

实验1

简单计算机系统基本模块设计A

主要内容

- 1、简单计算机系统实验任务 **简介**
- 2、模块设计： **数字电路知识**
ROM、寄存器组、ALU
- 3、动手练习： 仿真验证模块功能

实验任务简介

■ 实现一种简单计算机系统的设计

- ✓ 精简的MIPS指令集.
- ✓ EDA仿真.

■ 编写程序，仿真验证所设计系统的功能

- ✓ 用汇编格式编写程序，并翻译成机器码.
- ✓ 将机器码程序放入ROM，通过仿真验证简单计算机系统的功能.

实验目的

- 学习、掌握计算机部件模块电路设计方法
 - 熟悉计算机的组成与各部件的功能
 - 熟悉简单计算机的指令集和数据通路
- 掌握编写汇编语言程序和机器码程序.
- 设计一个简单计算机系统

更好地理解 and 掌握计算机的组成与原理

简单计算机系统指令集

操作名称	操作码	汇编语言格式指令	执行操作
与	0000	AND Rd, Rs, Rt	$Rd \leftarrow Rs \text{ and } Rt$; $PC \leftarrow PC + 1$
或	0001	OR Rd, Rs, Rt	$Rd \leftarrow Rs \text{ or } Rt$; $PC \leftarrow PC + 1$
不带进位加	0010	ADD Rd, Rs, Rt	$Rd \leftarrow Rs + Rt$; $PC \leftarrow PC + 1$
不带借位减	0011	SUB Rd, Rs, Rt	$Rd \leftarrow Rs - Rt$; $PC \leftarrow PC + 1$
无符号数比较	0100	SLT Rd, Rs, Rt	If $Rs < Rt$, $Rd = 1$ else $Rd = 0$; $PC \leftarrow PC + 1$
带借位减	0101	SUBC Rd, Rs, Rt	$Rd \leftarrow Rs - Rt - (1 - C)$; $PC \leftarrow PC + 1$
带进位加	0110	ADDC Rd, Rs, Rt	$Rd \leftarrow Rs + Rt + C$; $PC \leftarrow PC + 1$
立即数与	1000	ANDI Rt, Rs, imm	$Rt \leftarrow Rs \text{ and } imm$; $PC \leftarrow PC + 1$
立即数或	1001	ORI Rt, Rs, imm	$Rt \leftarrow Rs \text{ or } imm$; $PC \leftarrow PC + 1$
立即数加	1010	ADDI Rt, Rs, imm	$Rt \leftarrow Rs + imm$; $PC \leftarrow PC + 1$
读存储器	1011	LW Rt, Rs, imm	$Rt \leftarrow \text{MEM}[Rs + imm]$; $PC \leftarrow PC + 1$
写存储器	1100	SW Rt, Rs, imm	$\text{MEM}[Rs + imm] \leftarrow Rt$; $PC \leftarrow PC + 1$
无条件跳转	0111	JMP imm	$PC \leftarrow imm$
相等时跳转	1101	BEQ Rs, Rt, imm	If $Rt = Rs$, $PC \leftarrow PC + imm + 1$ else $PC \leftarrow PC + 1$
不等时跳转	1110	BNE Rs, Rt, imm	If $Rt \neq Rs$, $PC \leftarrow PC + imm + 1$ else $PC \leftarrow PC + 1$

R型指令二进制编码结构

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				Rs		Rt		Rd		---					

- Op: 指令操作码;
- Rs: 第1个源操作数的寄存器号;
- Rt: 第2个源操作数的寄存器号;
- Rd: 目的操作数的寄存器号;
- : 表示任意值, 编码时通常取0

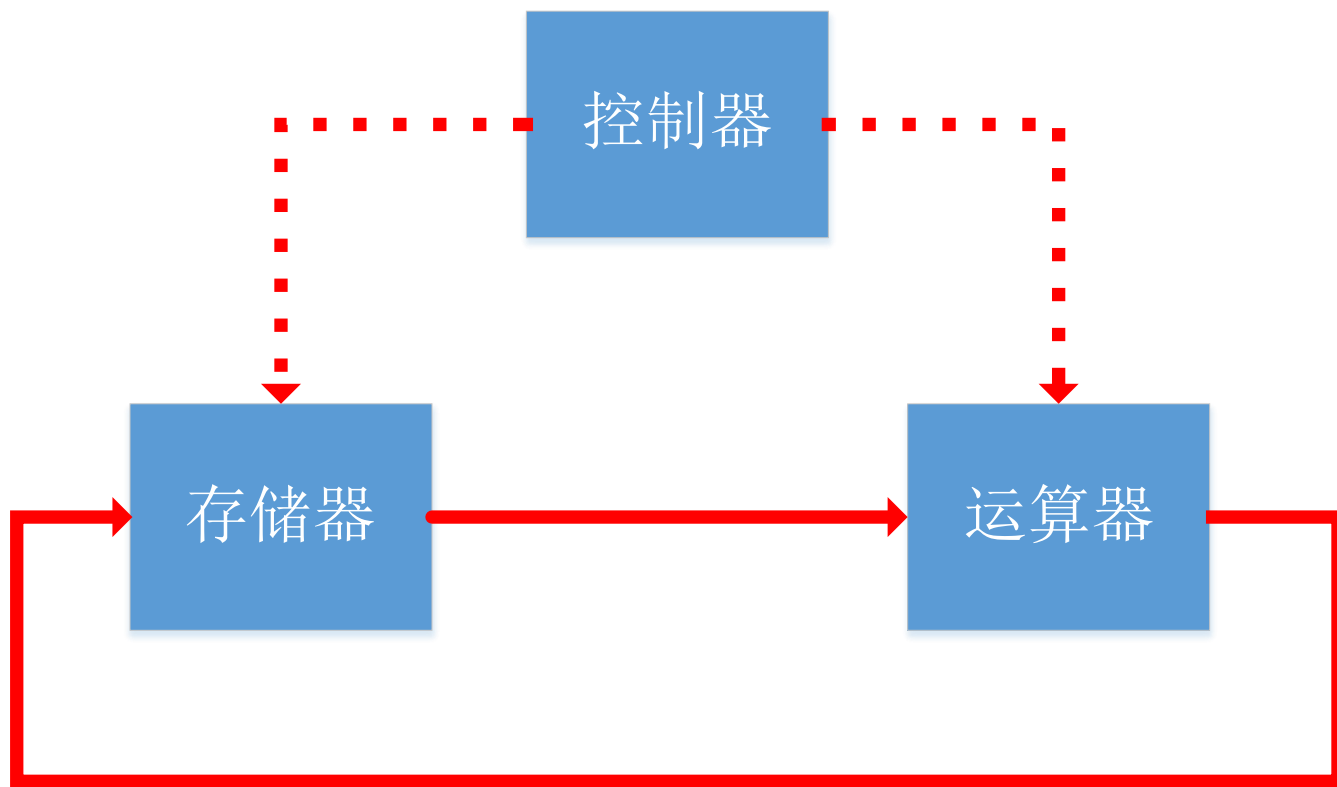
操作码 操作数

OR Rd, Rs, Rt **OR R3, R1, R2** Rs=01, Rt=10, Rd=11

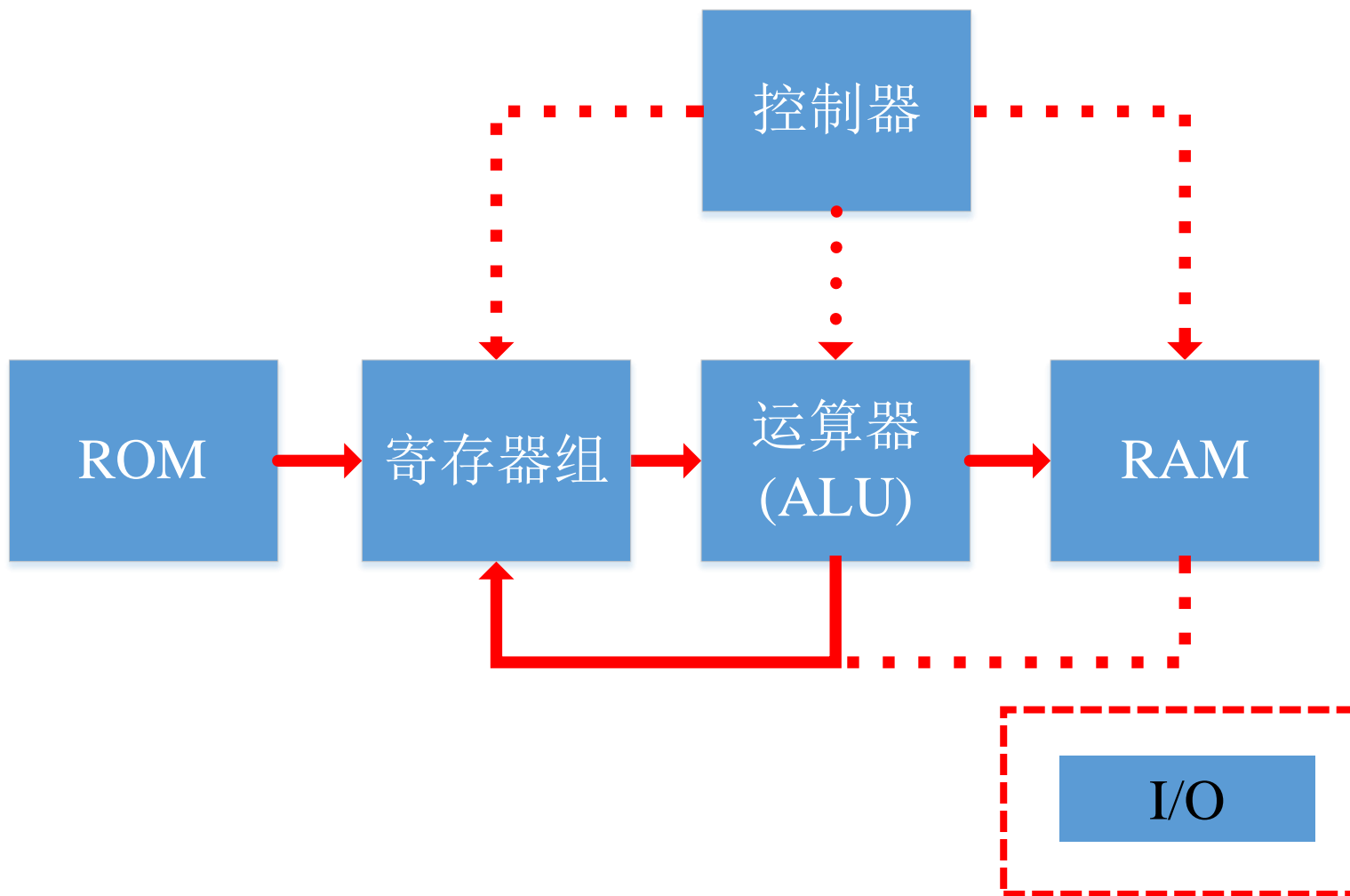
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				Rs		Rt		Rd		-----					
0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0

