Demo!

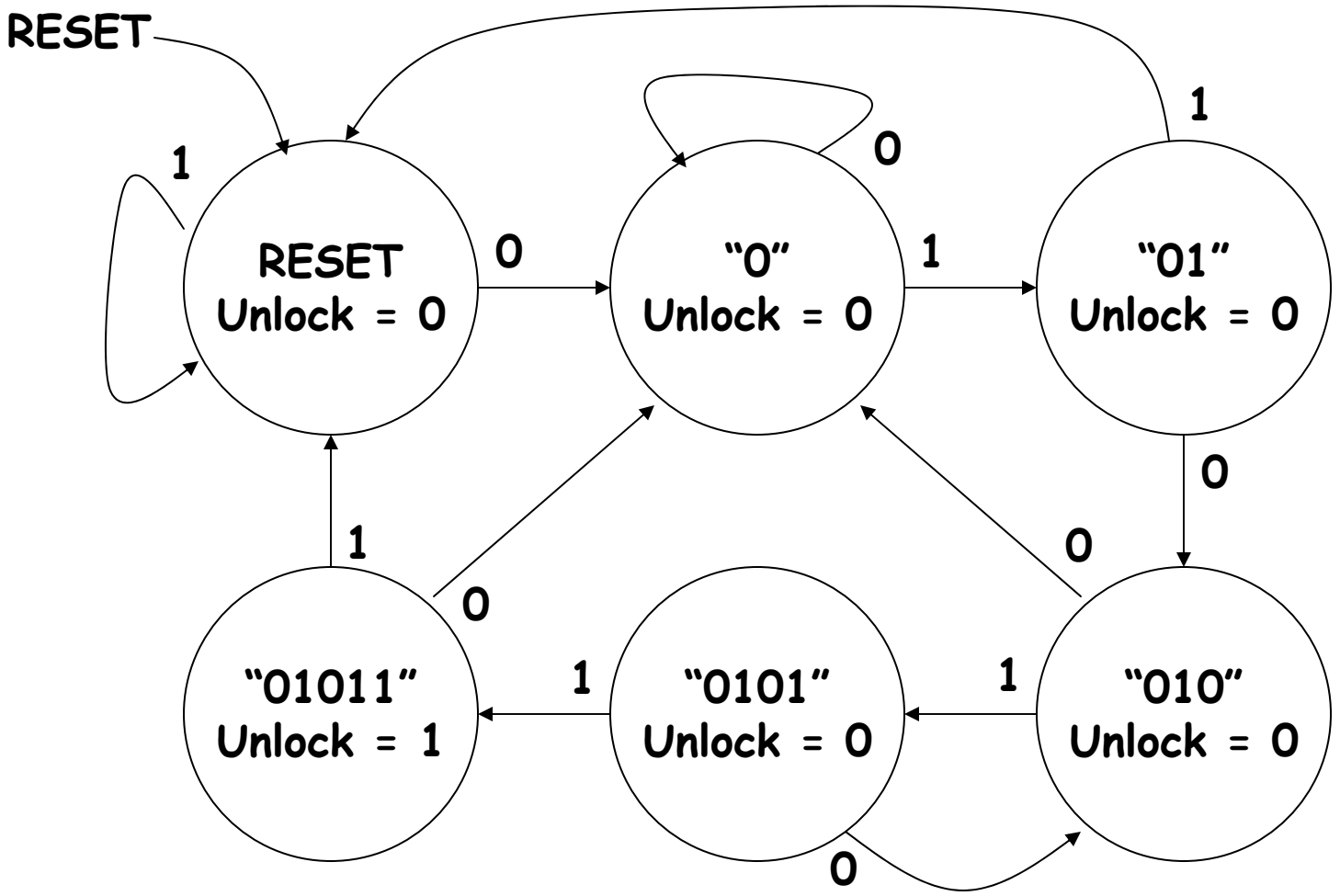
GOAL:

Build an electronic combination lock with a reset button, two number buttons (0 and 1), and an unlock output. The combination should be 01011.

STEPS:

1. Create state transition diagram for lock FSM
2. Write Verilog module(s) that implement FSM
3. Use MAX+plusII (synthesis, simulation)
4. Program FGPA, wire up buttons, give it a whirl!

Step 1: State transition diagram



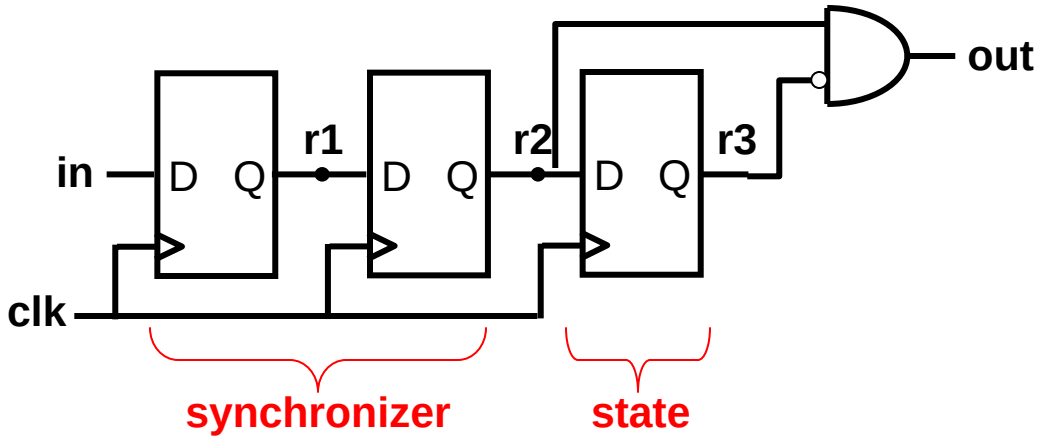
6 states → 3 bits

Step 2: Write Verilog

```
module lock(clk,reset_in,b0_in,b1_in,out);
  input clk,reset,b0_in,b1_in;
  output out;

  // synchronize push buttons, convert to pulses

  // implement state transition diagram
  reg [2:0] state;
  always @ (posedge clk)
  begin
    state <= ???;
  end

  // generate output
  assign out = ???;

  // debugging?
endmodule
```

Step 2A: synch buttons

```
// button -- push button synchronizer and level-to-pulse converter
// OUT goes high for one cycle of CLK whenever IN makes a
// low-to-high transition.
module button(clk,in,out);
  input clk;
  input in;
  output out;


  reg r1,r2,r3;
  always @ (posedge clk)
  begin
    r1 <= in;   // first reg in synchronizer
    r2 <= r1;   // second reg in synchronizer, output is in sync!
    r3 <= r2;   // remembers previous state of button
  end

  // rising edge = old value is 0, new value is 1
  assign out = ~r3 & r2;
endmodule
```



Step 2B: state transition diagram

```
// state assignments
parameter S_RESET = 0;
parameter S_0 = 1;
parameter S_01 = 2;
parameter S_010 = 3;
parameter S_0101 = 4;
parameter S_01011 = 5;

// implement state transition diagram
reg [2:0] state;
always @ (posedge clk)
begin
  if (reset) state <= S_RESET;
  else case (state)
    S_RESET: state <= b0 ? S_0   : b1 ? S_RESET : state;
    S_0:     state <= b0 ? S_0   : b1 ? S_01    : state;
    S_01:    state <= b0 ? S_010 : b1 ? S_RESET : state;
    S_010:   state <= b0 ? S_0   : b1 ? S_0101  : state;
    S_0101:  state <= b0 ? S_010 : b1 ? S_01011 : state;
    S_01011: state <= b0 ? S_0   : b1 ? S_RESET : state;
    default: state <= S_RESET;  // handle unused states
  endcase
end
```

Step 2C: generate output

```
// it's a Moore machine!  Output only depends on current state

assign out = (state == S_01011);
```

Step 2D: debugging?

```
// hmmm.  What would be useful to know?  Current state?

led_decoder l(state,segments);
```

