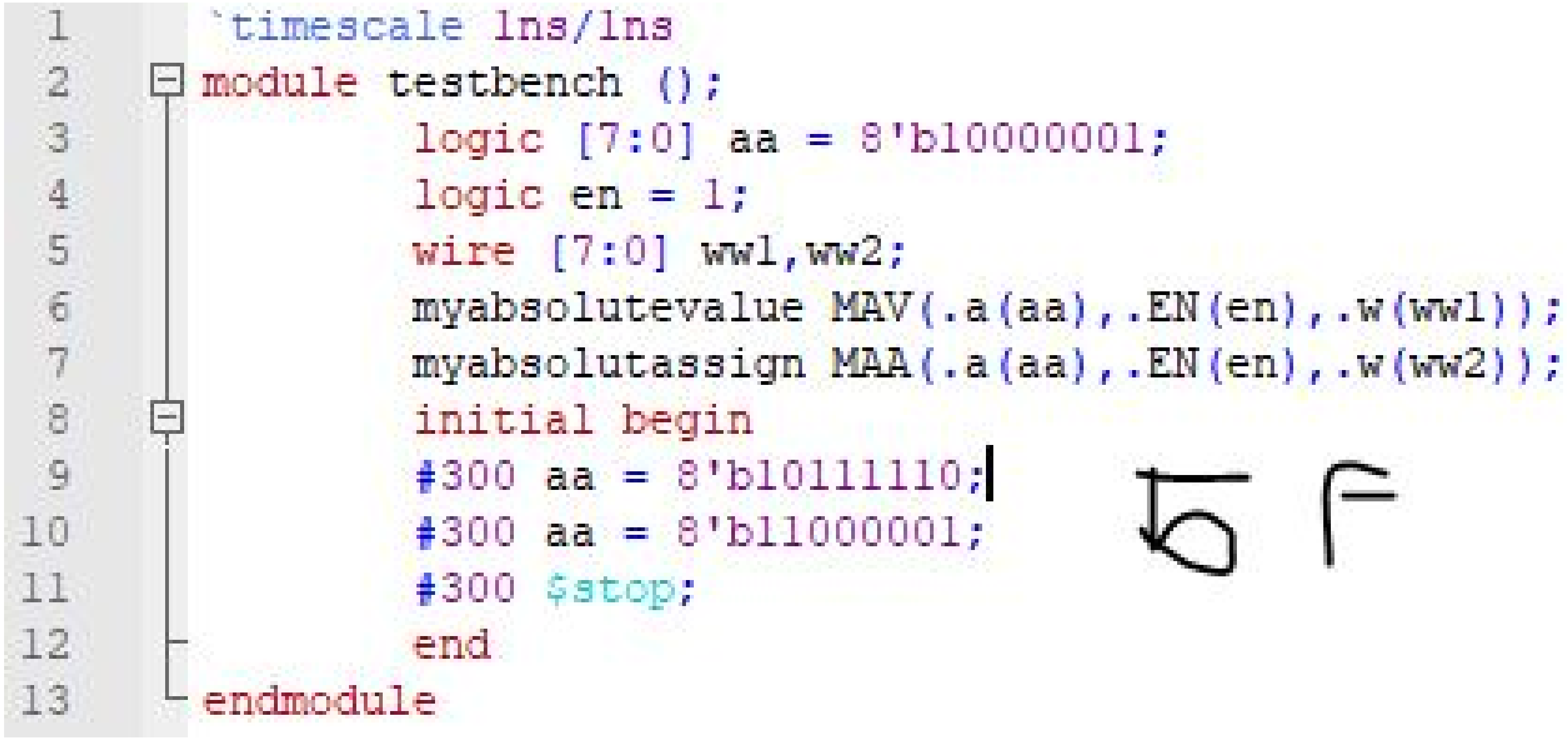
```
`timescale 1ns/1ns
module myabsolutevalue (input [7:0] a, input EN, output [7:0] w);
        wire [7:0] i;
        my2com TwoCom(a,i);
        mymulti8 Multi8(a,i,a[7],EN,w);
endmodule
```

Ба



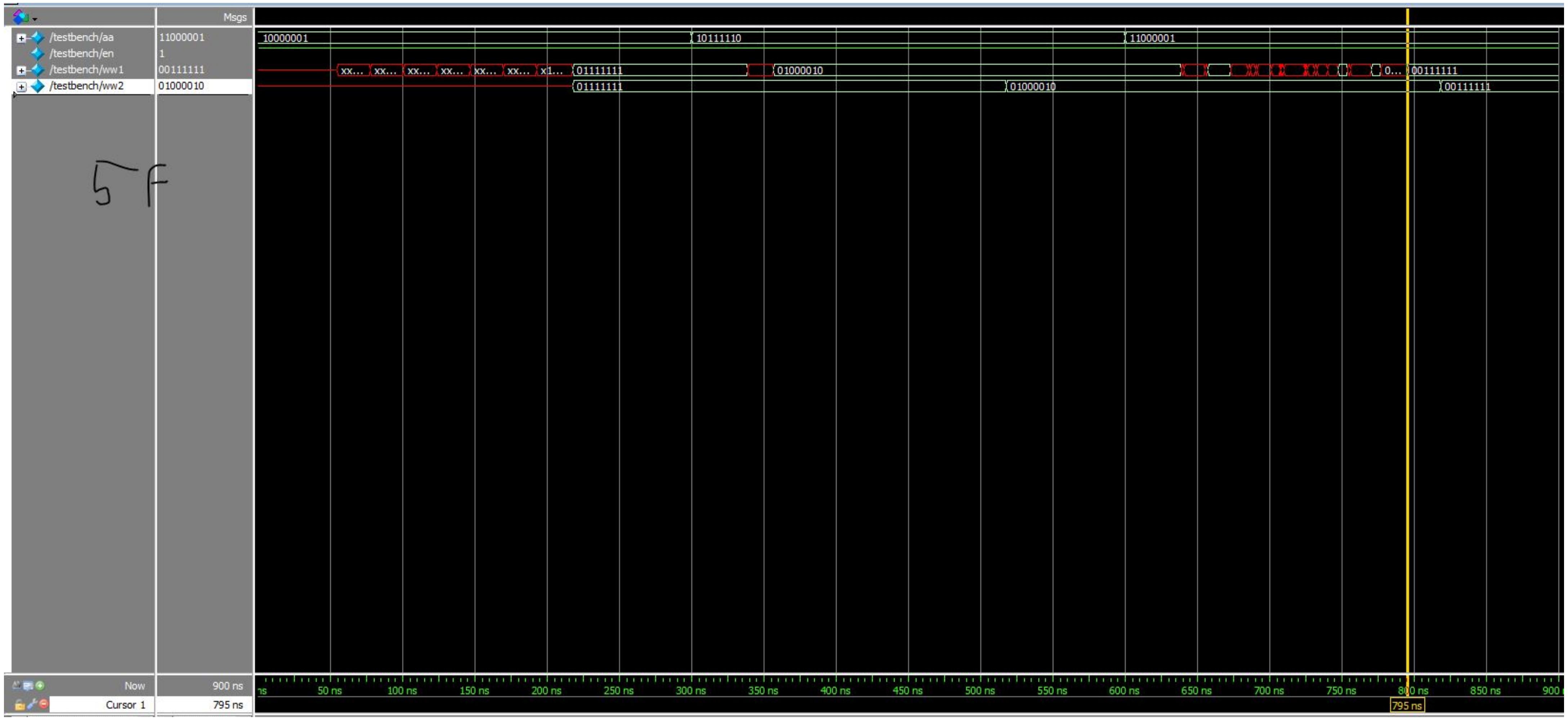
```
`timescale 1ns/1ns
module myabsolutassign (input [7:0] a, input EN, output [7:0] w);