0001 0110 1100 0000B, 以十六进制形式表示为0x16C0

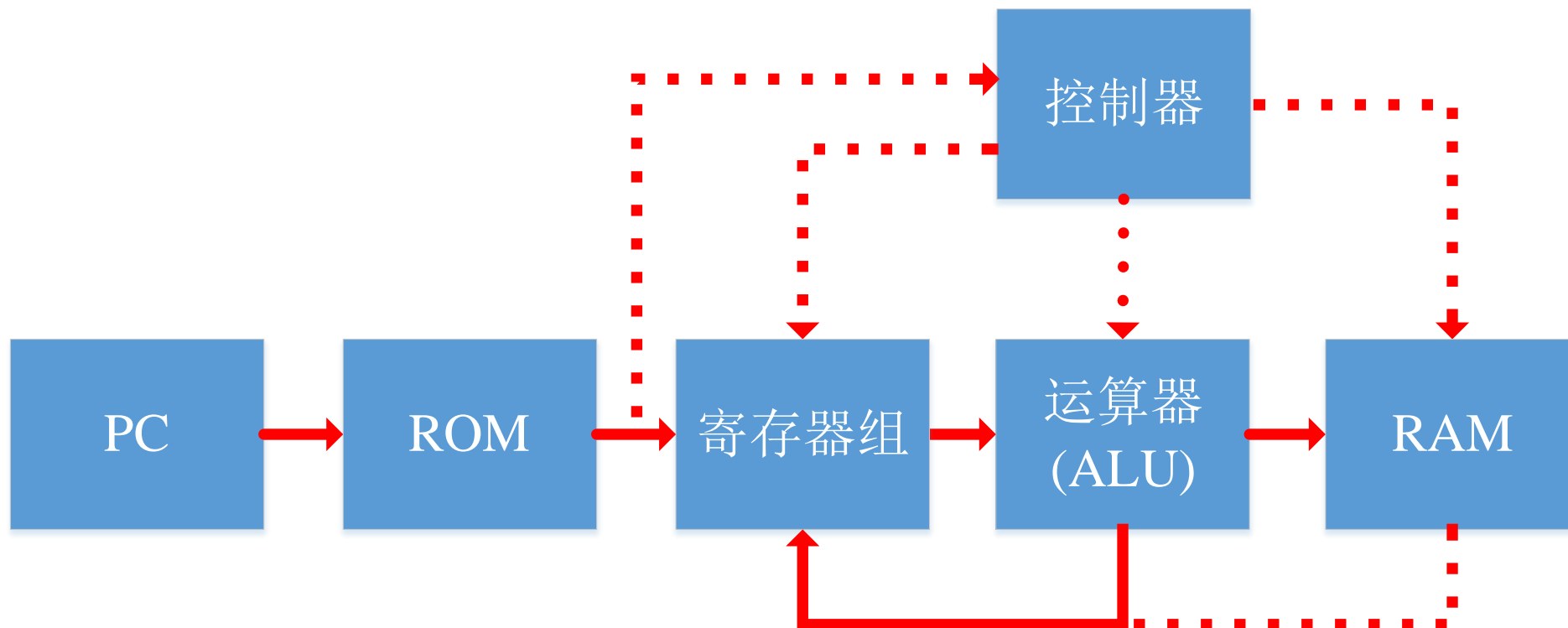
计算机模型



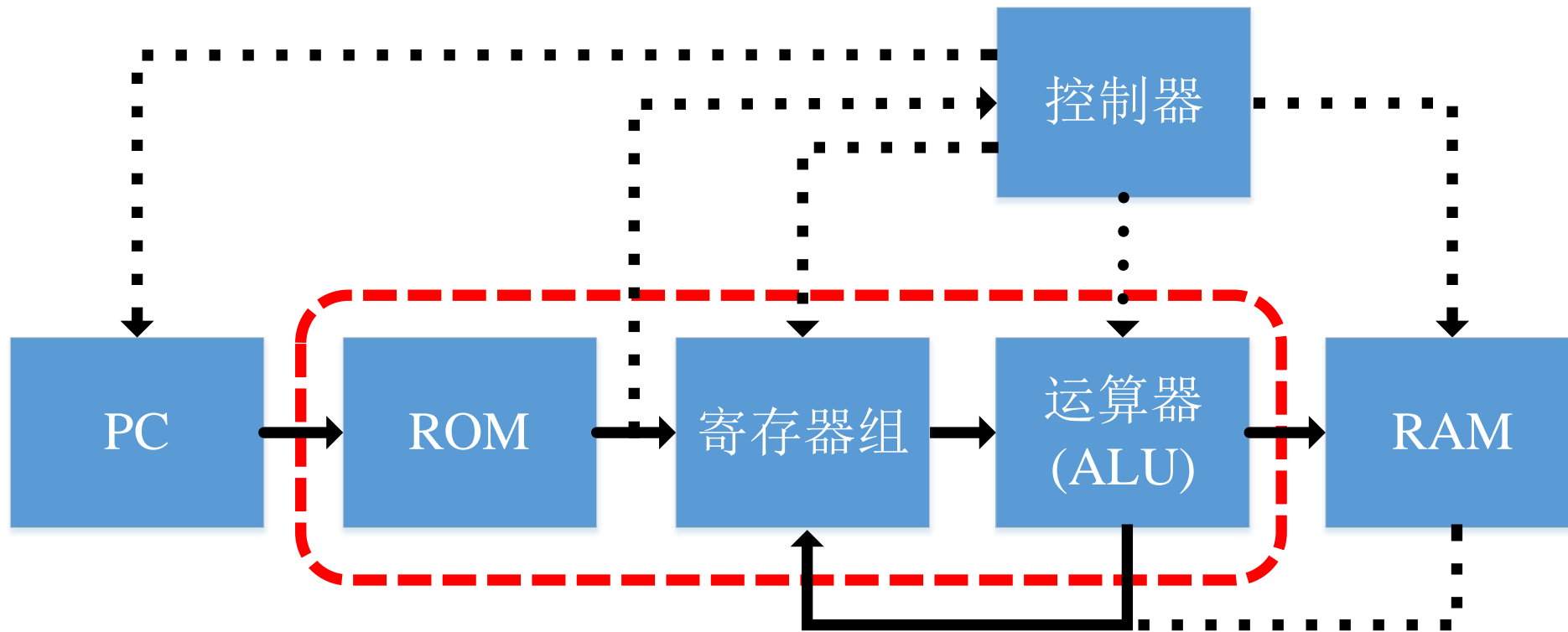
计算机模型



计算机模型



计算机模型



1、ROM模块设计

采用Quartus Prime

ROM模块

■ 输入信号：

- ✓ address[7..0]: ROM单元的地址
- ✓ clock: 时钟，在时钟的上升沿，将address指定的ROM单元内容输出

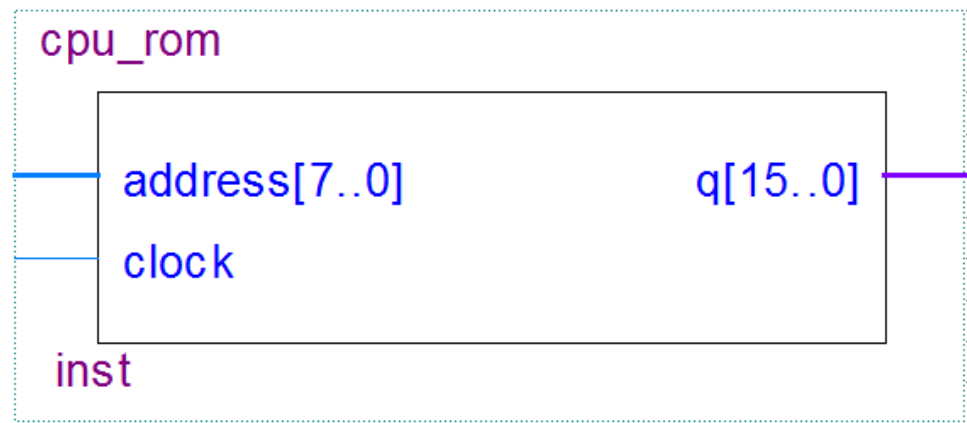
■ 输出信号：

- ✓ q[15..0]: 输出address指定的ROM单元内容

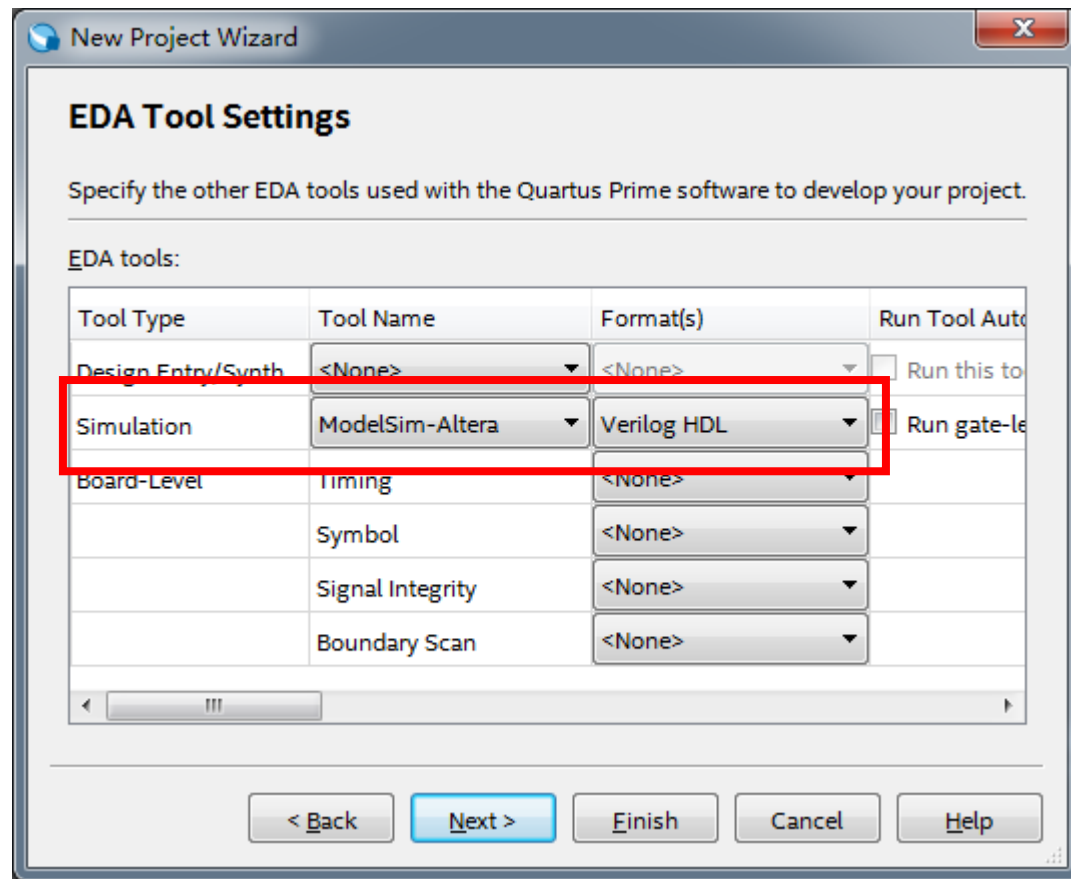
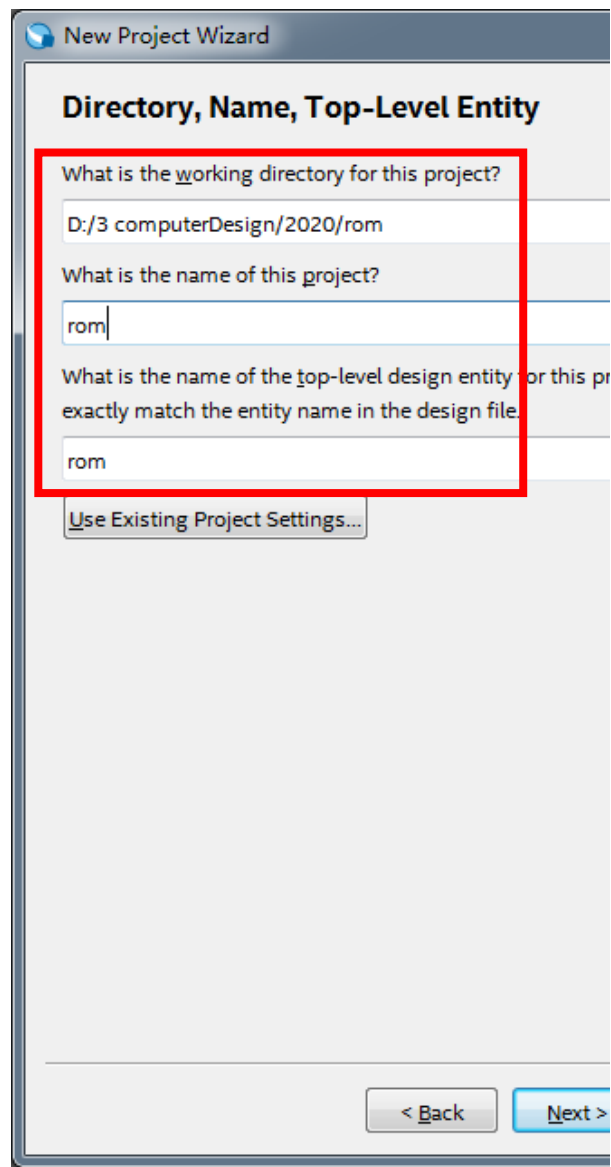
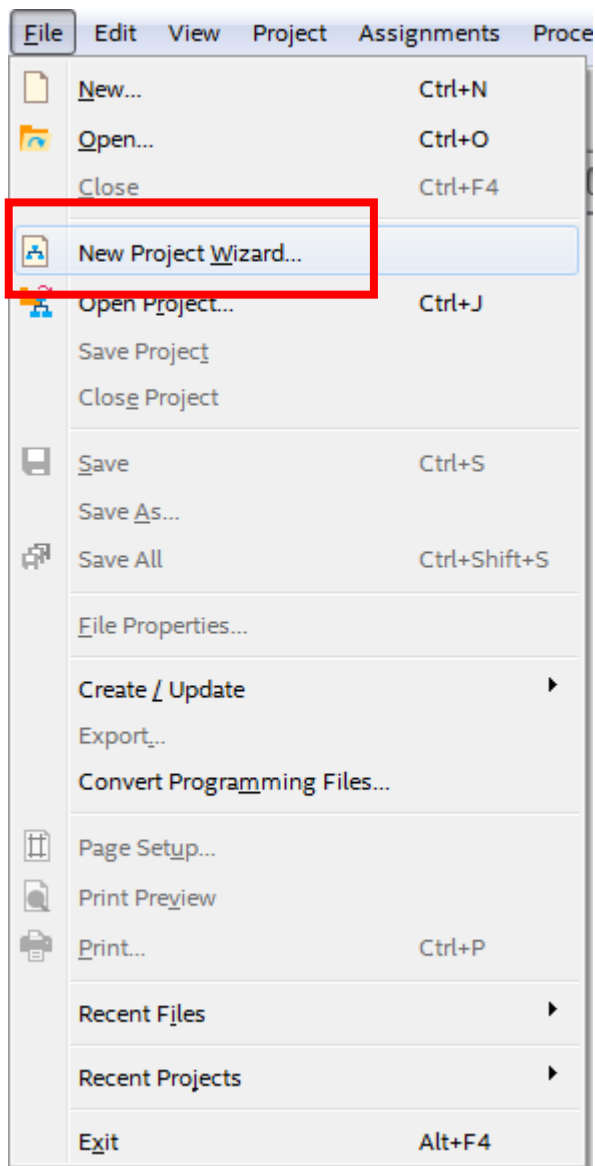
■ 位宽，深度