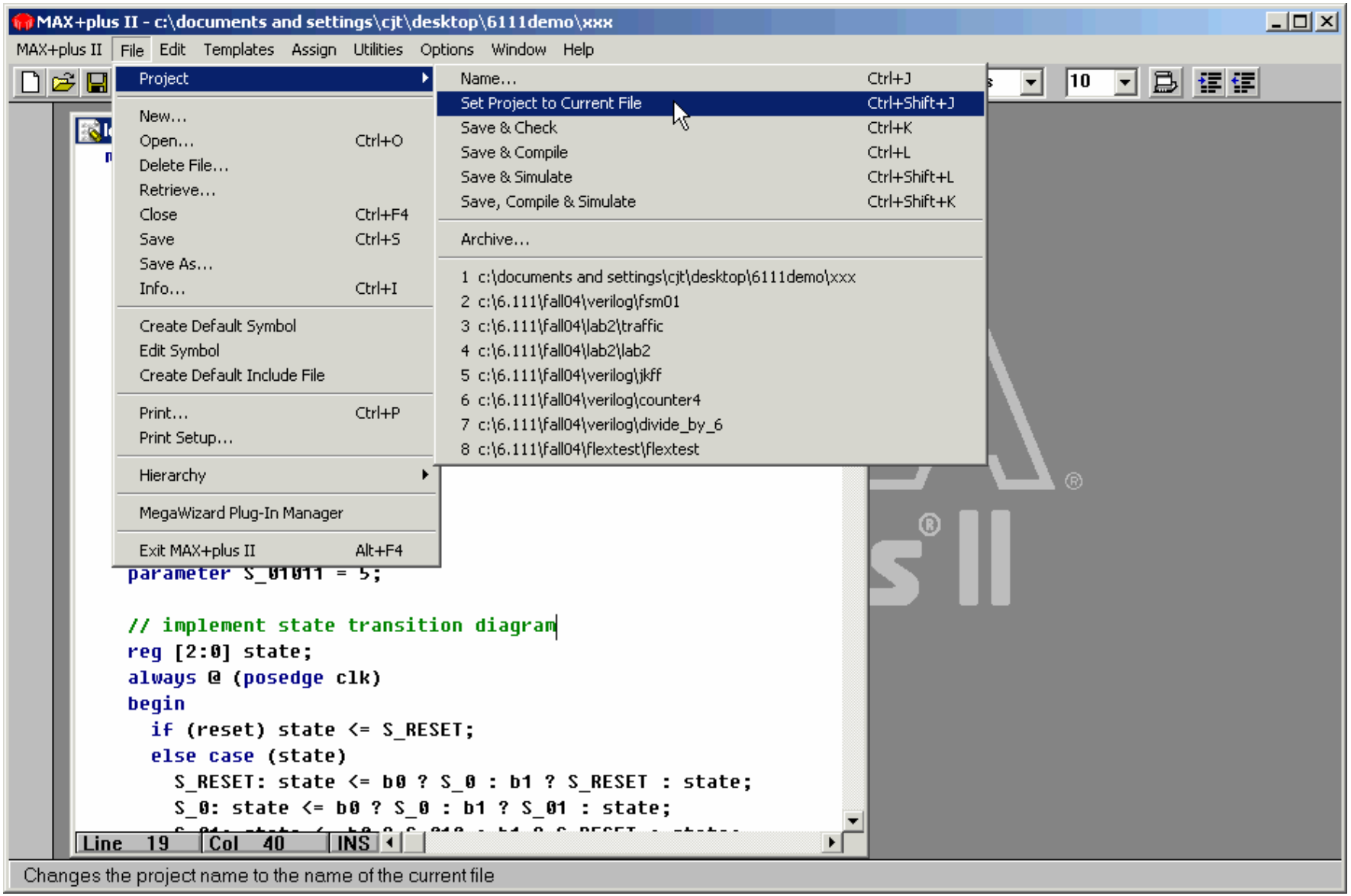
Step 2E: led_decoder

```
// led_decoder -- decode 4-bit input into 7-segment enables
module led_decoder(in,segments);
  input [3:0] in;
  output [6:0] segments;
  wire a,b,c,d,e,f,g;   // the seven segments

  // top segment
  assign a = (in==0) || (in==2) || (in==3) || (in==5) || (in==6) || (in==7) || (in==8) ||
             (in==9) || (in==10) || (in==14) ||(in==15);
  // upper right segment
  assign b = (in==0) || (in==1) || (in==2) || (in==3) || (in==4) || (in==7) || (in==8) ||
             (in==9) || (in==10) || (in==13);
  // lower right segment
  assign c = (in==0) || (in==1) || (in==3) || (in==4) || (in==5) || (in==6) || (in==7) ||
             (in==8) || (in==9) || (in==10) || (in==11) || (in==13);
  // bottom segment
  assign d = (in==0) || (in==2) || (in==3) || (in==5) || (in==6) || (in==8) || (in==11) ||
             (in==12) || (in==13) || (in==14);
  // bottom left segment
  assign e = (in==0) || (in==2) || (in==6) || (in==8) || (in==10) || (in==11) || (in==12) ||
             (in==13) || (in==14) || (in==15);
  // upper left segment
  assign f = (in==0) || (in==4) || (in==5) || (in==6) || (in==8) || (in==9) || (in==10) ||
             (in==11) || (in==14) || (in==15);
  // middle segment
  assign g = (in==2) || (in==3) || (in==4) || (in==5) || (in==6) || (in==8) || (in==9) ||
             (in==10) || (in==11) || (in==12) || (in==13) || (in==14) || (in==15);
  // bundle results into a single 7-bit vector
  assign segments = {g,f,e,d,c,b,a};
endmodule
```

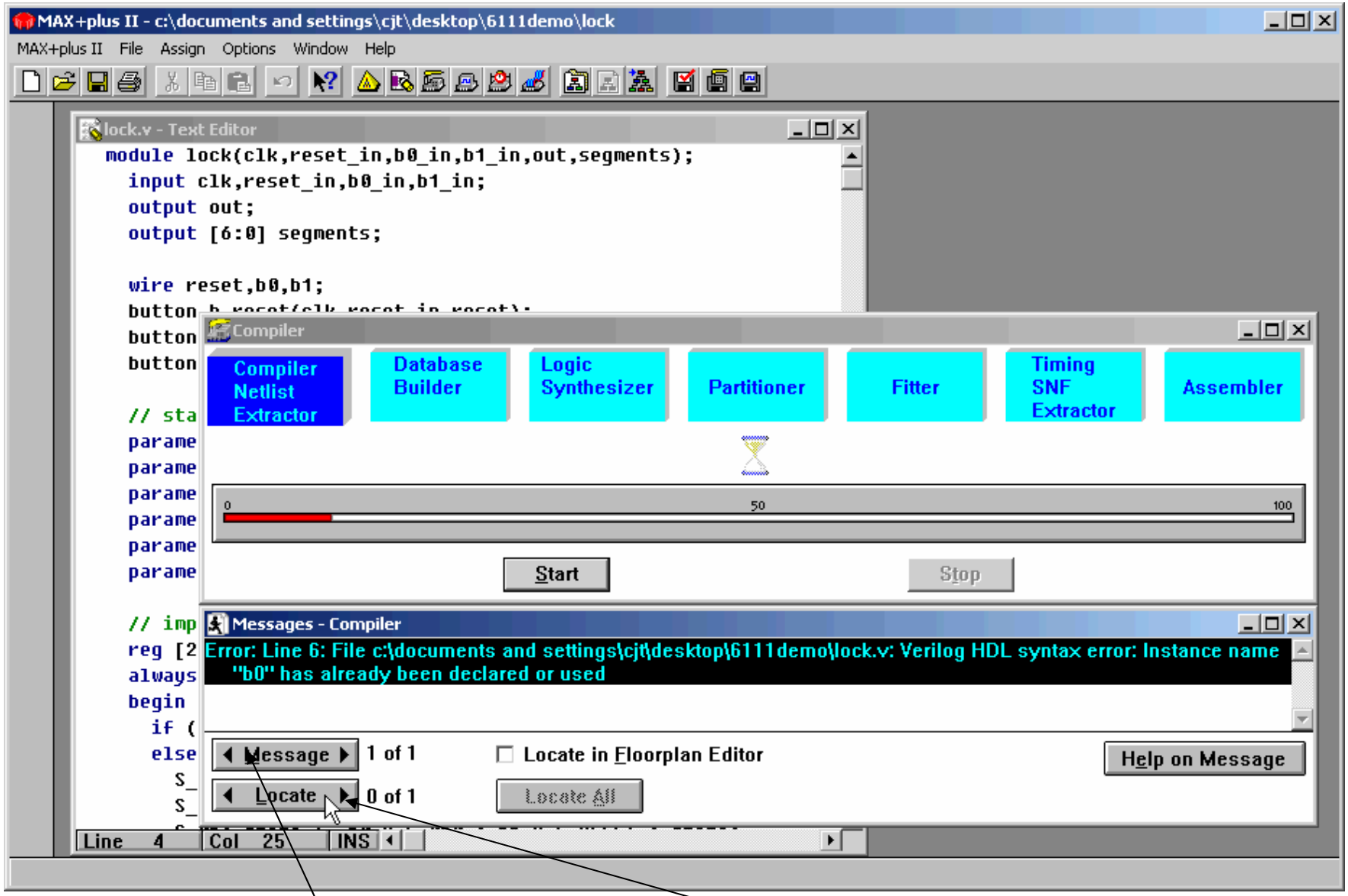
Step 2: final Verilog implementation

```
module lock(clk,reset_in,b0_in,b1_in,out,segments);
  input clk,reset_in,b0_in,b1_in;
  output out;
  output [6:0] segments;

  wire reset,b0,b1;
  button b_reset(clk,reset_in,reset);
  button b_0(clk,b0_in,b0);
  button b_1(clk,b1_in,b1);

  // state assignments
  parameter S_RESET = 0;  parameter S_0 = 1; parameter S_01 = 2;
  parameter S_010 = 3; parameter S_0101 = 4;  parameter S_01011 = 5;

  // implement state transition diagram
  reg [2:0] state;
  always @ (posedge clk) begin
    if (reset) state <= S_RESET;
    else case (state)
      S_RESET: state <= b0 ? S_0 : b1 ? S_RESET : state;
      S_0: state <= b0 ? S_0 : b1 ? S_01 : state;
      S_01: state <= b0 ? S_010 : b1 ? S_RESET : state;
      S_010: state <= b0 ? S_0 : b1 ? S_0101 : state;
      S_0101: state <= b0 ? S_010 : b1 ? S_01011 : state;
      S_01011: state <= b0 ? S_0 : b1 ? S_RESET : state;
      default: state <= S_RESET;  // handle unused states
    endcase
  end

  assign out = (state == S_01011);
  led_decoder l(state,segments);  // debugging!
endmodule
```