        assign #218 w = EN ? (~a[7] ? a : (~a + 1'b1)) : 8'bz ;
endmodule
```

5 C



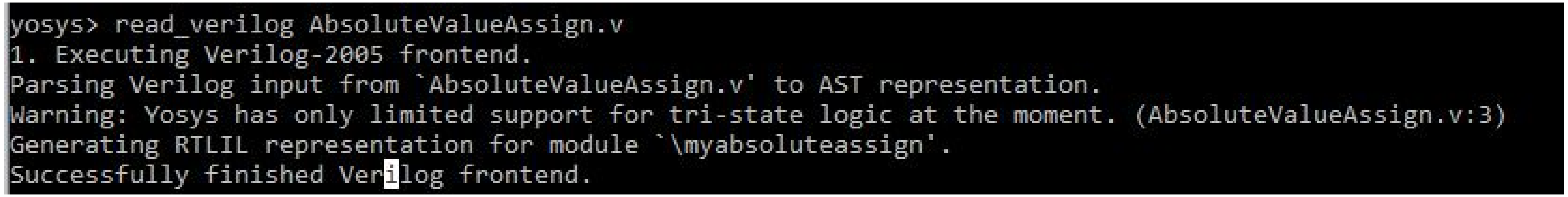
```verilog
`timescale 1ns/1ns
module testbench ();
            logic [7:0] aa = 8'b10000001;
            logic en = 1;
            wire [7:0] ww1,ww2;
            myabsolutevalue MAV(.a(aa),.EN(en),.w(ww1));
            myabsolutassign MAA(.a(aa),.EN(en),.w(ww2));
            initial begin