■ 如何写入数据？

- ✓ 数据初始化文件：hex 或 mif文件



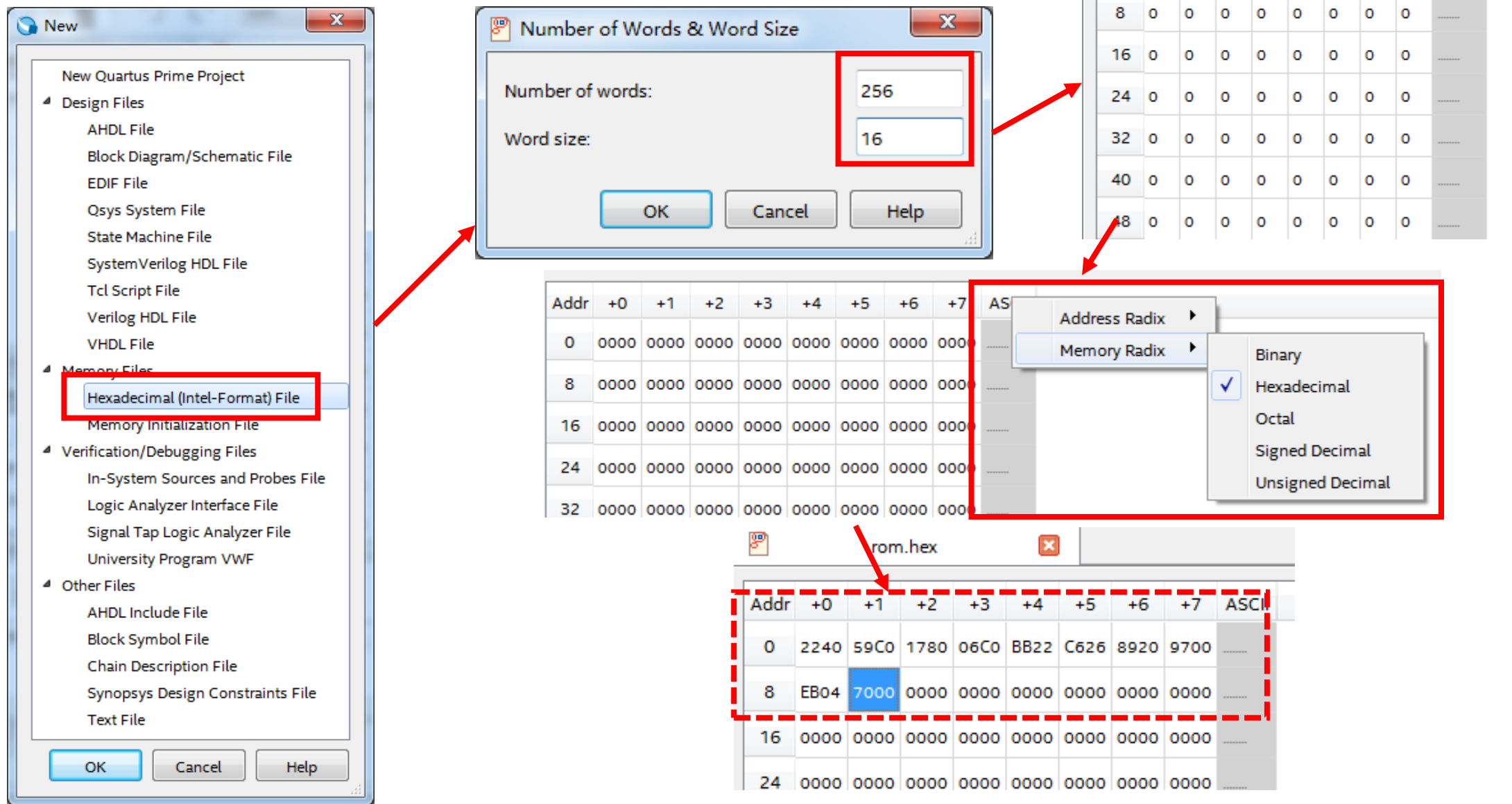
新建工程



FPGA型号任意

■ 建立数据初始化文件

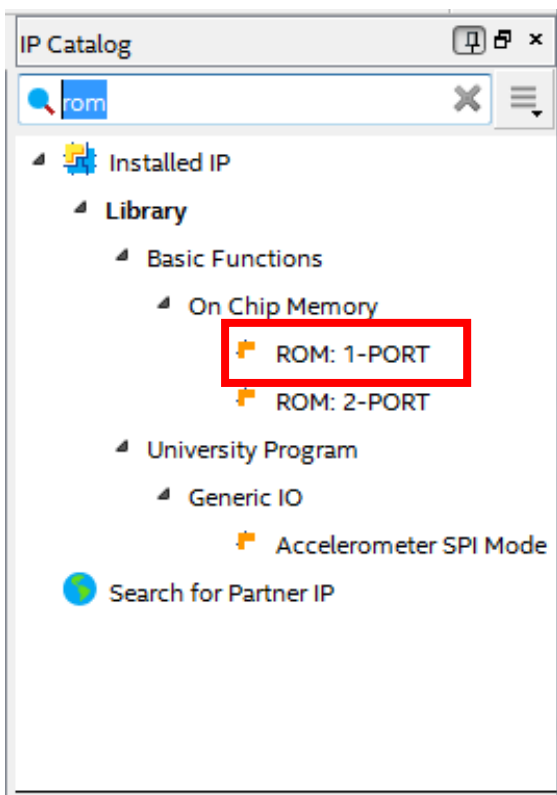
File->New->Hexadecimal (Intel-Format) File



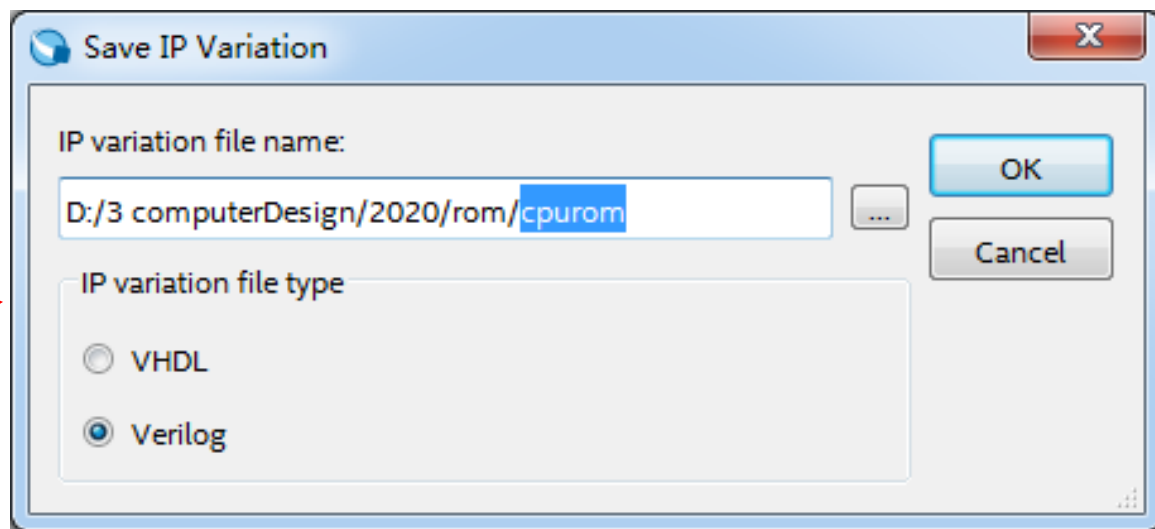
■ 搜索**ROM** IP

Tools->IP Catalog

■ 双击ROM:1-PORT



命名



注意：此模块的名称后面会用到，要前后一致

MegaWizard Plug-In Manager [page 1 of 5]

ROM: 1-PORT

About Documentation

1 Parameter Settings 2 EDA 3 Summary

General > Regs/Clen/Adrs > Mem Init >

Currently selected device family: Cyclone 10 LP

☒ Match project/default

How wide should the 'q' output bus be? 16 bits

How many 16-bit words of memory? 256 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

☒ Auto ☐ MLAB ☐ M9K

☐ M144K ☐ LCs Options...

Set the maximum block depth to Auto words

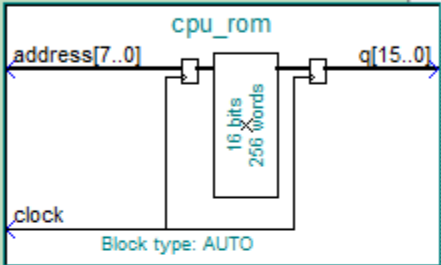
What clocking method would you like to use?

☒ Single clock ☐ Dual clock: use separate 'input' and 'output' clocks

Resource Usage

1 M9K

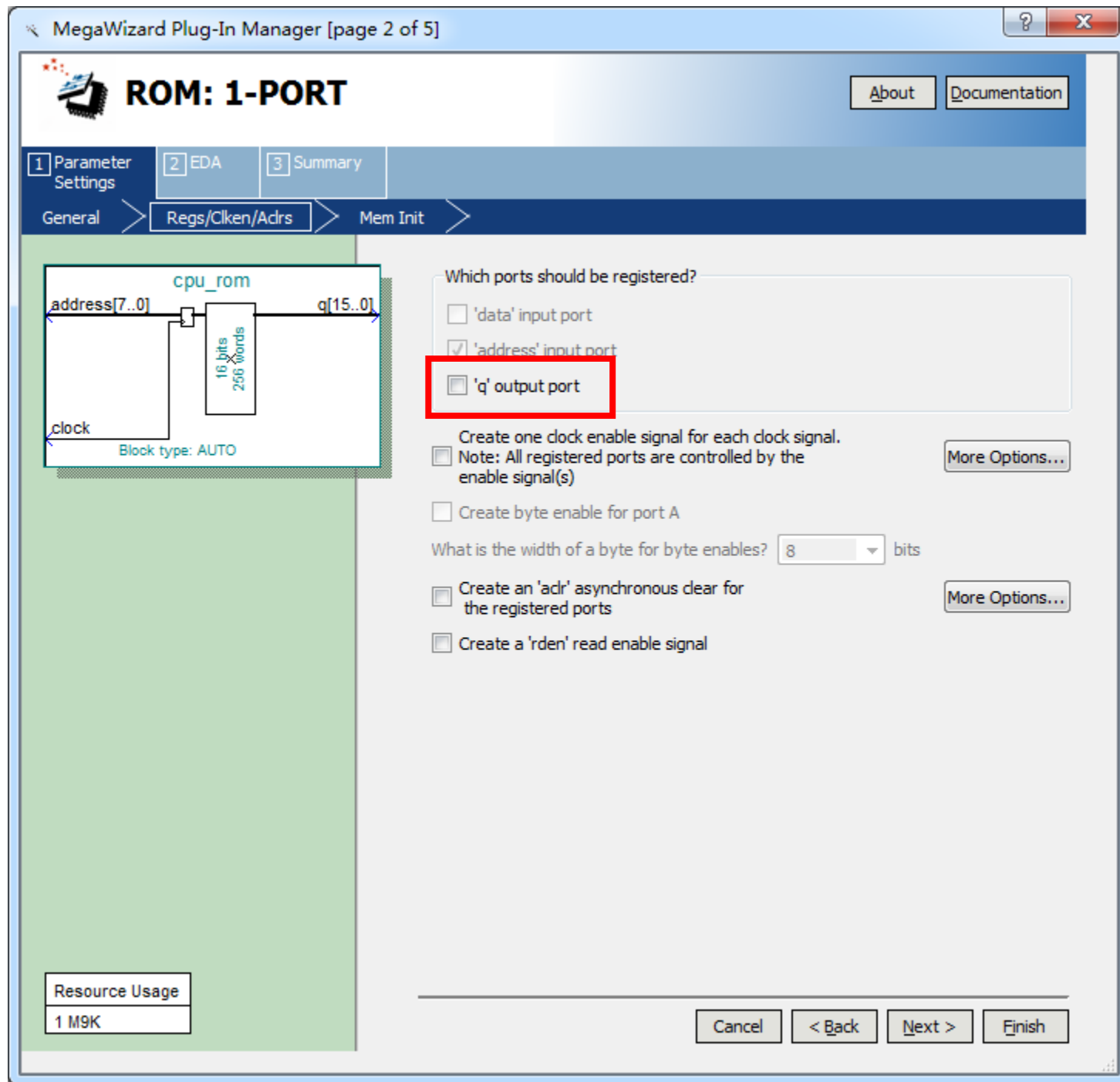
Cancel < Back Next > Finish

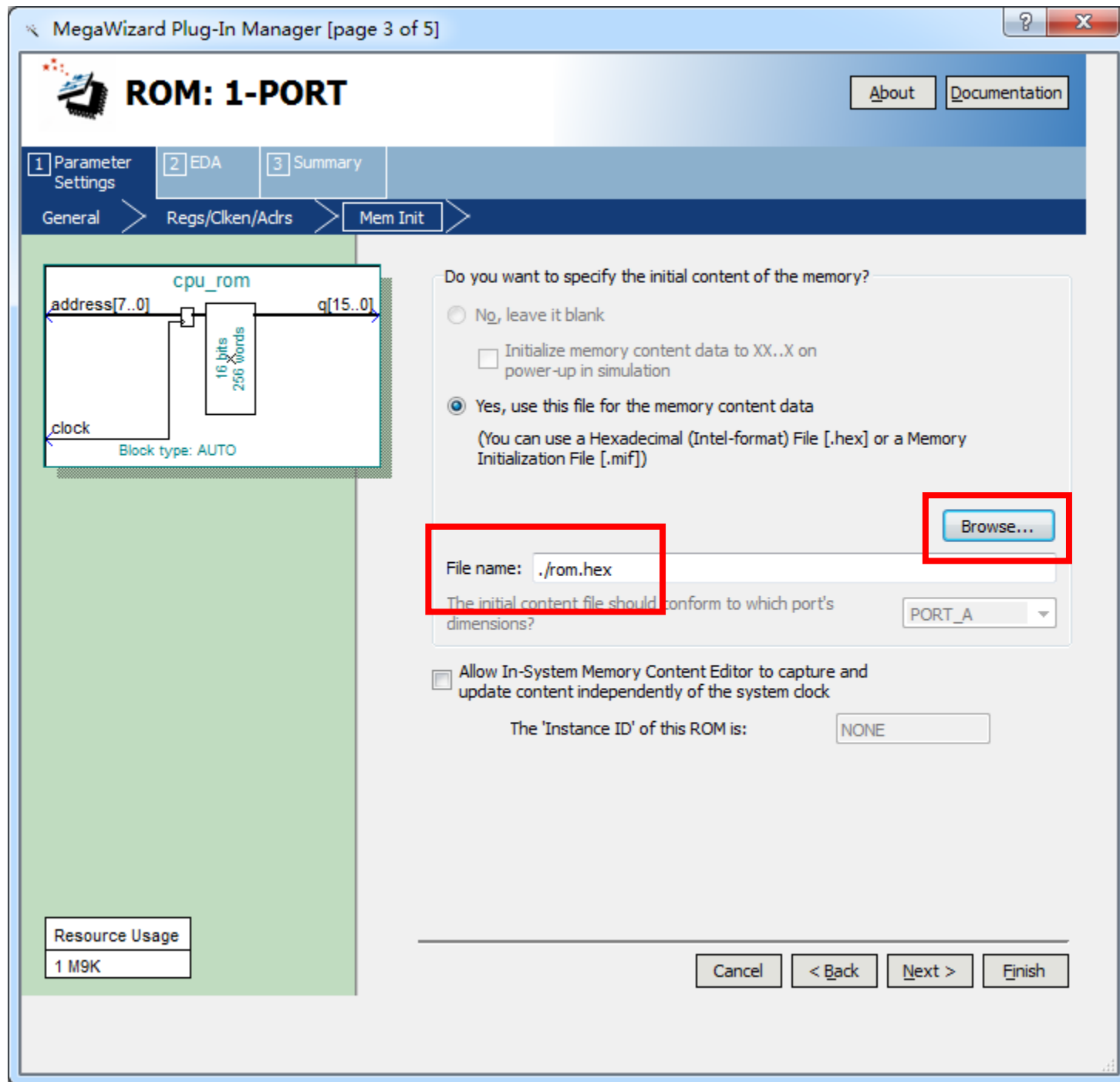


The diagram shows a block labeled 'cpu_rom'. It has an input 'address[7..0]' on the left, a clock input 'clock' at the bottom left, and an output 'q[15..0]' on the right. Inside the block, it is labeled '16 bits' and '256 words'. Below the block, it says 'Block type: AUTO'.

■ 位宽，深度

- 去掉输出端口的寄存器



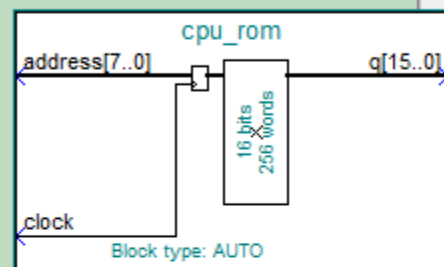


✓ 数据初始化文件：hex 或 mif 文件

要先建立.hex文件



ROM: 1-PORT

[About](#)[Documentation](#)[1 Parameter Settings](#)[2 EDA](#)[3 Summary](#)

Resource Usage

1 M9K

Simulation Libraries

To properly simulate the generated design files, the following simulation model file(s) are needed

File	Description
altera_mf	Altera megafunction simulation library

Timing and resource estimation

Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party EDA synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.

Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.

Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.

☐ Generate netlist[Cancel](#)[< Back](#)[Next >](#)[Finish](#)



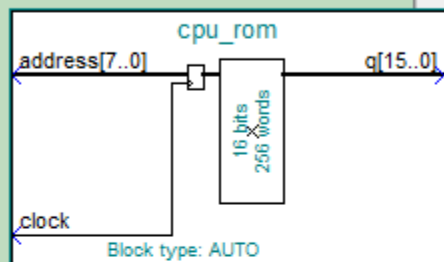
ROM: 1-PORT

[About](#)[Documentation](#)

1 Parameter Settings

2 EDA

3 Summary



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

D:\3 computerDesign\2020\rom\

File	Description
<input checked="" type="checkbox"/> cpu_rom.v	Variation file
<input type="checkbox"/> cpu_rom.inc	AHDL Include file
<input type="checkbox"/> cpu_rom.cmp	VHDL component declaration file
<input type="checkbox"/> cpu_rom.bsf	Quartus Prime symbol file
<input checked="" type="checkbox"/> cpu_rom_inst.v	Instantiation template file
<input checked="" type="checkbox"/> cpu_rom_bb.v	Verilog HDL black-box file

Resource Usage

1 M9K

Cancel

< Back

Next >

Finish

Quartus Prime IP Files

When you create an Intel IP variation, a Quartus Prime IP File is generated. Quartus Prime IP Files are used to represent the Intel IP in your design. Do you want to add the Quartus Prime IP File to the project?

☒ D:\3 computerDesign\2020\rom\cpu_rom.qip

☐ Automatically add Quartus Prime IP Files to all projects

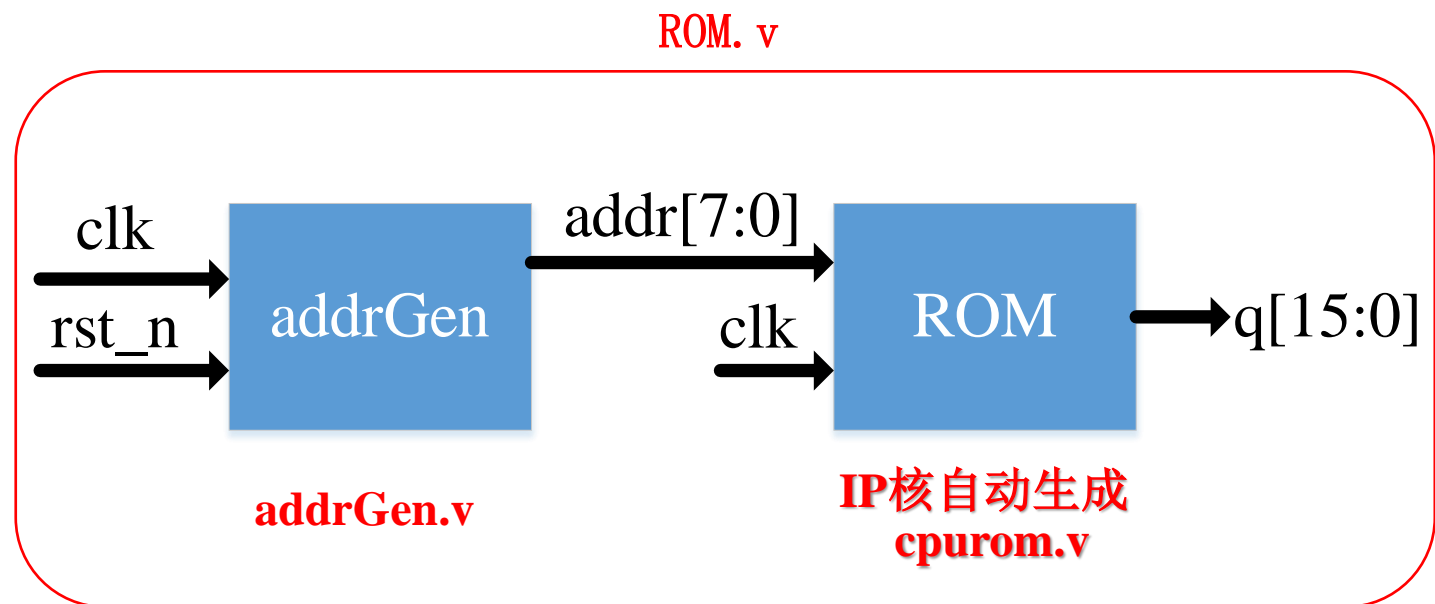
(Note: Turning on this option permanently suppresses this dialog box. You can change this setting in the Options dialog box)

Yes

No

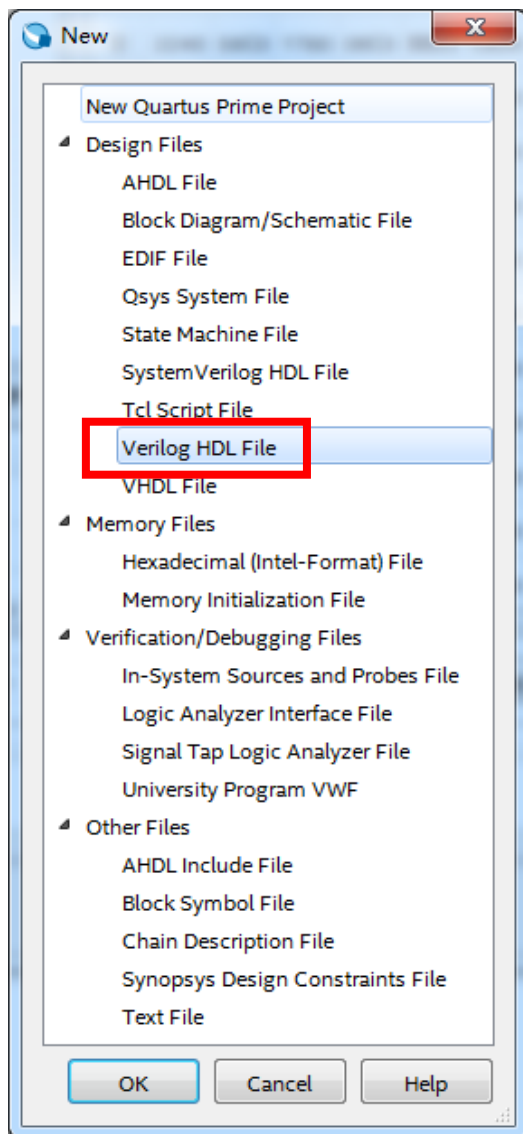
Help

测试新建立的ROM模块



此处为了说明如何使用IP核，增加了一个addrGen

■ 建立AddrGen.v和ROM.v



addrGen.v

仅供参考

```
module addrGen(clk,rst_n,addr);
    input clk;
    input rst_n;
    output reg [7:0] addr;
    always @ (posedge clk or negedge rst_n)
        begin
            .....
            .....
        end
endmodule
```

顶层文件

ROM.v

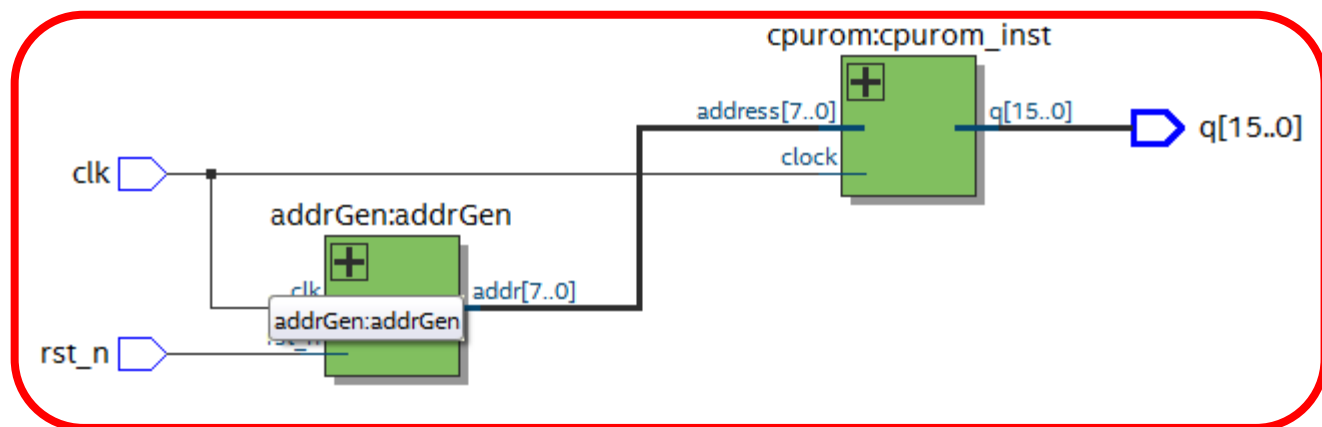
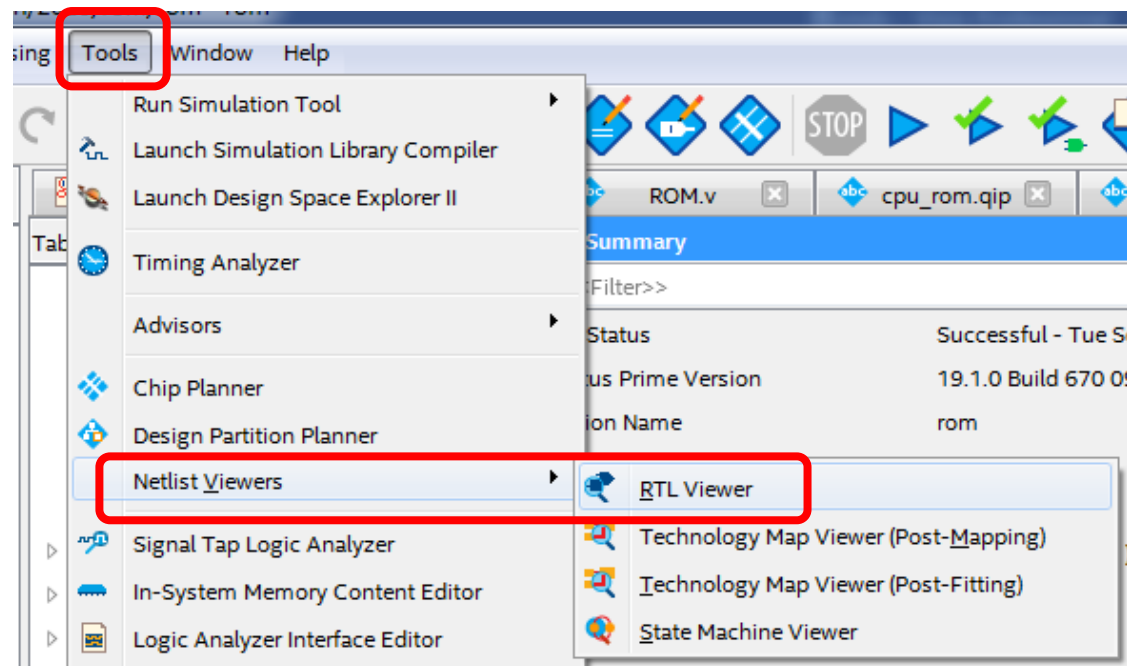
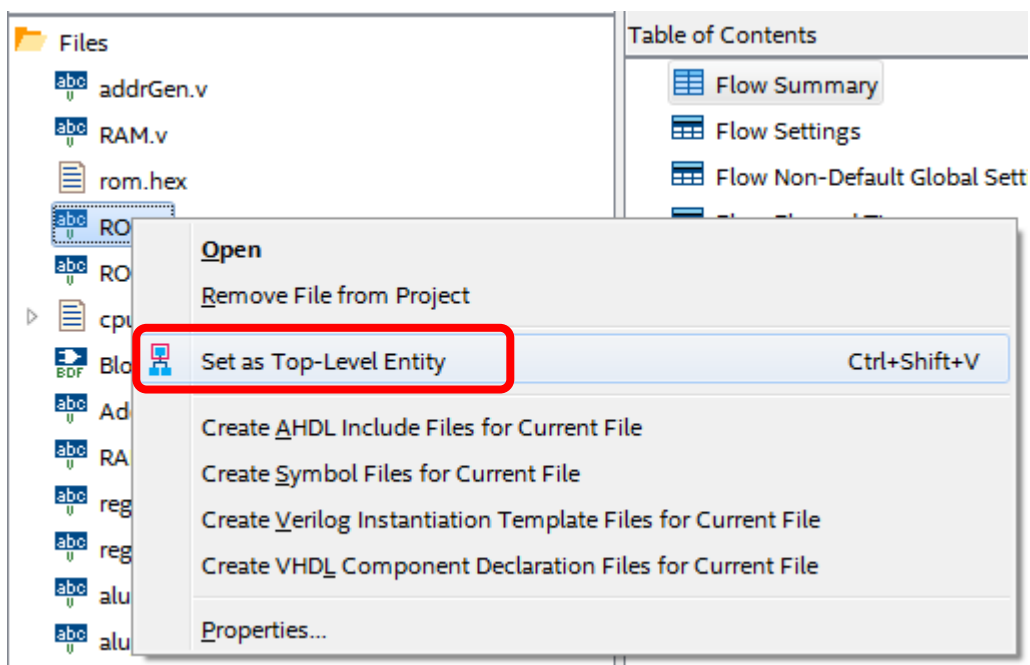
```
module ROM(clk,rst_n,q);
    input clk;
    input rst_n;
    output [15:0] q;
    wire [7:0] addr;
    addrGen addrGen(
        .clk(clk),
        .rst_n(rst_n),
        .addr(addr)
    );
    cpurom cpu_rom_inst(
        .address(addr),
        .clock(clk),
        .q(q)
    );
endmodule
```