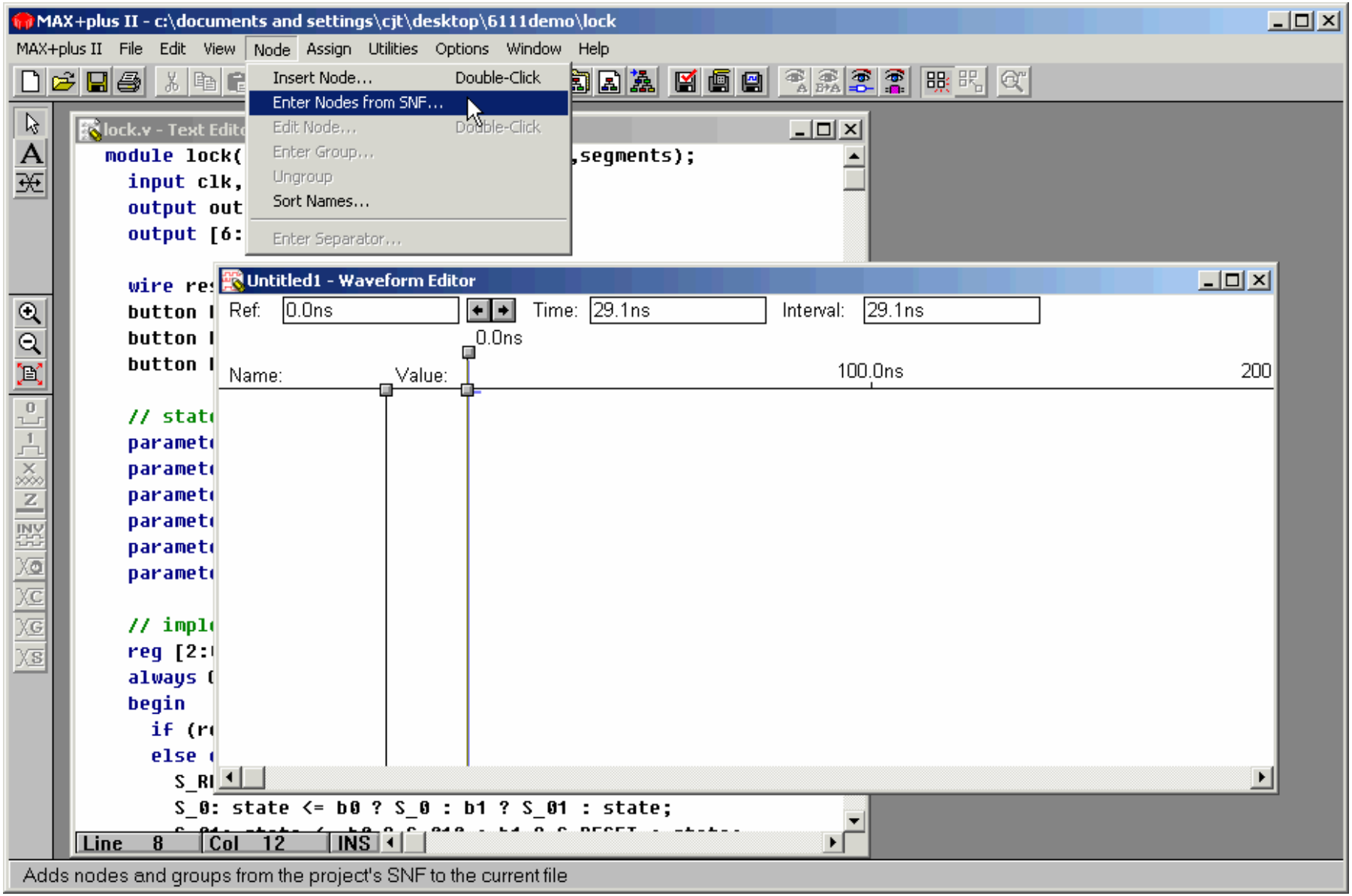
Step 3A: set top level file



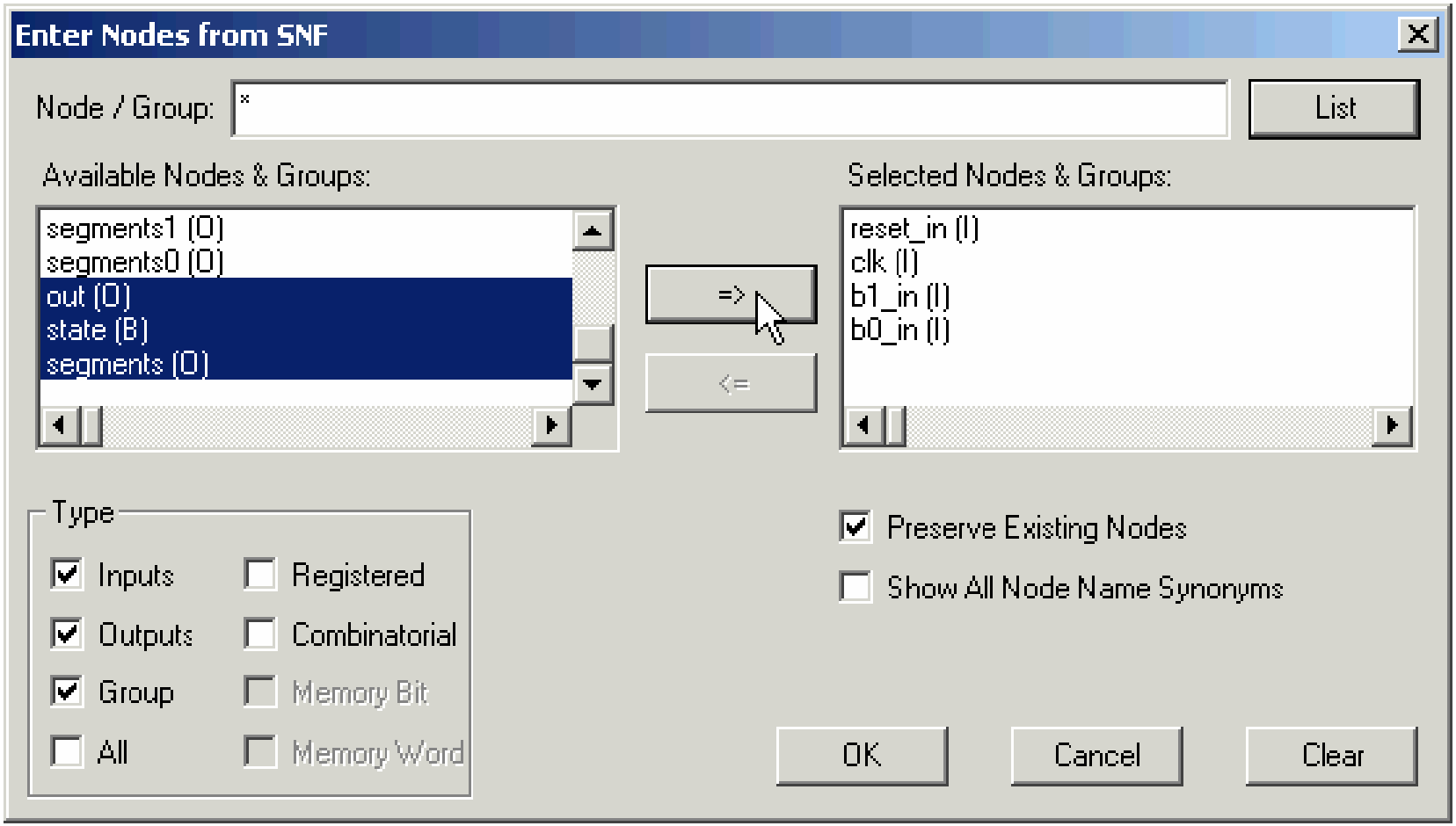
Step 3B: compile/synthesize/map



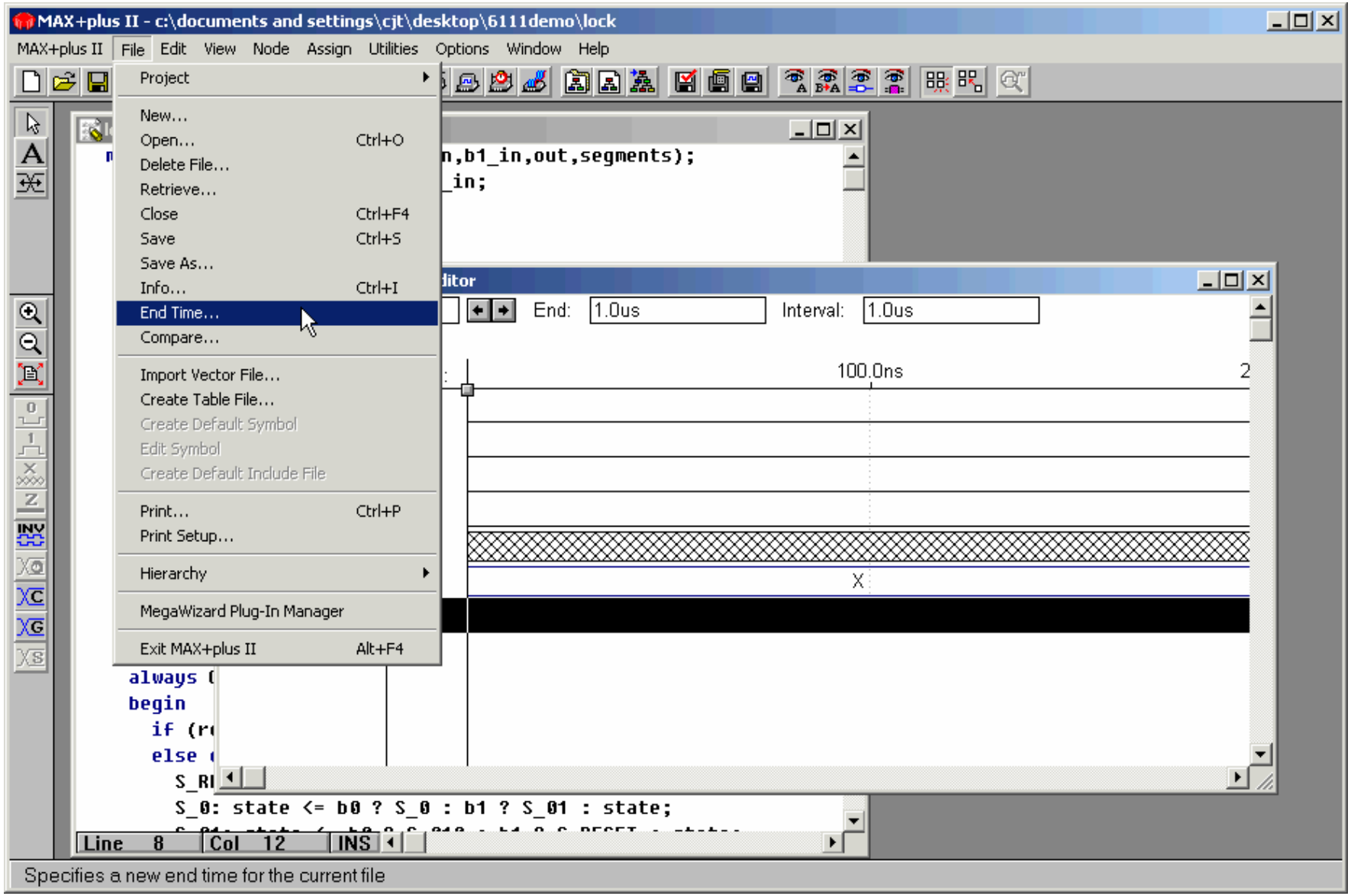
Step 3C: start waveform editor