            #300 aa = 8'b10111110;
            #300 aa = 8'b11000001;
            #300 $stop;
            end
endmodule
```

[illegible]

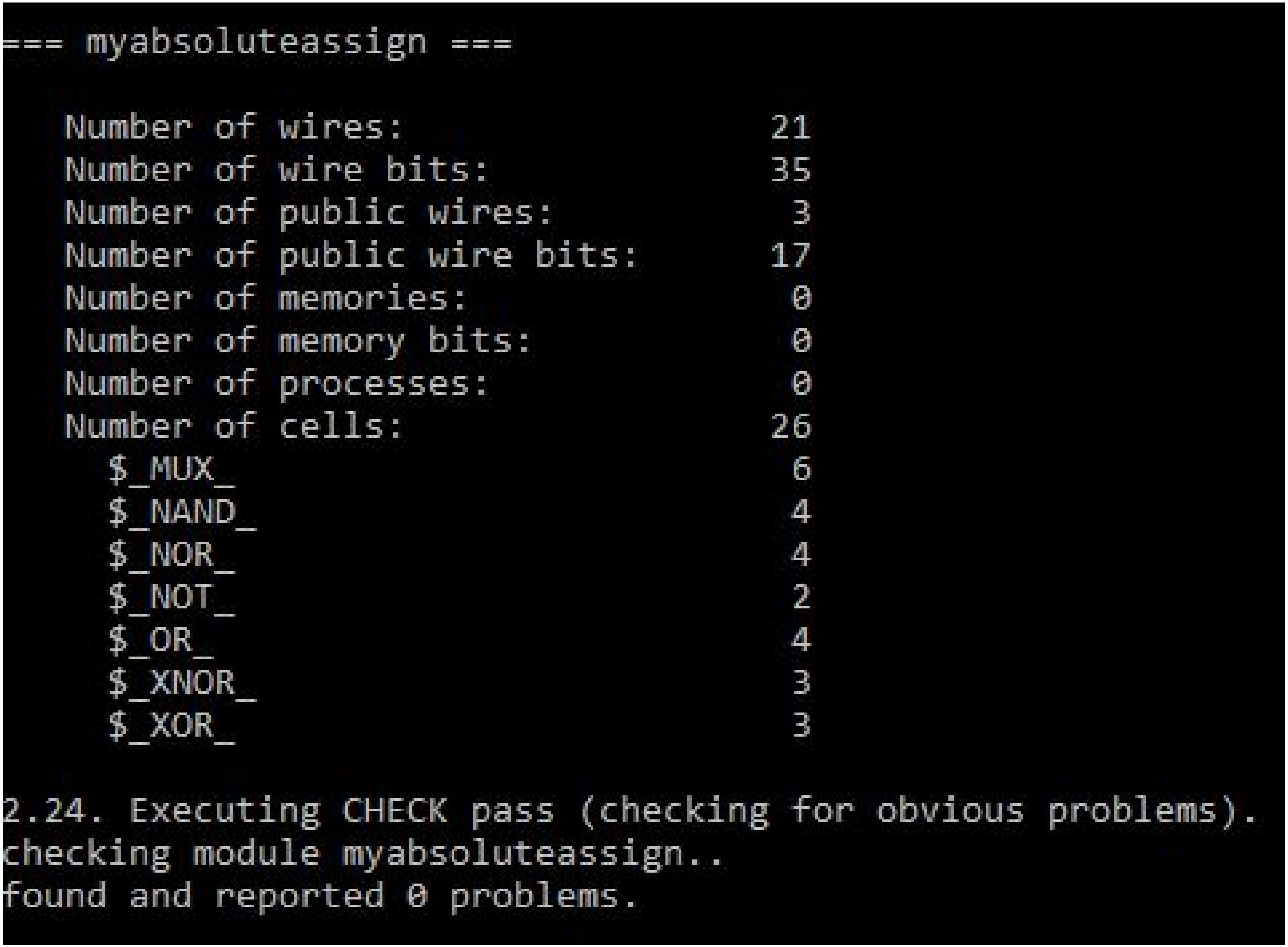


```
                      Msgs
/testbench/aa         11000001
/testbench/en         1
/testbench/ww1        00111111
/testbench/ww2        01000010

Now                   900 ns
Cursor 1              795 ns