IP核自动生成
cpurom.v

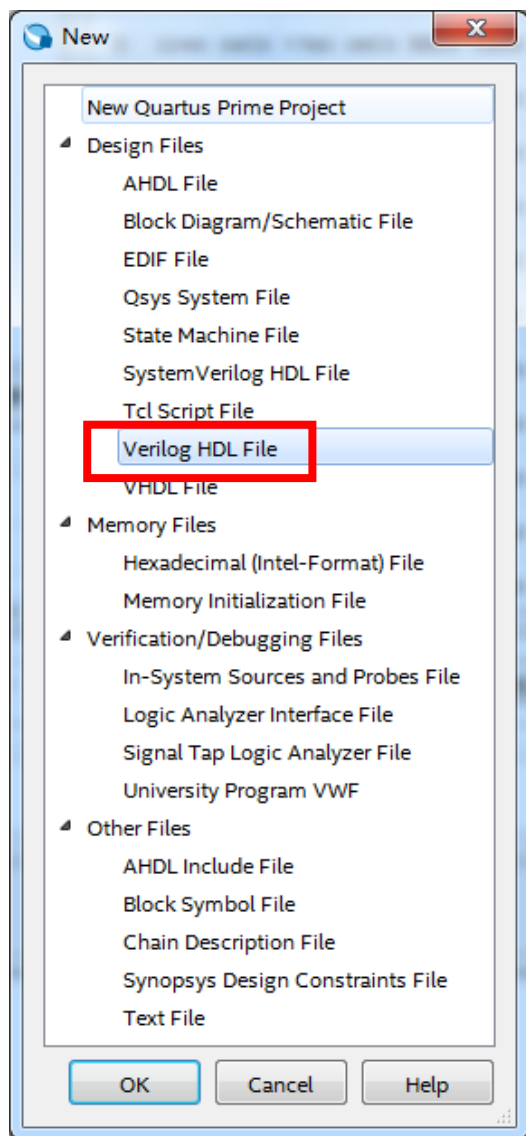
```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module cpurom (
    address,
    clock,
    q);
    input [7:0] address;
    input clock;
    output [15:0] q;
```

■ 设置**ROM.v**为顶层文件

■ 编译

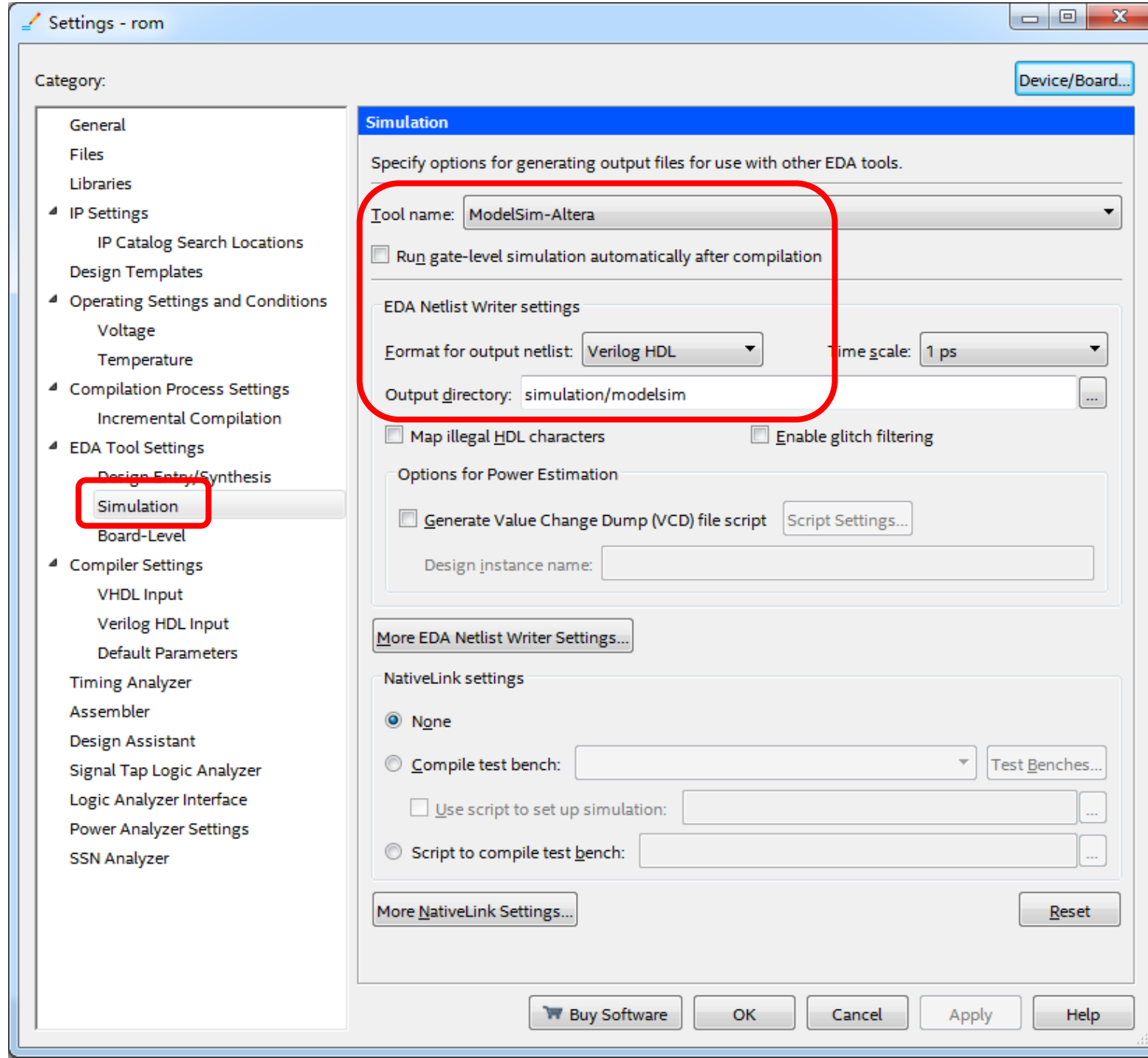
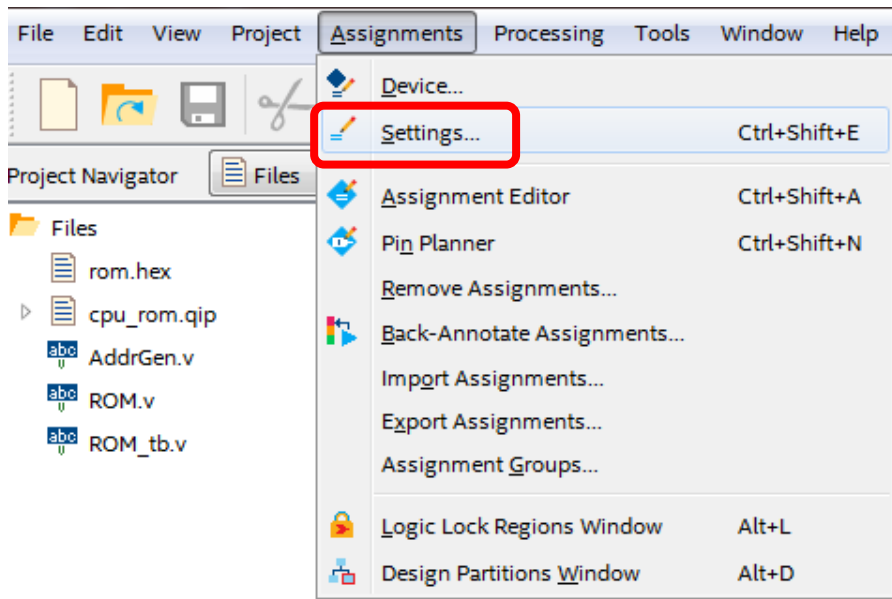


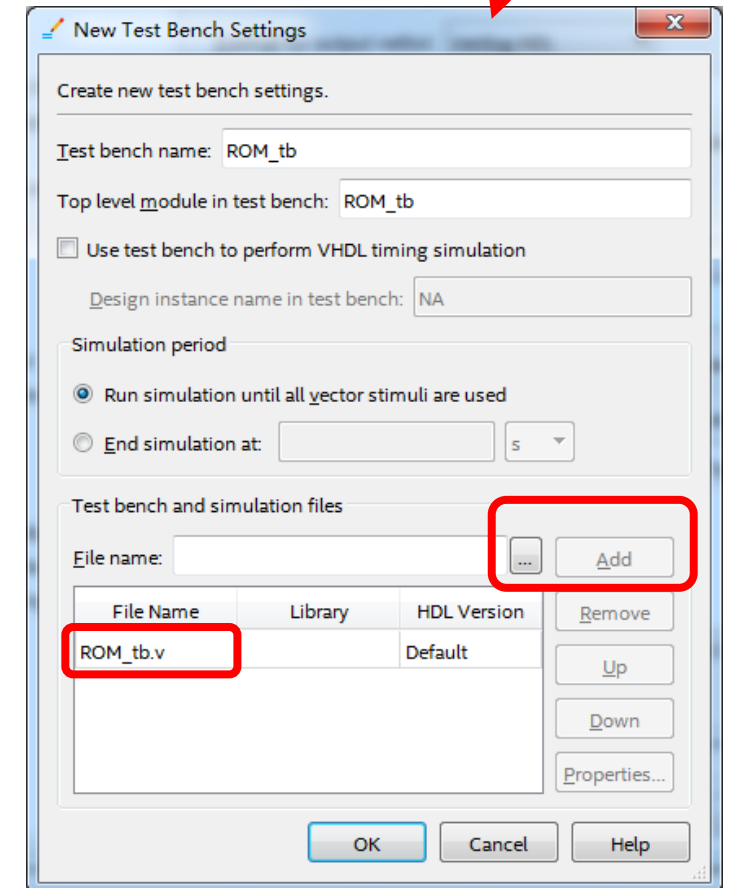
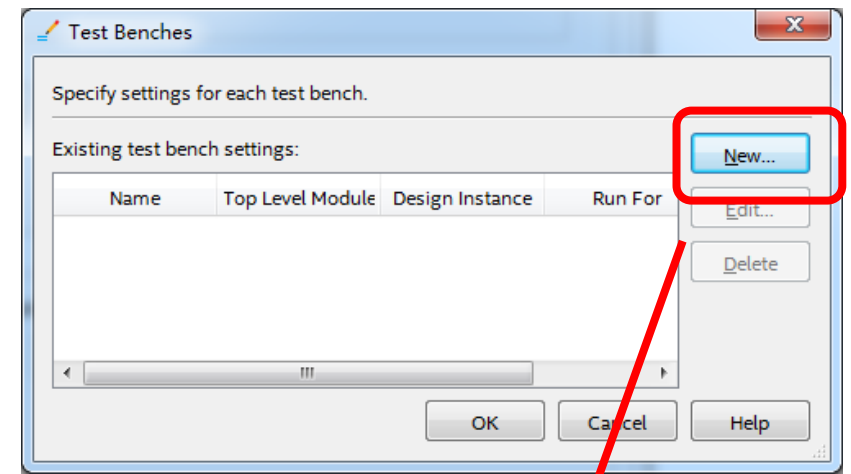
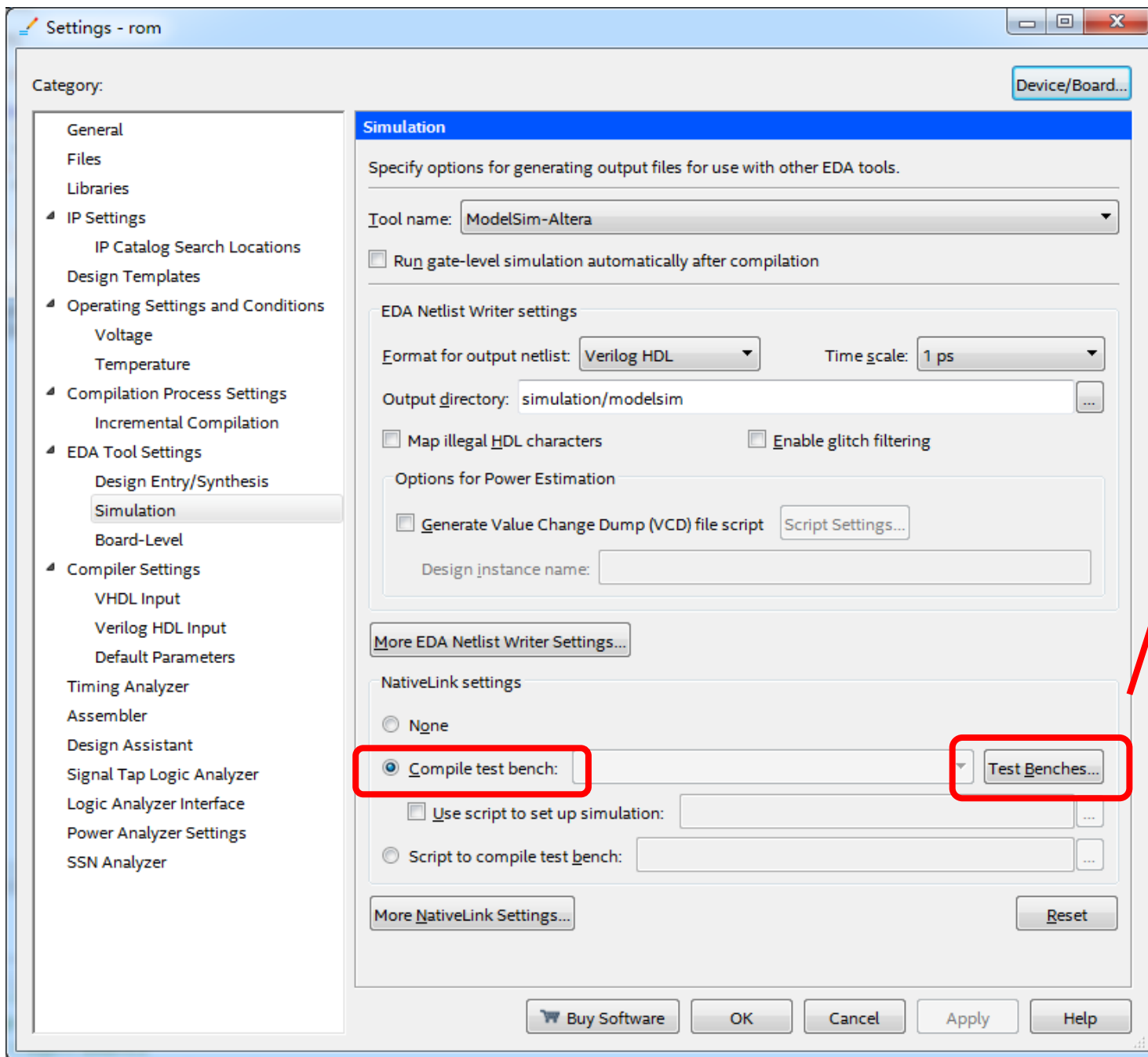
■ 编写测试代码ROM_tb.v