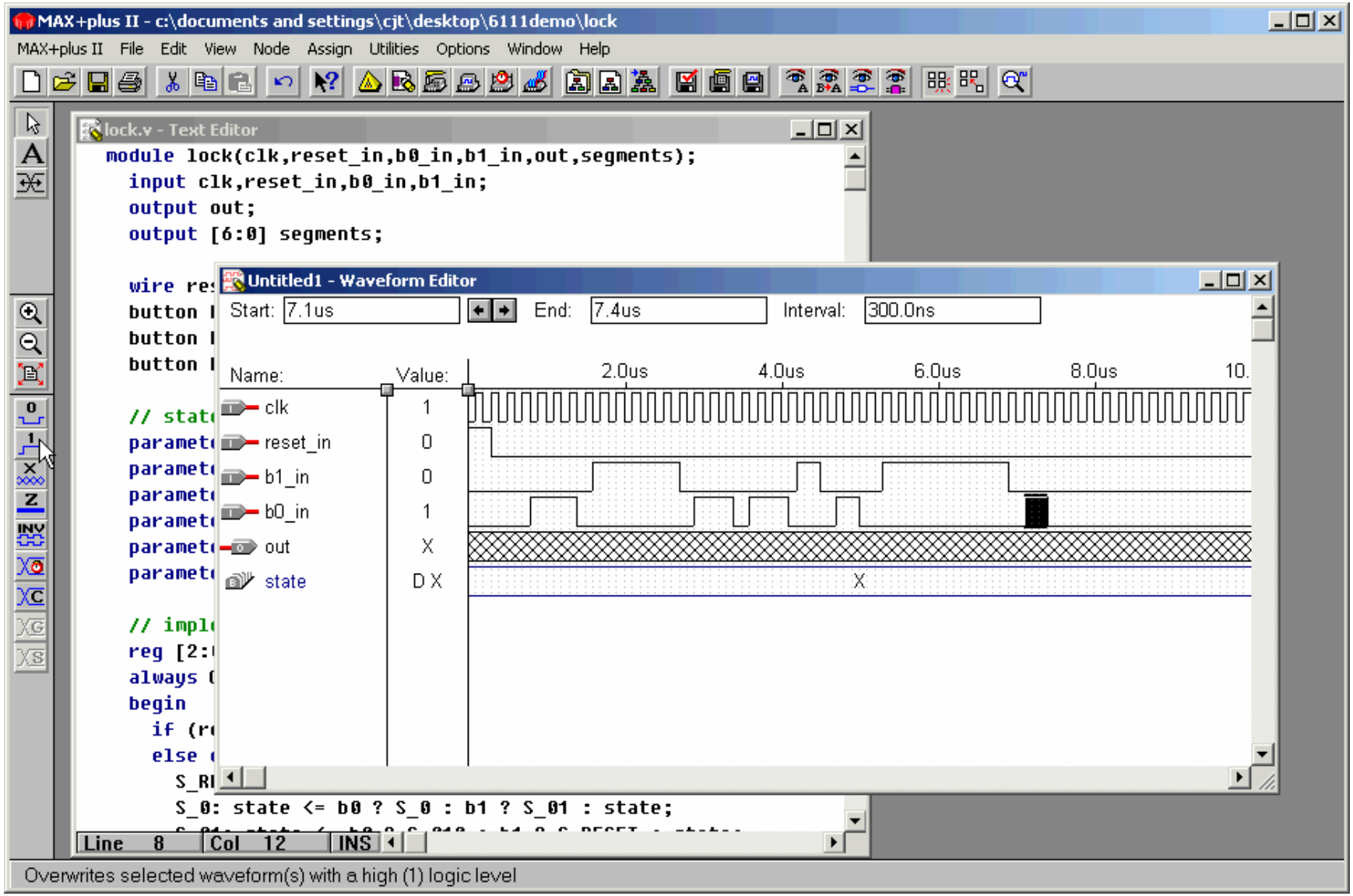
Step 3D: add I/O nodes



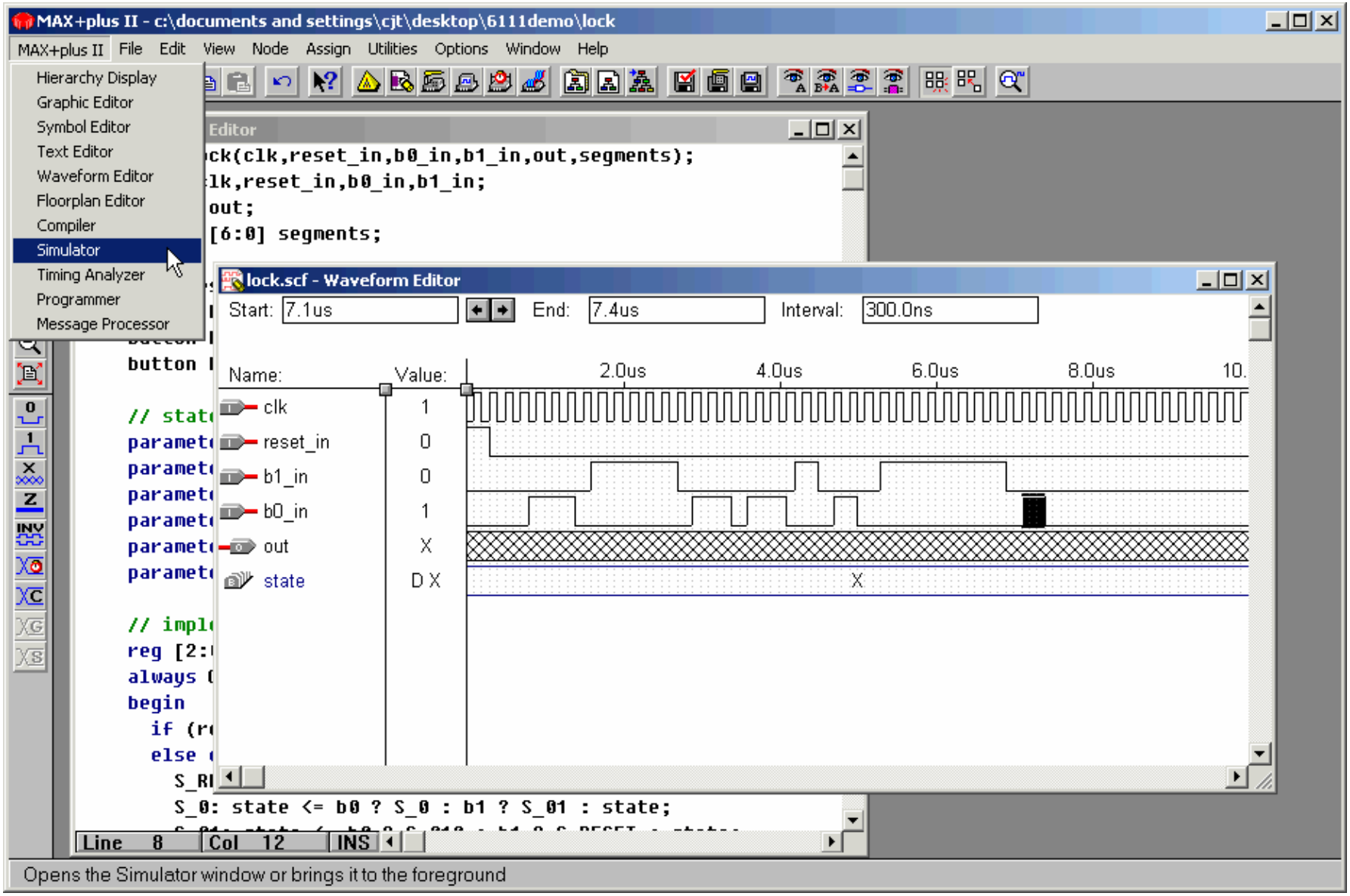
Step 3E: set simulation end time



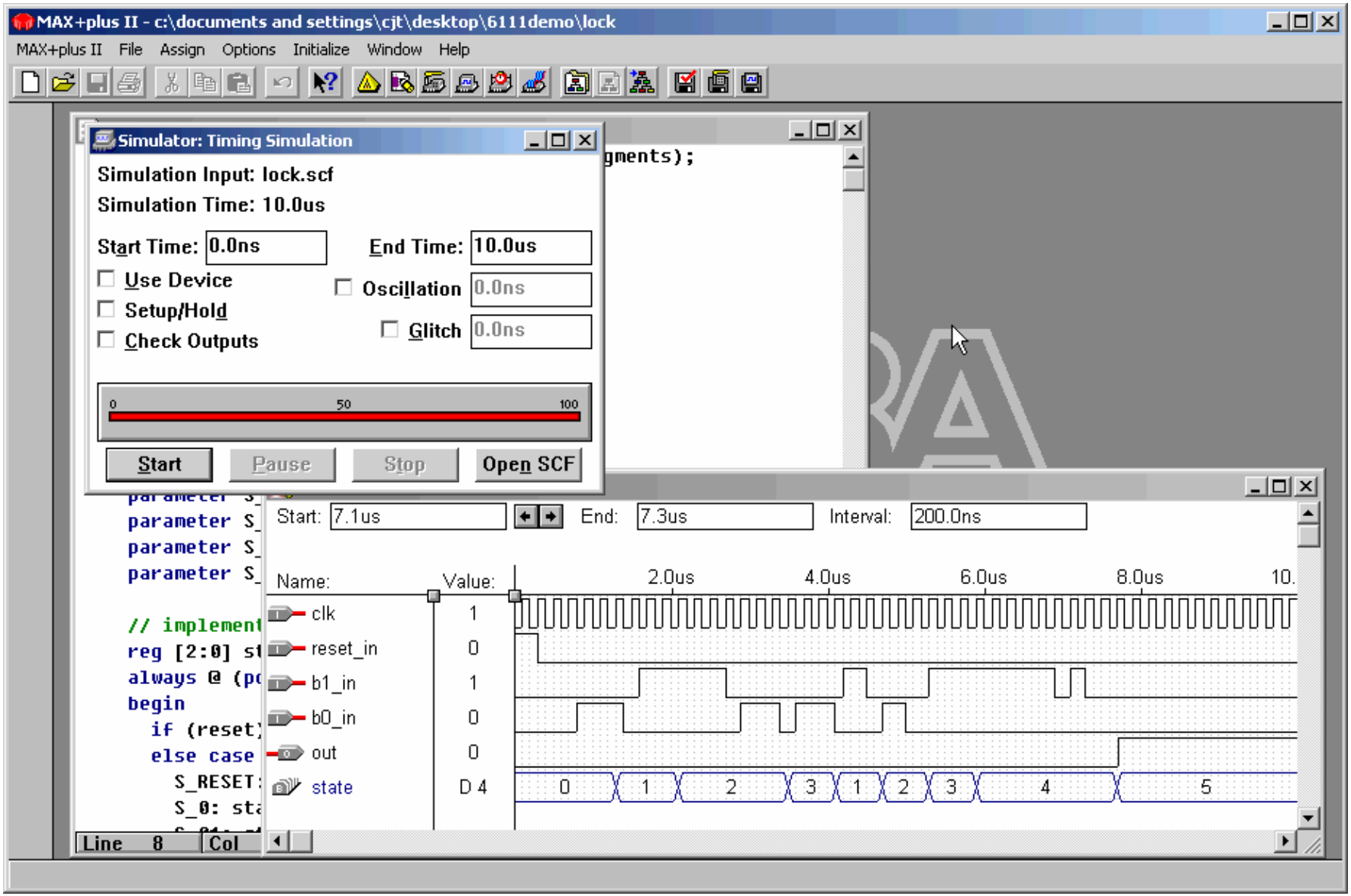