```

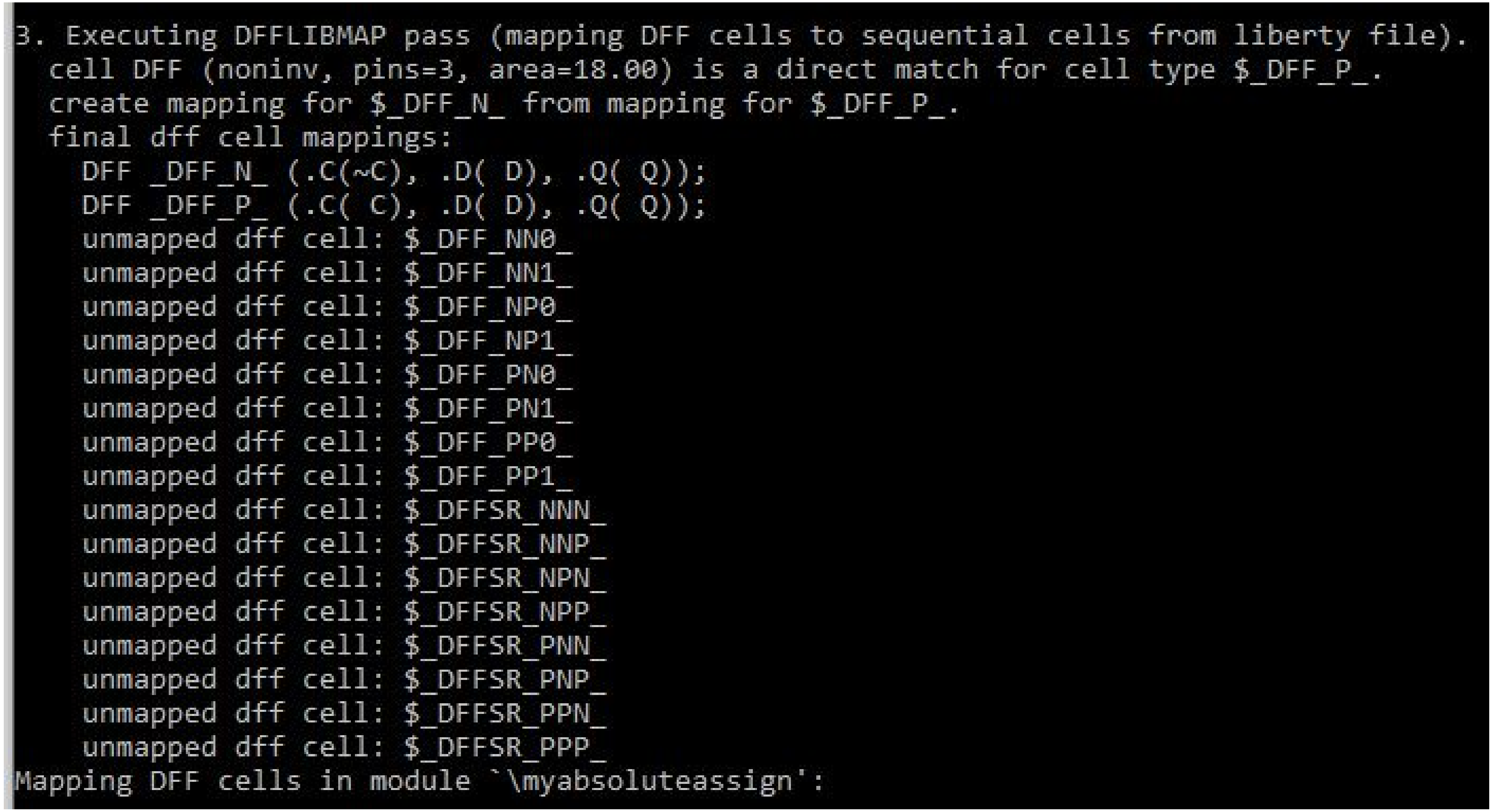
5F



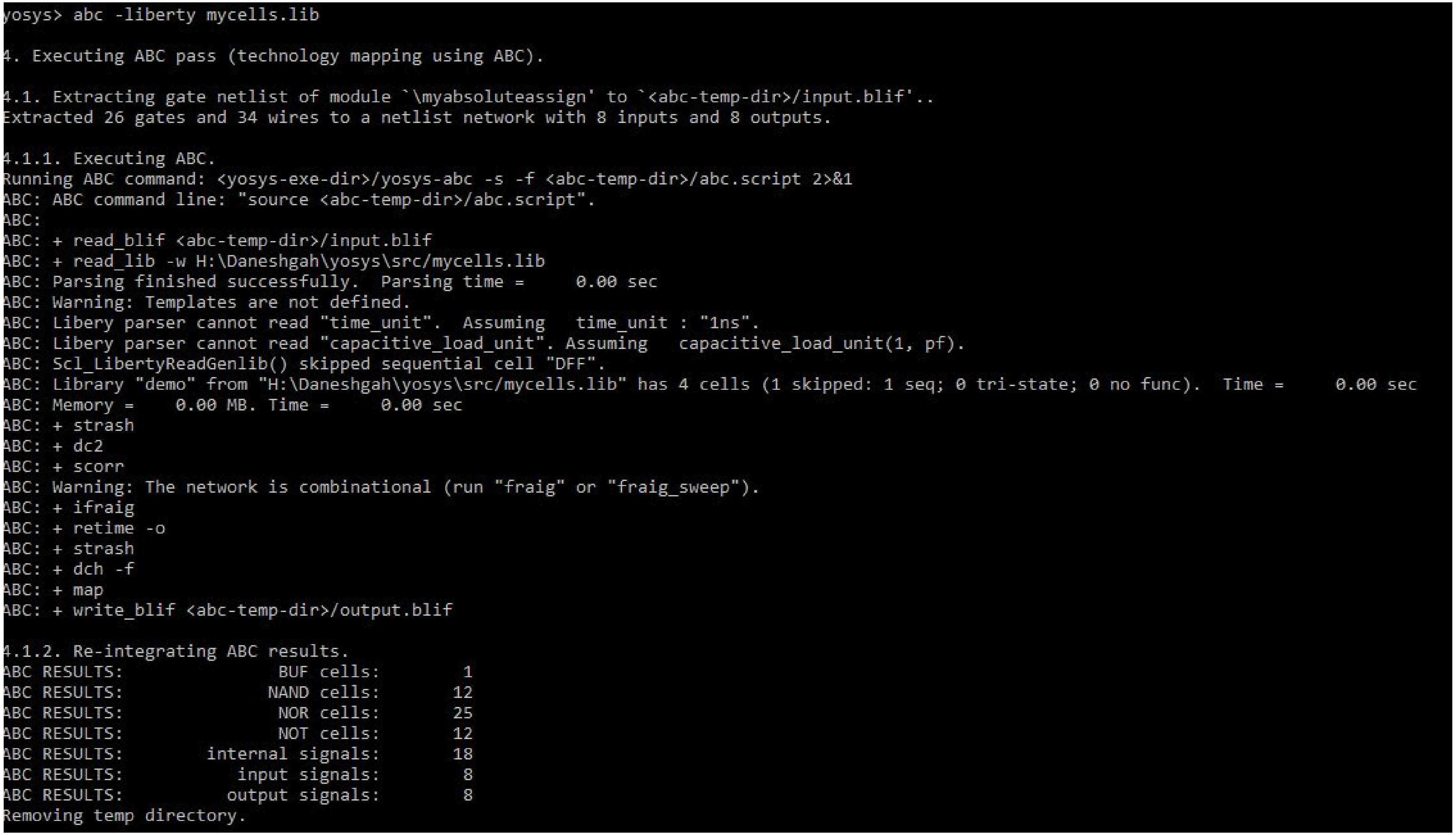
```
yosys> read_verilog AbsoluteValueAssign.v
1. Executing Verilog-2005 frontend.
Parsing Verilog input from `AbsoluteValueAssign.v' to AST representation.
Warning: Yosys has only limited support for tri-state logic at the moment. (AbsoluteValueAssign.v:3)