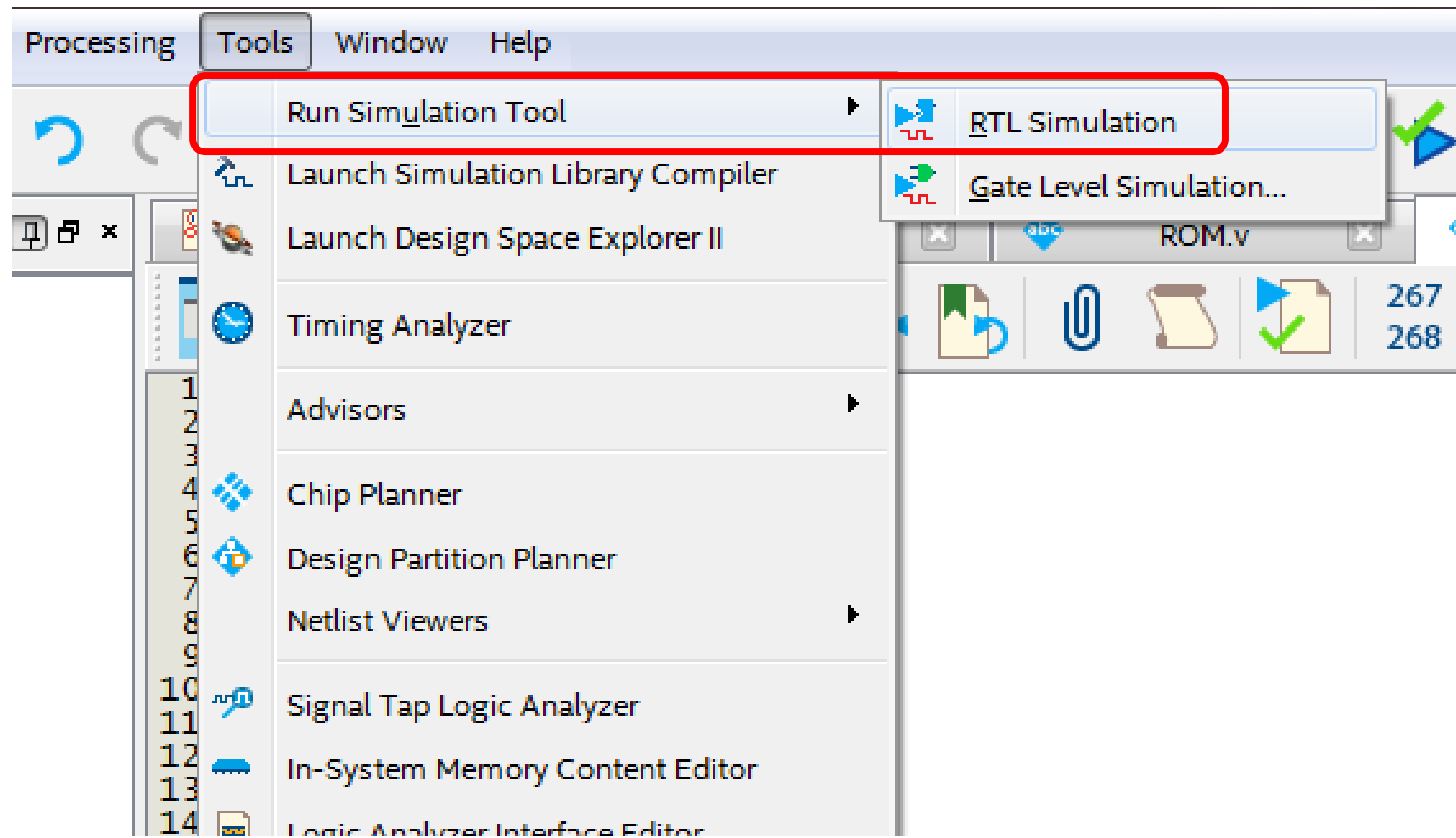


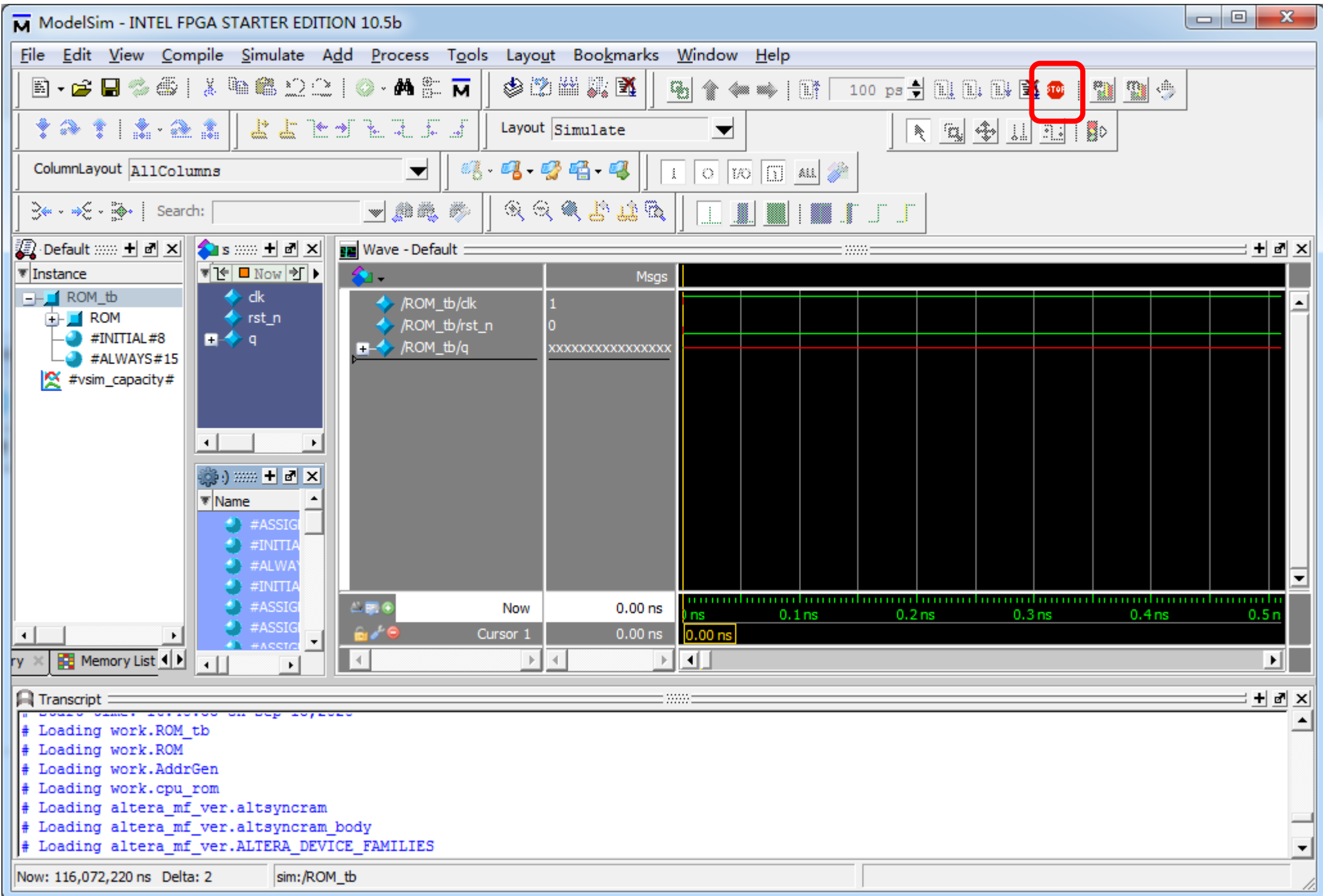
仅供参考

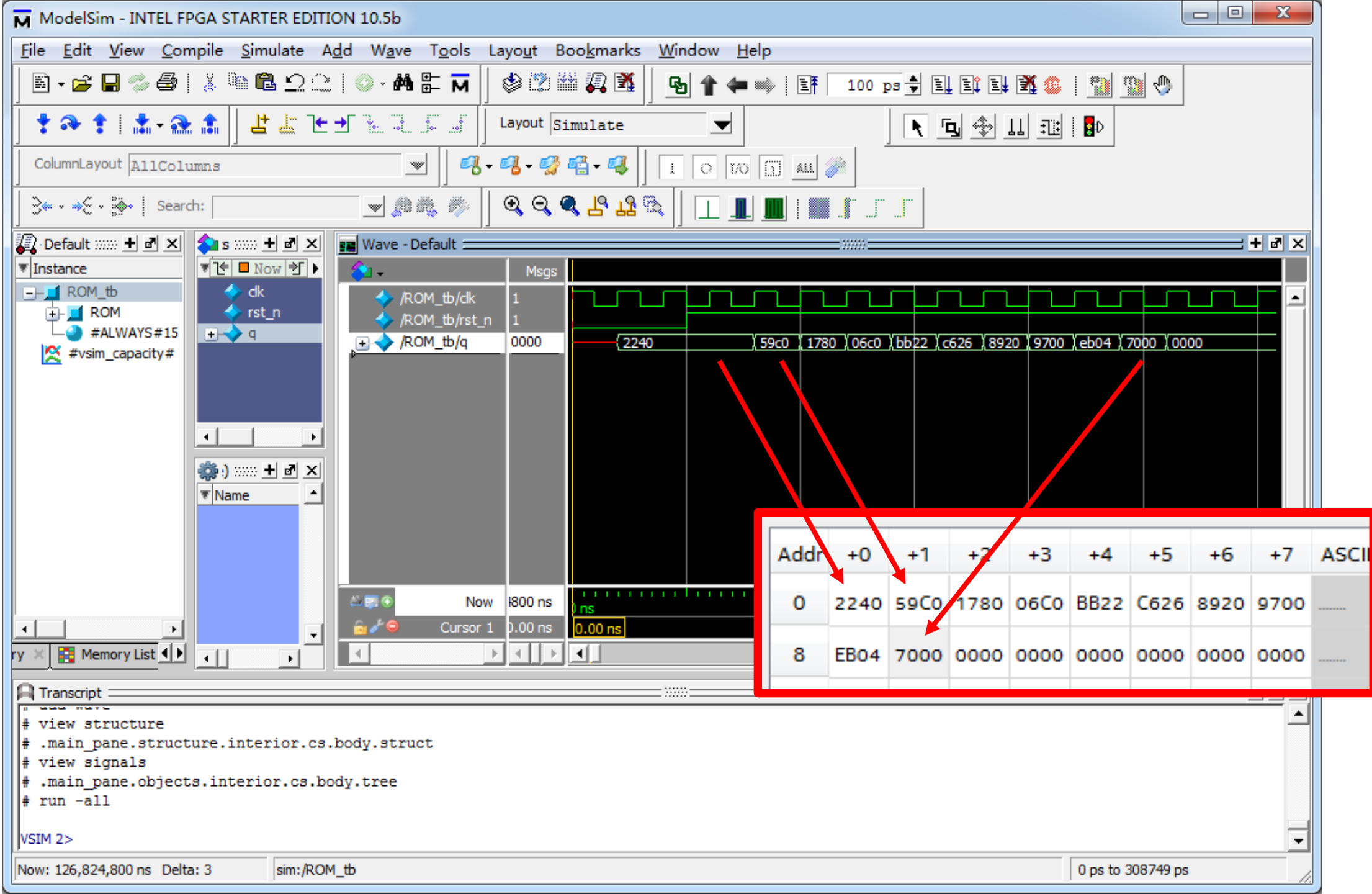
```
`timescale 1ns/1ps
module ROM_tb;
    reg clk;
    reg rst_n;
    wire [15:0] q;
    initial begin
        clk = 0;
        rst_n = 0;
        #50.1
        rst_n = 1;
    end
    always # 10 clk = ~clk;
    ROM ROM(
        .clk(clk),
        .rst_n(rst_n),
        .q(q)
    );
endmodule
```