Step 3F: set input values, save



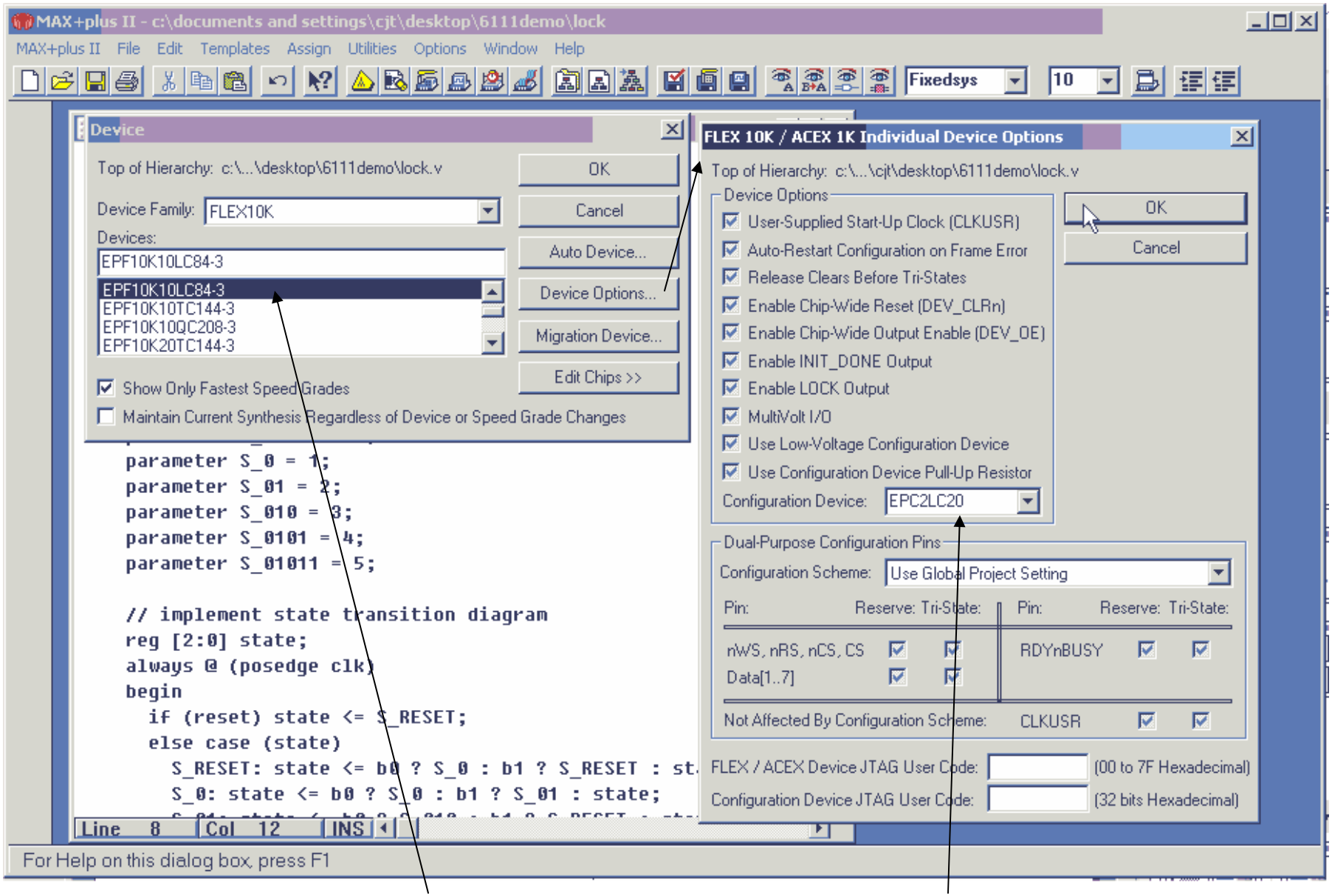
Step 3G: run simulation



Step 3H: examine simulation results



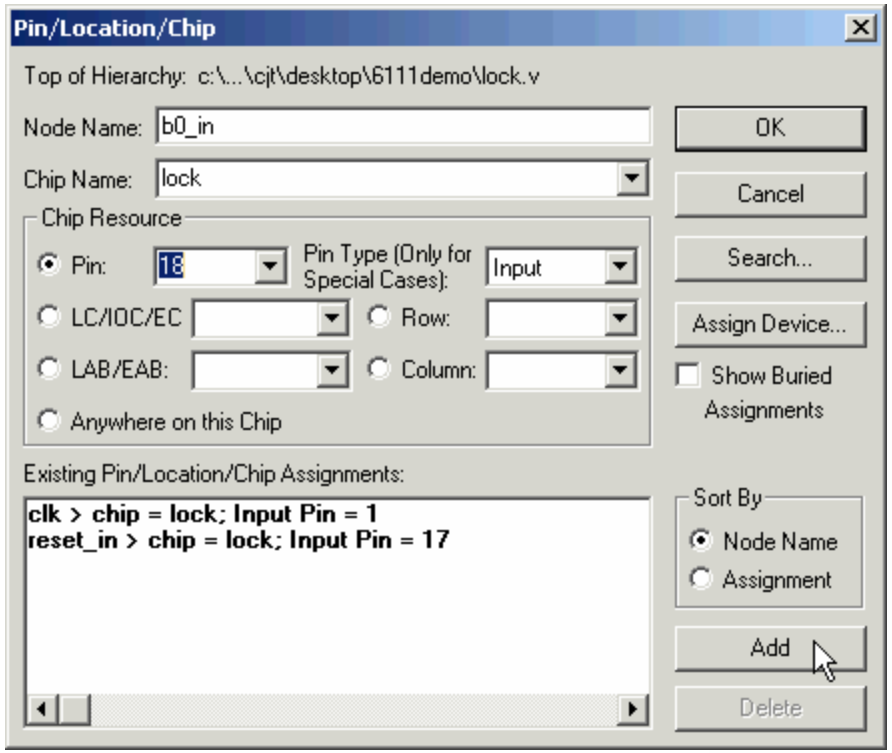
Step 4A: Assign→Device…