Generating RTLIL representation for module `\myabsoluteassign'.
Successfully finished Verilog frontend.
```



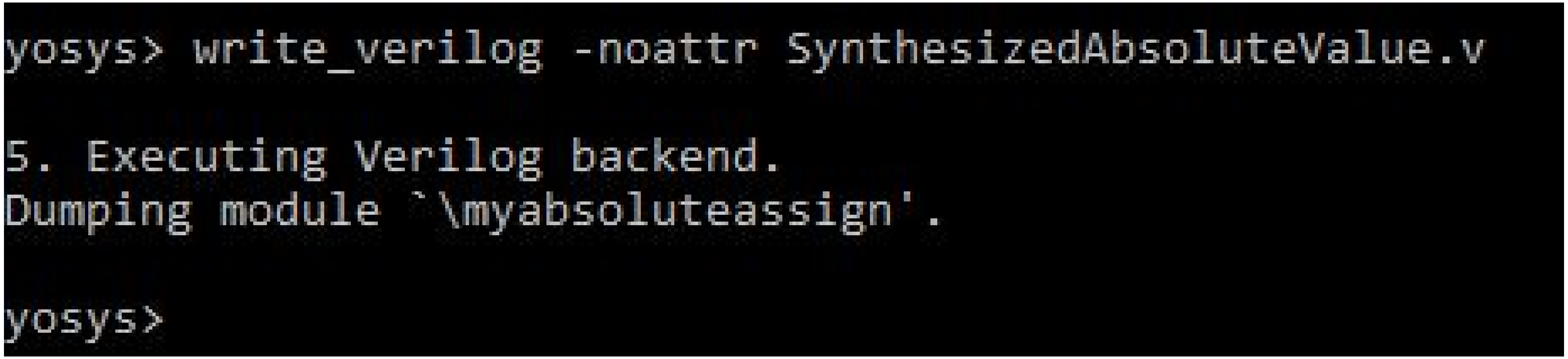
```
=== myabsoluteassign ===

   Number of wires:                 21
   Number of wire bits:             35
   Number of public wires:           3
   Number of public wire bits:      17
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                 26
     $_MUX_                          6
     $_NAND_                         4
     $_NOR_                          4
     $_NOT_                          2
     $_OR_                           4
     $_XNOR_                         3
     $_XOR_                          3

2.24. Executing CHECK pass (checking for obvious problems).
checking module myabsoluteassign..
found and reported 0 problems.