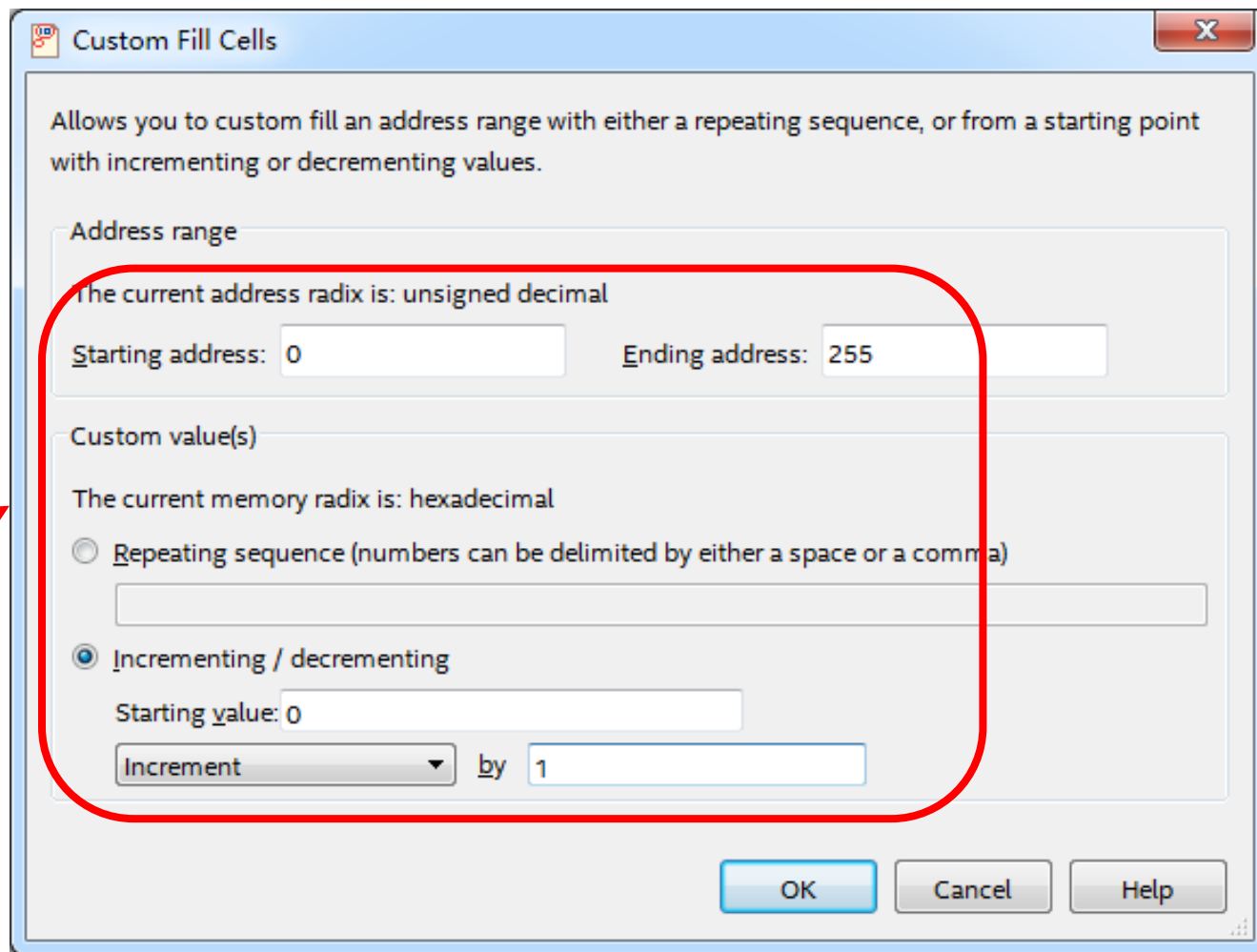
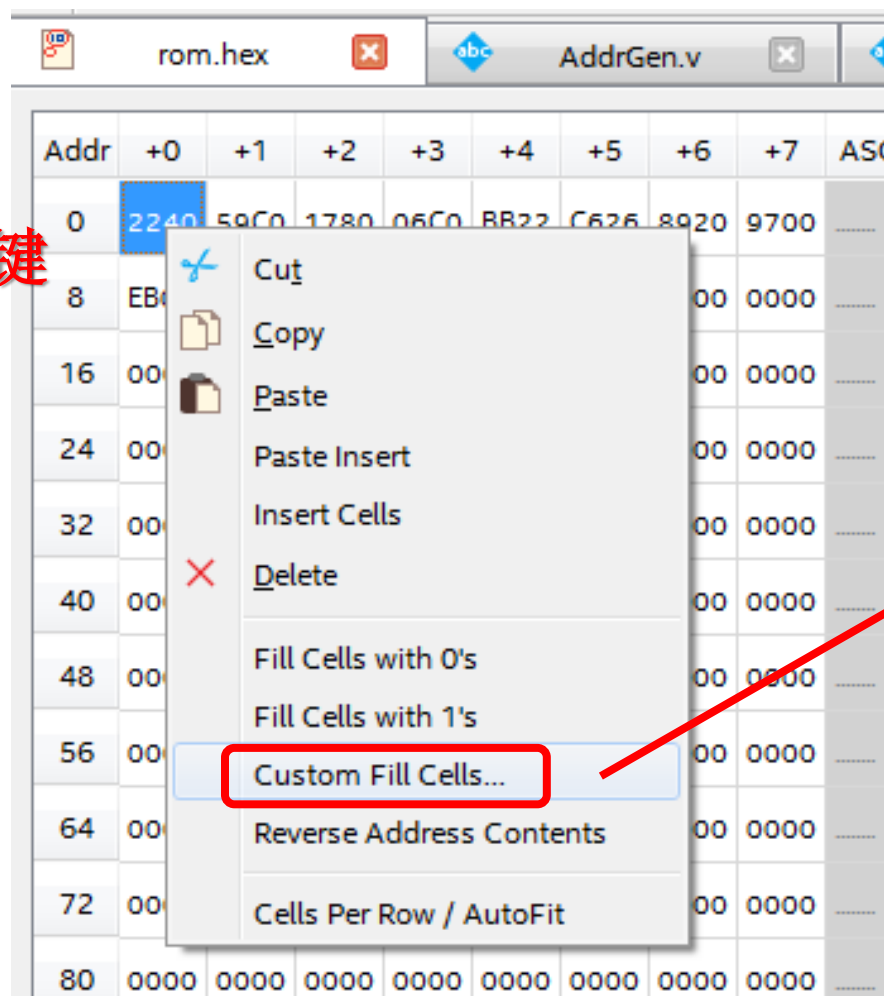






修改ROM中的初始化数据

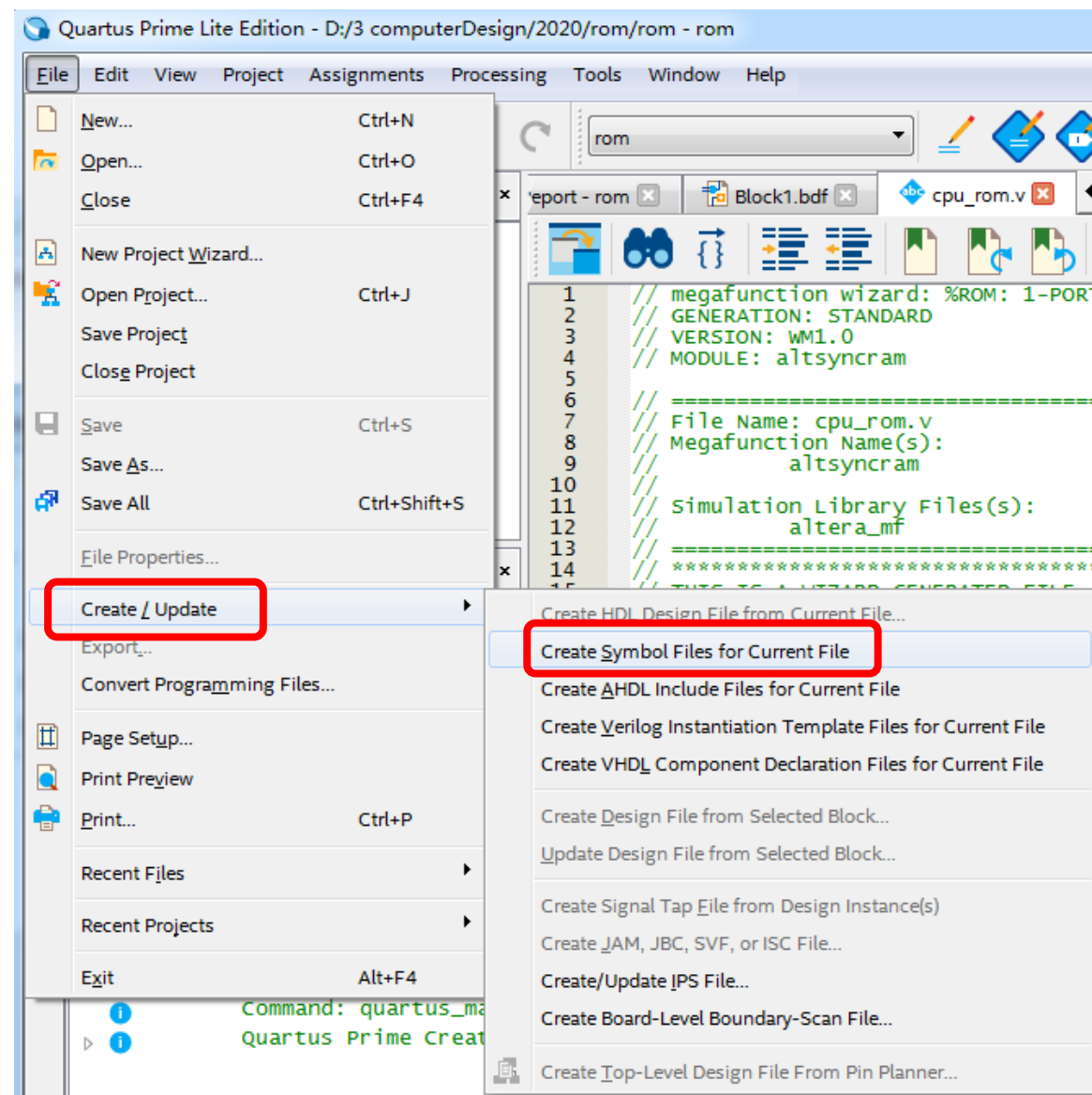
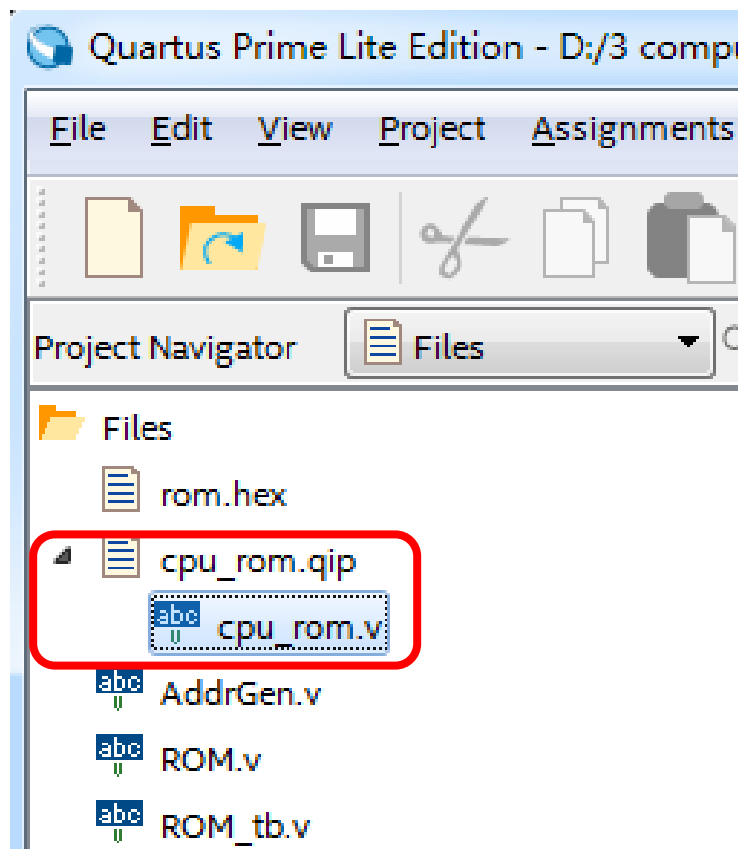
右键



rom.hex*									
AddrGen.v									
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0000	0001	0002	0003	0004	0005	0006	0007
8	0008	0009	000A	000B	000C	00...	000E	000F
16	0010	0011	0012	0013	0014	0015	0016	0017
24	0018	0019	001A	001B	001C	00...	001E	001F
32	0020	0021	0022	0023	0024	0025	0026	0027	!"#...
40	0028	0029	002A	002B	002C	00...	002E	002F	()*+,-./
48	0030	0031	0032	0033	0034	0035	0036	0037	012...
56	0038	0039	003A	003B	003C	00...	003E	003F	89;...
64	0040	0041	0042	0043	0044	0045	0046	0047	@A...
72	0048	0049	004A	004B	004C	00...	004E	004F	HIJ...
80	0050	0051	0052	0053	0054	0055	0056	0057	PQR...
88	0058	0059	005A	005B	005C	00...	005E	005F	XYZ...
96	0060	0061	0062	0063	0064	0065	0066	0067	`ab...
104	0068	0069	006A	006B	006C	00...	006E	006F	hijkl...
112	0070	0071	0072	0073	0074	0075	0076	0077	pqrs...
120	0078	0079	007A	007B	007C	00...	007E	007F	xyz[...
128	0080	0081	0082	0083	0084	0085	0086	0087
136	0088	0089	008A	008B	008C	00...	008E	008F

144	0090	0091	0092	0093	0094	0095	0096	0097
152	0098	0099	009A	009B	009C	00...	009E	009F
160	00A0	00A1	00A2	00A3	00A4	00A5	00A6	00A7
168	00A8	00A9	00AA	00AB	00AC	00...	00AE	00AF
176	00B0	00B1	00B2	00B3	00B4	00B5	00B6	00B7
184	00B8	00B9	00BA	00BB	00BC	00...	00BE	00BF
192	00C0	00C1	00C2	00C3	00C4	00C5	00C6	00C7
200	00C8	00C9	00CA	00CB	00CC	00...	00CE	00CF
208	00...	00...	00...	00...	00...	00...	00...	00...
216	00...	00...	00...	00...	00...	00...	00DE	00DF
224	00E0	00E1	00E2	00E3	00E4	00E5	00E6	00E7
232	00E8	00E9	00EA	00EB	00EC	00ED	00EE	00EF
240	00F0	00F1	00F2	00F3	00F4	00F5	00F6	00F7
248	00F8	00F9	00FA	00FB	00FC	00FD	00FE	00FF

转成符号



实验任务1

任务1.1

- (1) 参照上面的步骤，实现对ROM模块【16位】的仿真验证；修改ROM中的初始化数据，给出仿真结果.
- (2) 建立32位位宽的ROM模块；给出仿真结果.

2、寄存器组模块设计

采用Quartus Prime

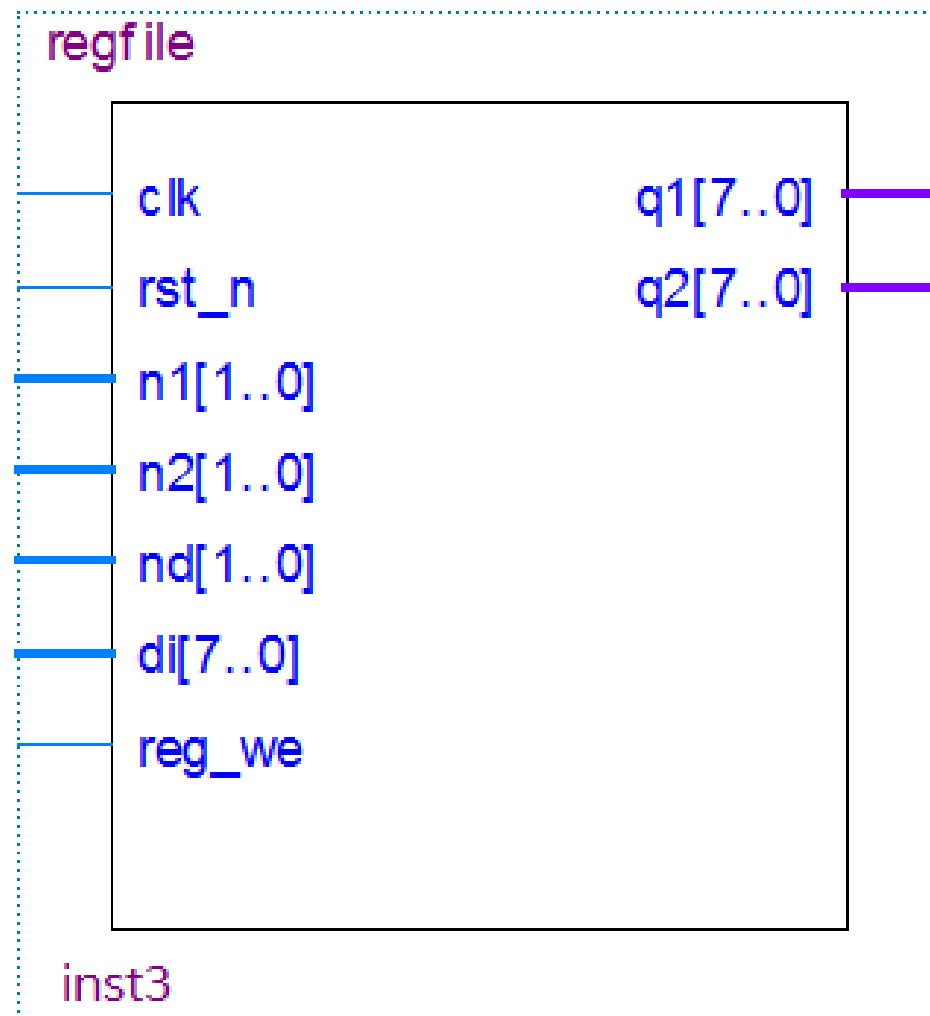
寄存器组模块

■ 输入信号

- ✓ n1[1..0]: 读通道1的寄存器号
- ✓ n2[1..0]: 读通道2的寄存器号
- ✓ nd[1..0]: 写通道的寄存器号
- ✓ di[7..0]: 写通道的输入数据
- ✓ clk: 时钟脉冲信号, 上升沿有效
- ✓ reg_we: 写允许; 为1时, 在clk上升沿, 将数据di写入nd指定的寄存器; 为0时, 禁止对寄存器组进行写操作
- ✓ rst_n: 异步复位信号, 清空所有寄存器的内容