EPF10K10LC84 or EPF10K70RC240

EPC2LC20

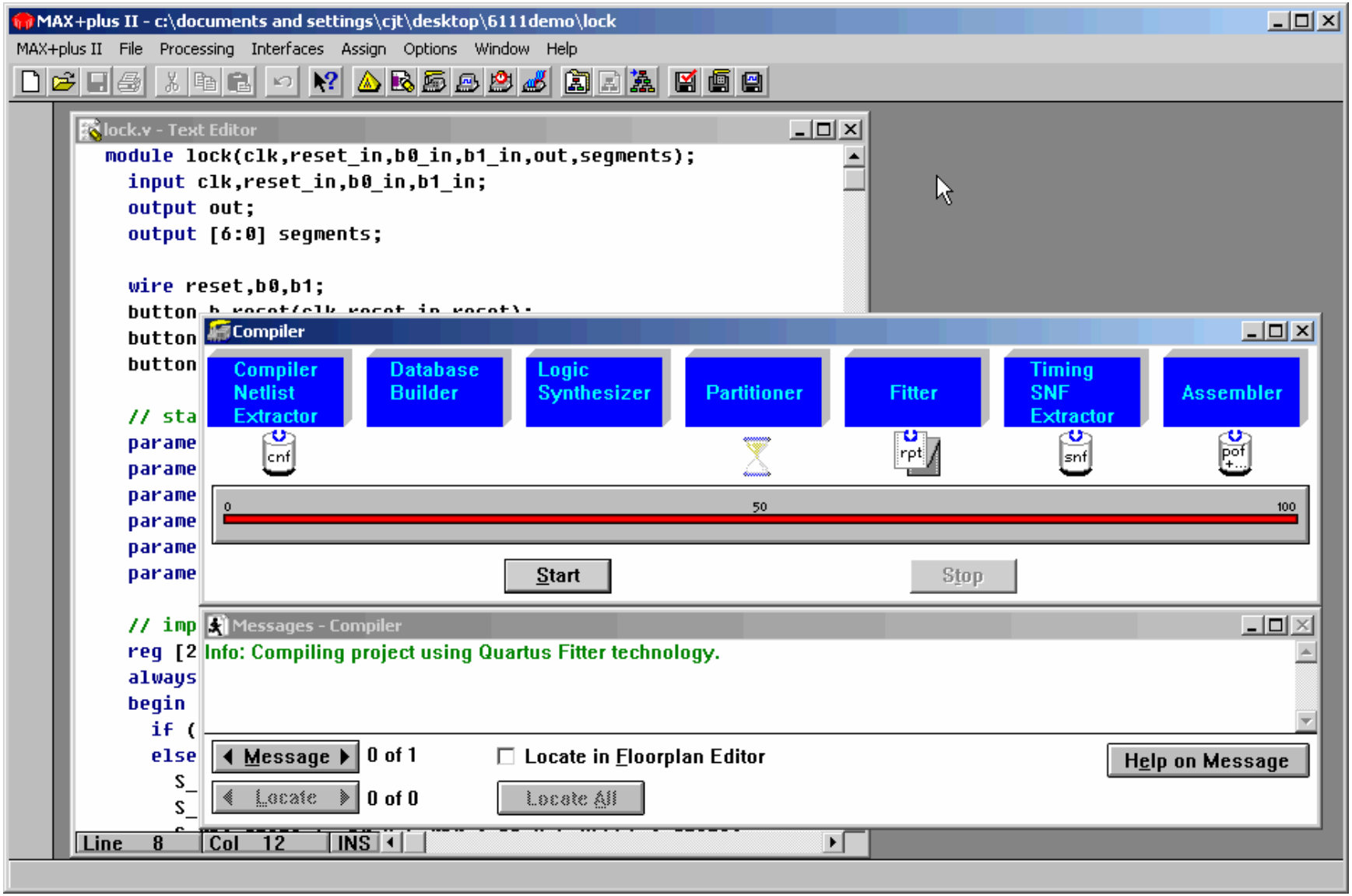
Step 4B: Assign→Pins…



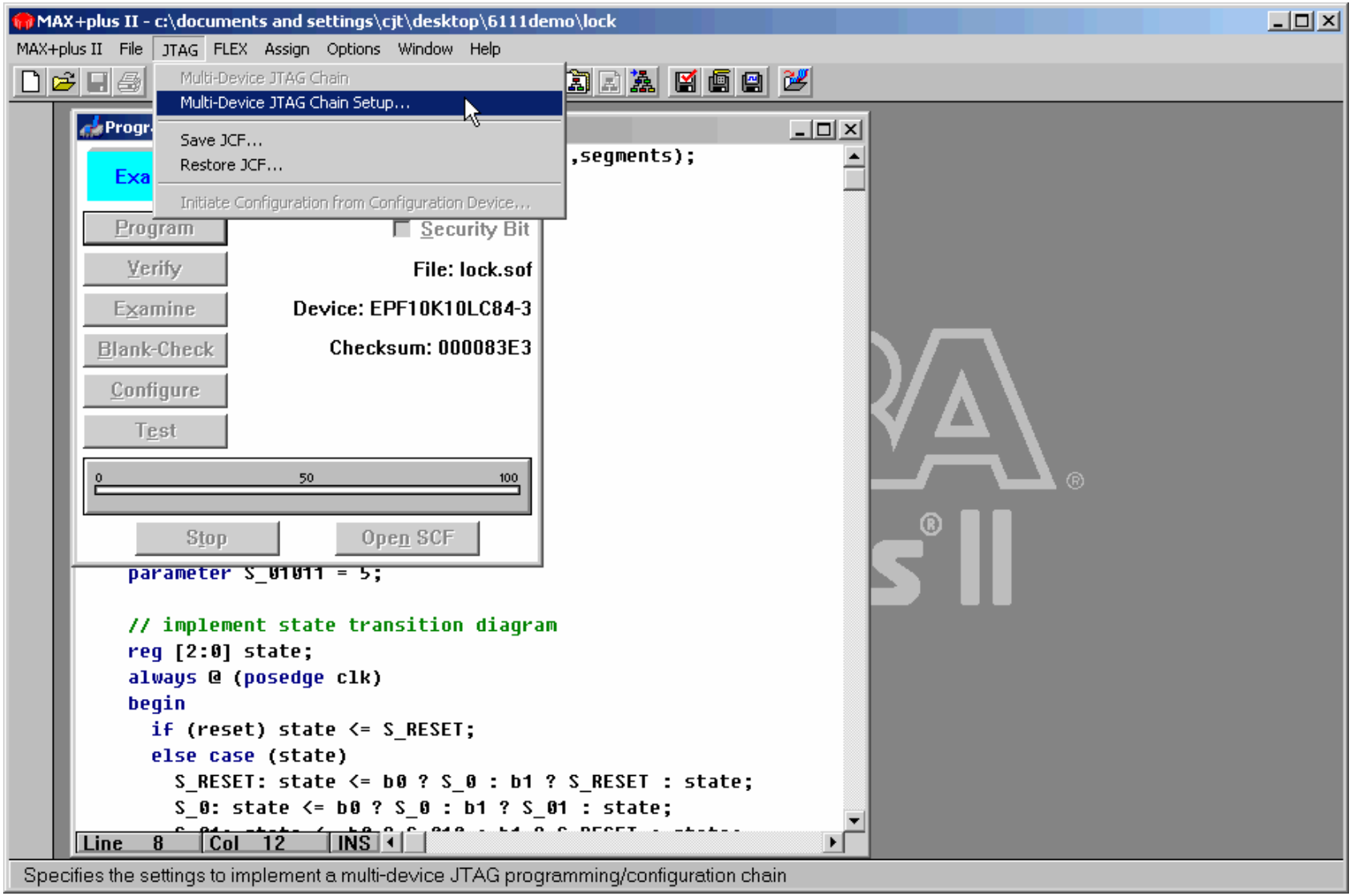
Correlation between NuBus Pins and FLEX 10K Pins

NuBus Address	FLEX 10K10 Pin Number	FLEX 10K70 Pin Number
AD0	1 (CLK)	91 (CLK)
AD1	NONE	NONE
AD2	17	7
AD3	18	8
AD4	19	9
AD5	21	12
AD6	22	13
AD7	23	14
AD8	24	15
AD9	25	17
AD10	27	18
AD11	28	19
AD12	29	20
AD13	30	21