```



```
3. Executing DFFLIBMAP pass (mapping DFF cells to sequential cells from liberty file).
  cell DFF (noninv, pins=3, area=18.00) is a direct match for cell type $_DFF_P_.
  create mapping for $_DFF_N_ from mapping for $_DFF_P_.
  final dff cell mappings:
    DFF _DFF_N_ (.C(~C), .D( D), .Q( Q));
    DFF _DFF_P_ (.C( C), .D( D), .Q( Q));
    unmapped dff cell: $_DFF_NN0_
    unmapped dff cell: $_DFF_NN1_
    unmapped dff cell: $_DFF_NP0_
    unmapped dff cell: $_DFF_NP1_
    unmapped dff cell: $_DFF_PN0_
    unmapped dff cell: $_DFF_PN1_
    unmapped dff cell: $_DFF_PP0_
    unmapped dff cell: $_DFF_PP1_
    unmapped dff cell: $_DFFSR_NNN_
    unmapped dff cell: $_DFFSR_NNP_
    unmapped dff cell: $_DFFSR_NPN_
    unmapped dff cell: $_DFFSR_NPP_
    unmapped dff cell: $_DFFSR_PNN_
    unmapped dff cell: $_DFFSR_PNP_
    unmapped dff cell: $_DFFSR_PPN_
    unmapped dff cell: $_DFFSR_PPP_
Mapping DFF cells in module `\myabsoluteassign':
```



```
yosys> abc -liberty mycells.lib

4. Executing ABC pass (technology mapping using ABC).

4.1. Extracting gate netlist of module `\myabsoluteassign' to `<abc-temp-dir>/input.blif'..
Extracted 26 gates and 34 wires to a netlist network with 8 inputs and 8 outputs.

4.1.1. Executing ABC.
Running ABC command: <yosys-exe-dir>/yosys-abc -s -f <abc-temp-dir>/abc.script 2>&1
ABC: ABC command line: "source <abc-temp-dir>/abc.script".
ABC:
ABC: + read_blif <abc-temp-dir>/input.blif
ABC: + read_lib -w H:\Daneshgah\yosys\src/mycells.lib
ABC: Parsing finished successfully.  Parsing time =     0.00 sec
ABC: Warning: Templates are not defined.
ABC: Libery parser cannot read "time_unit".  Assuming   time_unit : "1ns".
ABC: Libery parser cannot read "capacitive_load_unit". Assuming   capacitive_load_unit(1, pf).
ABC: Scl_LibertyReadGenlib() skipped sequential cell "DFF".
ABC: Library "demo" from "H:\Daneshgah\yosys\src/mycells.lib" has 4 cells (1 skipped: 1 seq; 0 tri-state; 0 no func).  Time =     0.00 sec
ABC: Memory =    0.00 MB. Time =     0.00 sec
ABC: + strash
ABC: + dc2
ABC: + scorr
ABC: Warning: The network is combinational (run "fraig" or "fraig_sweep").
ABC: + ifraig
ABC: + retime -o
ABC: + strash
ABC: + dch -f
ABC: + map
ABC: + write_blif <abc-temp-dir>/output.blif

4.1.2. Re-integrating ABC results.
ABC RESULTS:               BUF cells:        1
ABC RESULTS:              NAND cells:       12
ABC RESULTS:               NOR cells:       25
ABC RESULTS:               NOT cells:       12
ABC RESULTS:        internal signals:       18
ABC RESULTS:           input signals:        8
ABC RESULTS:          output signals:        8
Removing temp directory.
```



```
yosys> write_verilog -noattr SynthesizedAbsoluteValue.v

5. Executing Verilog backend.
Dumping module `\myabsoluteassign'.

yosys>
```