■ 输出信号

- ✓ q1[7..0]: 输出n1[1..0]指定寄存器的内容
- ✓ q2[7..0]: 输出n2[1..0]指定寄存器的内容



仅供参考

```
module regfile(clk,rst_n,n1,n2,nd,di,reg_we,q1,q2);  
    input clk;  
    input rst_n;  
    input [1:0] n1;  
    input [1:0] n2;  
    input [1:0] nd;  
    input [7:0] di;  
    input reg_we;  
    output [7:0] q1;  
    output [7:0] q2;  
    reg [7:0] rf[3:0];  
  
    .....  
    .....  
endmodule
```

regfile.v

```
.....  
initial begin
```

```
    clk = 0;
```

```
    rst_n = 0;
```

```
    n1 = 0;
```

```
    n2 = 0; regfile_tb.v
```

```
    nd = 0;
```

```
    di = 0;
```

```
    reg_we = 0;
```

```
    #100.1 rst_n = 1;
```

```
    #100 nd = 0;
```

```
        di = 11;
```

```
    #100 nd = 0;
```

```
        di = 12;
```







```
    #100 nd = 0;
```

```
        di = 11;
```

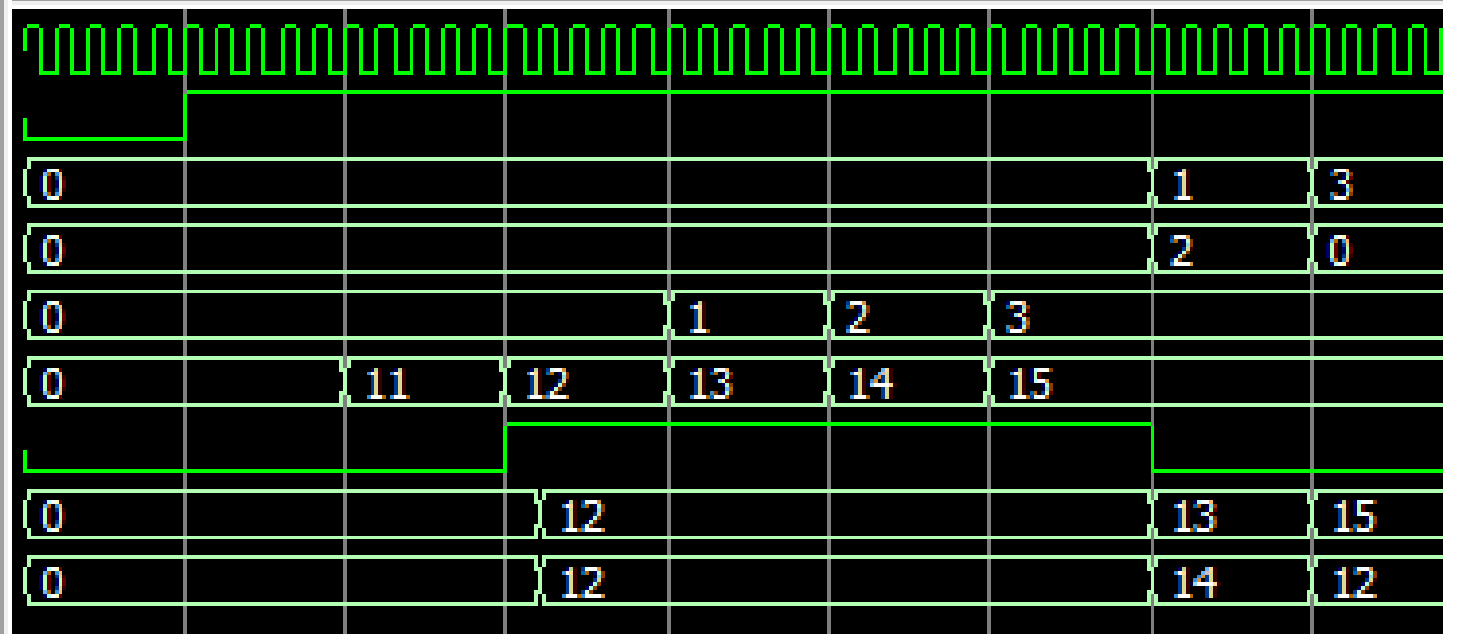
```
        reg_we = 1;
```

```
    .....  
    .....  
end
```

```
.....
```

 /regfile_tb/dk
 /regfile_tb/rst_n
  /regfile_tb/n1
  /regfile_tb/n2
  /regfile_tb/nd
  /regfile_tb/di
 /regfile_tb/reg_we
  /regfile_tb/q1
  /regfile_tb/q2

1
1
3
0
3
15
0
15
12



实验任务1

任务1.2

(1) 参照上面的步骤，实现对寄存器组模块【8位，4个】的仿真验证；分析仿真结果.

(2) 建立32位位宽，包含32个寄存器的寄存器组模块；给出仿真结果.

3、ALU模块设计

采用Quartus Prime

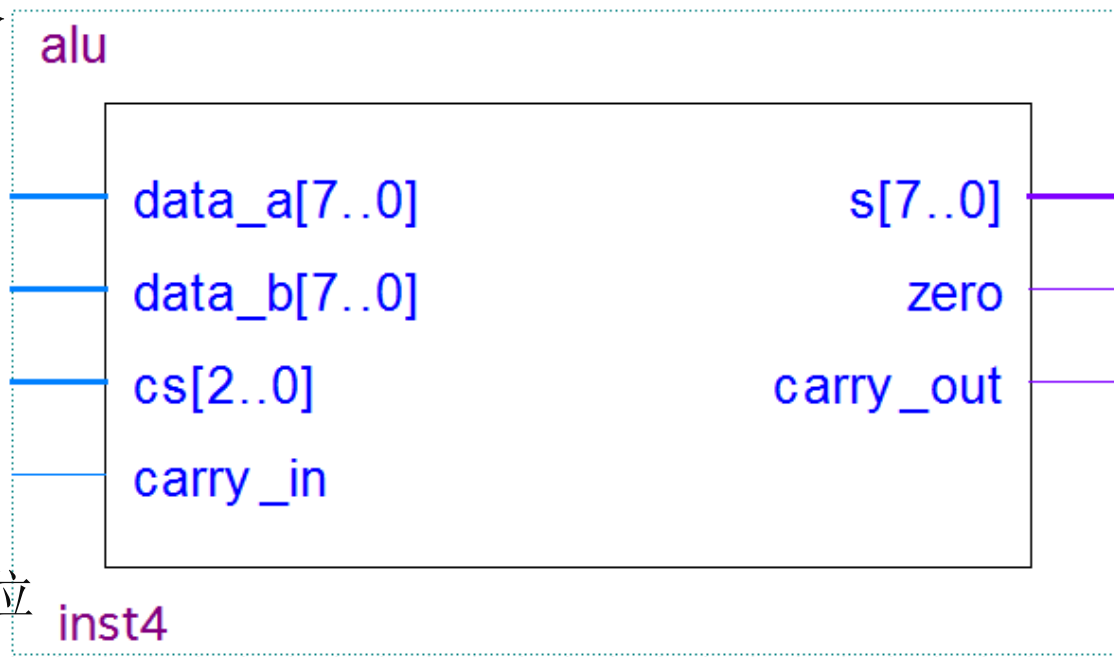
ALU模块

■ 输入信号

- ✓ data_a[7..0]、data_b[7..0]: 参与运算的两个8位二进制数
- ✓ cs[2..0]: 存放7种运算操作的编码
- ✓ carry_in: 参与运算的进/借位值

■ 输出信号

- ✓ s[7..0]: 运算结果
- ✓ zero: 零标志; 当s[7..0]为0时, zero=1; 否则zero=0;
- ✓ carry_out: 进位/借位标志; 当加法运算过程最高位有进位时, carry_out=1; 否则carry_out=0; 当减法运算最高位产生借位时, carry_out=0; 否则, carry_out=1



简单计算机系统指令集

操作名称	操作码	汇编语言格式指令	执行操作
与	0000	AND Rd, Rs, Rt	$Rd \leftarrow Rs \text{ and } Rt$; $PC \leftarrow PC + 1$
或	0001	OR Rd, Rs, Rt	$Rd \leftarrow Rs \text{ or } Rt$; $PC \leftarrow PC + 1$
不带进位加	0010	ADD Rd, Rs, Rt	$Rd \leftarrow Rs + Rt$; $PC \leftarrow PC + 1$
不带借位减	0011	SUB Rd, Rs, Rt	$Rd \leftarrow Rs - Rt$; $PC \leftarrow PC + 1$
无符号数比较	0100	SLT Rd, Rs, Rt	If $Rs < Rt$, $Rd = 1$ else $Rd = 0$; $PC \leftarrow PC + 1$
带借位减	0101	SUBC Rd, Rs, Rt	$Rd \leftarrow Rs - Rt - (1 - C)$; $PC \leftarrow PC + 1$
带进位加	0110	ADDC Rd, Rs, Rt	$Rd \leftarrow Rs + Rt + C$; $PC \leftarrow PC + 1$
立即数与	1000	ANDI Rt, Rs, imm	$Rt \leftarrow Rs \text{ and } imm$; $PC \leftarrow PC + 1$
立即数或	1001	ORI Rt, Rs, imm	$Rt \leftarrow Rs \text{ or } imm$; $PC \leftarrow PC + 1$
立即数加	1010	ADDI Rt, Rs, imm	$Rt \leftarrow Rs + imm$; $PC \leftarrow PC + 1$
读存储器	1011	LW Rt, Rs, imm	$Rt \leftarrow \text{MEM}[Rs + imm]$; $PC \leftarrow PC + 1$
写存储器	1100	SW Rt, Rs, imm	$\text{MEM}[Rs + imm] \leftarrow Rt$; $PC \leftarrow PC + 1$
无条件跳转	0111	JMP imm	$PC \leftarrow imm$
相等时跳转	1101	BEQ Rs, Rt, imm	If $Rt = Rs$, $PC \leftarrow PC + imm + 1$ else $PC \leftarrow PC + 1$
不等时跳转	1110	BNE Rs, Rt, imm	If $Rt \neq Rs$, $PC \leftarrow PC + imm + 1$ else $PC \leftarrow PC + 1$

alu.v

```
module alu (data_a,data_b,s,zero,cs,carry_in,carry_out);
  input [7:0] data_a,data_b;
  input [2:0] cs;
  input carry_in;
  output reg [7:0] s;
  output zero;
  output reg carry_out;
  .....
  .....
  always @(*)
  begin
    case (cs)
      .....
      .....
    endcase
  end
endmodule
```

仅供参考

alu_tb.v

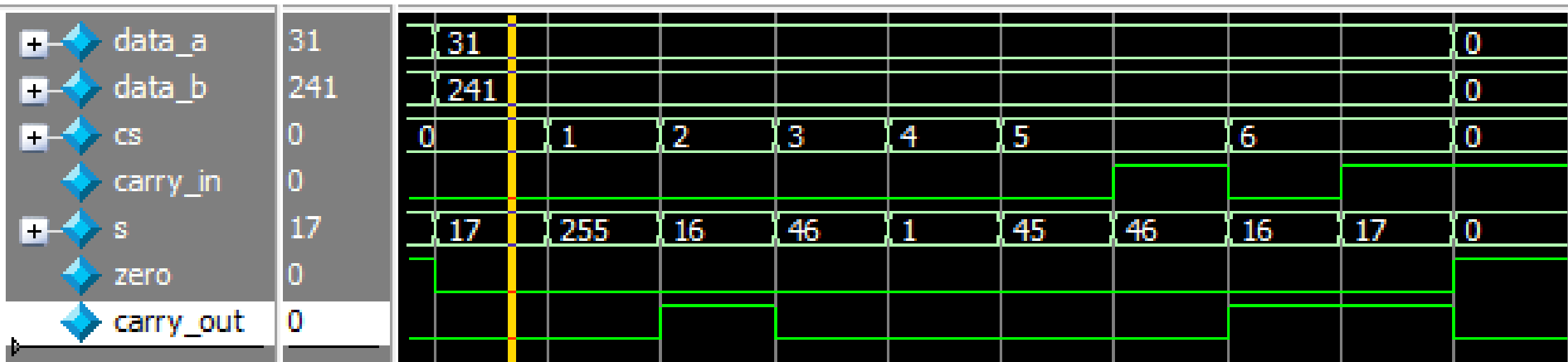
```
.....
initial begin
  data_a = 0;
  data_b = 0;
  cs = 0;
  carry_in = 0;
  #100 data_a = 8'h1f;
        data_b = 8'hf1;
        cs = AND;
  #100 cs = OR;
  #100 cs = ADD;
  .....
  #100;
end
.....
```

parameter

AND = 3'b000,
OR = 3'b001,
ADD = 3'b010,
SUB = 3'b011,
SLT = 3'b100,
SUBC= 3'b101,
ADDC= 3'b110;

```
data_a = 0;  
data_b = 0;  
cs = 0;  
carry_in = 0;  
#100 data_a = 8'h1f;  
        data_b = 8'hf1;  
cs = AND;  
#100 cs = OR;  
#100 cs = ADD;  
#100 cs = SUB;  
#100 cs = SLT;
```

```
#100 cs = SUBC;  
#100 cs = SUBC;  
carry_in = 1;  
#100 cs = ADDC;  
        carry_in = 0;  
#100 cs = ADDC;  
        carry_in = 1;  
#100 data_a = 0;  
        data_b = 0;  
cs = AND;  
#100;
```



实验任务1

任务1.3

- (1) 参照上面的步骤，实现对alu模块【8位】的仿真验证，分析仿真结果
- (2) 建立32位alu模块，给出仿真结果

THE END