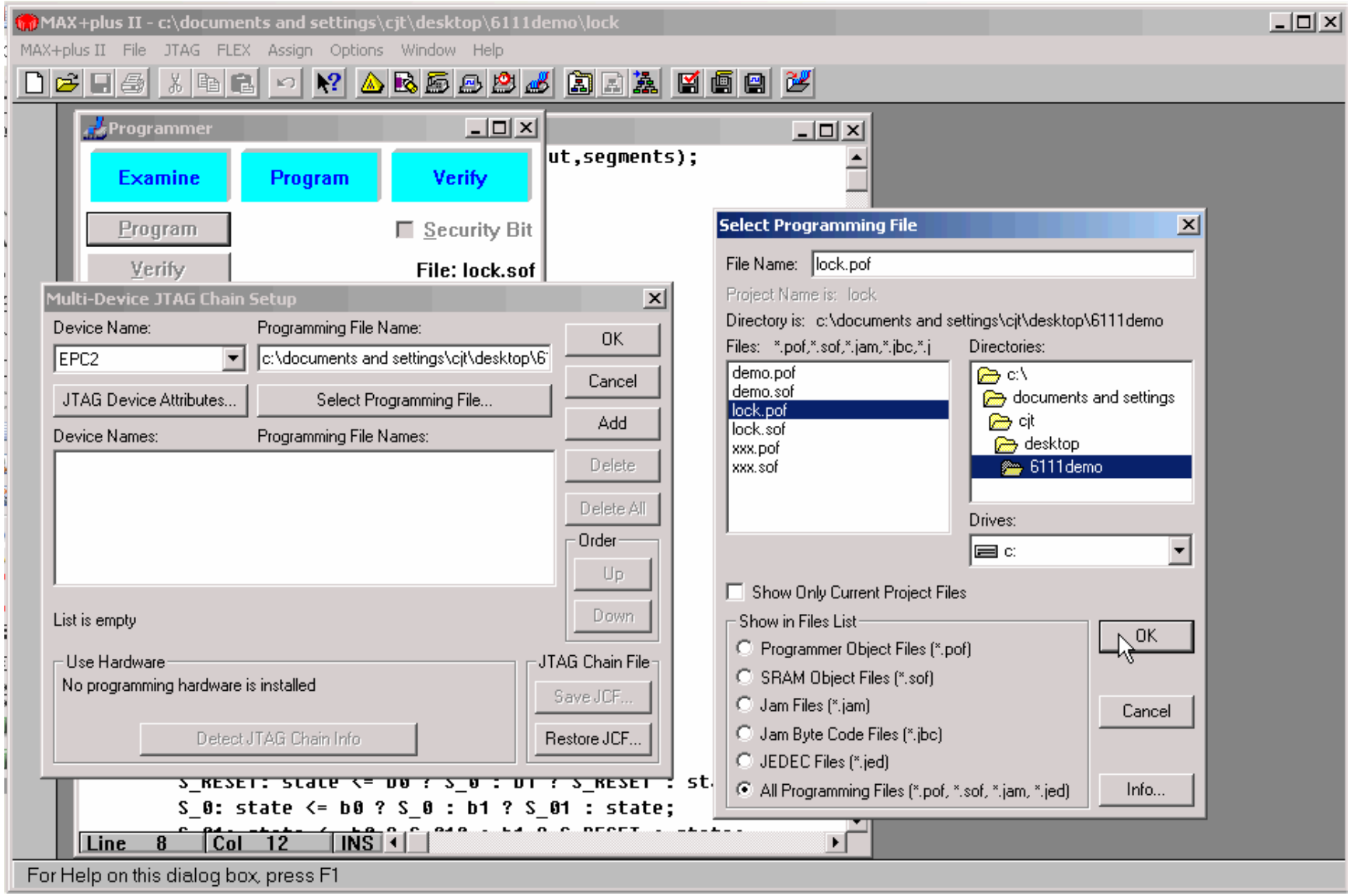
Step 4C: recompile



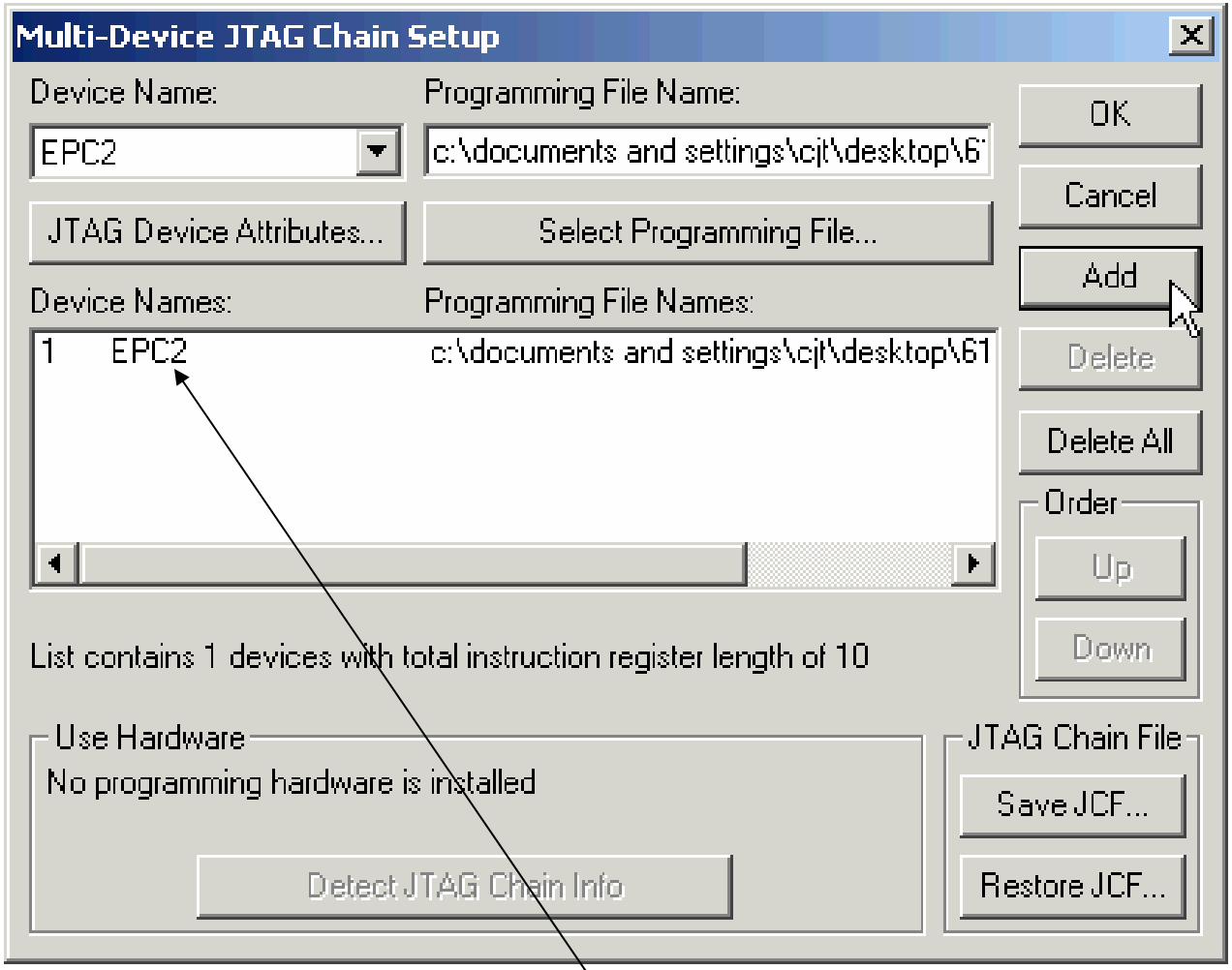
Step 4D: MAX+plusII→Programmer
Set up JTAG chain



Step 4E: select programming file

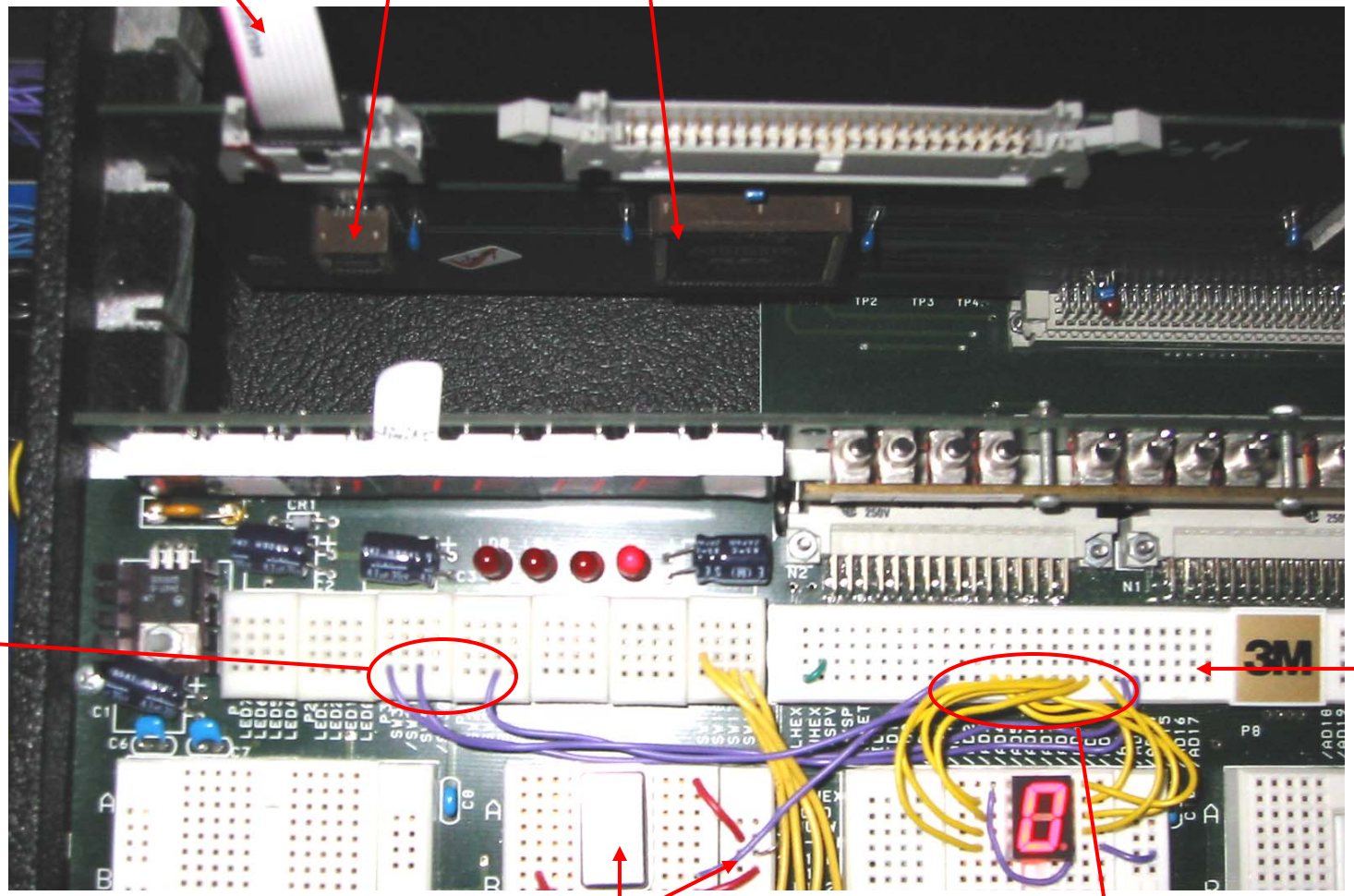


Step 4F: Add to list



We program the EPC2 serial flash memory whose contents is loaded into the Flex10K FPGA on power-up.

Step 4G: Attach ByteBlasterII cable



Step 4H: program device, cycle power