Experiment 4
Hardware Trojan Attack I

We describe an experiment on hardware Trojan attacks, in the form of malicious modifications of electronic hardware, that pose major security concerns in the electronics industry.

Instructor: Dr. Swarup Bhunia
Co-Instructors/TAs: Shuo Yang and Reiner Dizon-Paradis

Case Study

We describe an experiment on hardware Trojan attacks, in the form of malicious modifications of electronic hardware, that pose major security concerns in the electronics industry.

In this chapter, we describe an experiment on hardware Trojan attacks and countermeasures. Emerging trend of outsourcing the design and fabrication services to external facilities as well as increasing reliance on third-party Intellectual Property (IP) cores and electronic design automation (EDA) tools makes integrated circuits (ICs) increasingly vulnerable to hardware Trojan attacks at different stages of its life-cycle. The modern IC design, fabrication, test and deployment stages highlight the level of trust at each stage. This scenario raises a new set of challenges for trust validation against malicious design modification at various stages of an IC life-cycle, where untrusted components/personnel are involved. In particular, it emphasizes the requirement of reliable detection of malicious design modification made in an untrusted fabrication facility, during the post-manufacturing test. It also imposes a requirement for trust validation in IP cores obtained from untrusted third-party vendors.

Figure 1 illustrates different steps of a typical IC life cycle and the possibility of Trojan attacks in these steps. Each party associated with the design and fabrication of an IC can be a potential adversary who can tamper it. Such tampering can be accomplished through add/delete/alteration of circuit structure or through modification of manufacturing process steps that cause reliability issues in ICs. From an attacker's perspective, the objective of such attacks can be manifold, e.g., to malign the image of a company to gain competitive edge in the market; disrupt major national infrastructure by causing malfunction in electronics used in mission-critical systems; or leak secret information from inside a chip to illegally access a secure system.

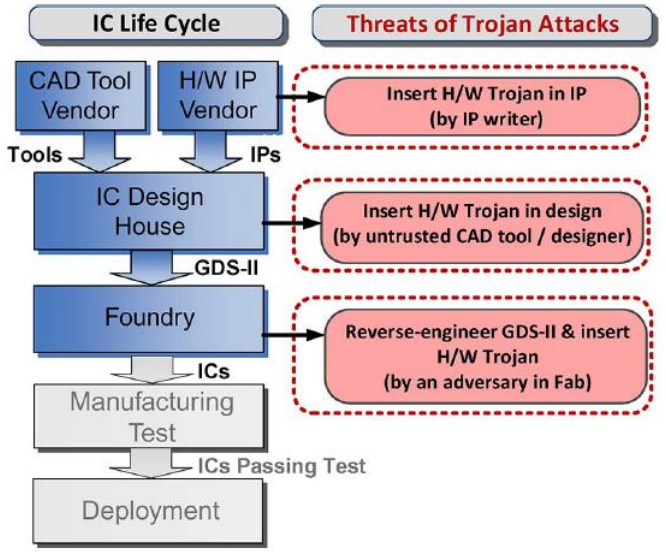


Figure 1 Hardware Trojan attacks by different parties at different stages of IC cycle.

Theory Background

Recently Intel announced a flaw in the implementation of the "TSX" instruction for its Haswell series of Central Processing Unit (CPU). This announcement came almost a year into the product's lifecycle and almost three years since the beginnings of Haswell's architecture was laid out. This is a legitimate mistake on Intel's part – there is no foul play or trickery here.

However, researchers at the University of Massachusetts were able to modify an Intel Ivy Bridge processor – the series that Haswell replaced – and significantly impair the Random Number Generator (RNG) of the processor. They did this by modifying the silicon that made up actual transistor. Their modification is completely undetectable without a Scanning Electron Microscope (SEM) and a known good chip to authenticate against. If the security of the RNG is compromised then everything generated from it is also compromised, for example, private encryption keys.

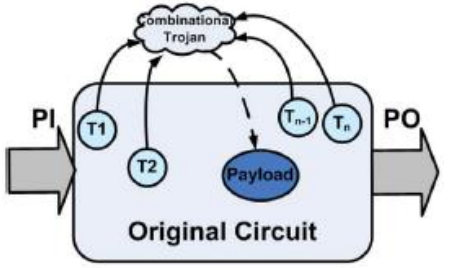


Figure 2 Combinational Trojan model

An intelligent adversary is expected to hide such tampering with an IC's behavior in a way that makes it extremely difficult to detect with conventional post-manufacturing testing. Intuitively, it means that the adversary would ensure that such tampering is manifested or triggered under very rare conditions at the internal nodes, which are unlikely to arise during testing but can occur during long hours of field operation.

Figure 2 and Figure 3 show general models of combinational and sequential Trojans, respectively. These abstract models of Trojans are useful for studying the space of possible Trojans, and, similar to fault models, help in test vector generation for Trojan detection.

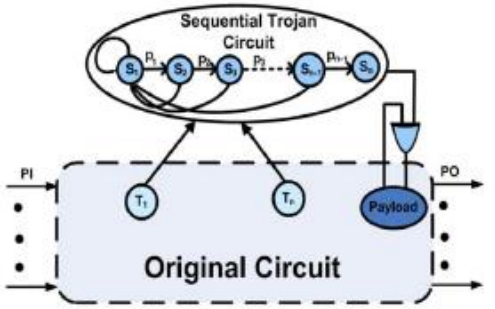


Figure 3 Sequential Trojan Model

Experiment Set-up: Configuration

1. The instruments needed for this experiment are the HaHa Board, a USB A to B cable, a USB Blaster, and a computer.
2. The software needed is Quartus, version 15 or higher.
3. Refer to the HaHa User Manual to see the steps of configuring the Altera MAX 10 FPGA.

Experiment Set-up: Instructions

In this experiment, you will need to implement a DES (Data Encryption Standard) into the Altera MAX 10 FPGA, and then hack it by inserting two kinds of hardware Trojan into it.

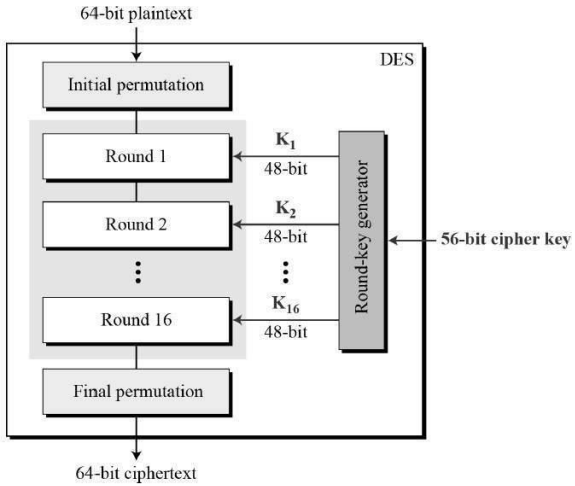


Figure 4 DES encryption illustration.

Part I: Implement a DES

The DES is a symmetric-key algorithm for encrypting electronic data. Although it is now considered insecure, it was highly influential in the advancement of modern cryptography. It uses 16 round Feistel structure. The block size is 64-bit, of which DES has an effective key length of 56 bits since 8 of the 64 bits of the key are not used by the encryption algorithm. The encryption steps are illustrated in Figure 4.

Download the 12 Verilog files from Canvas related to the DES implementation on the FPGA. If you prefer VHDL, you can use any open-source module online. Create a new Quartus project called `expt4_1` with top module called `des`, which is located in `des.v` Verilog file. Refer to Figure 5. There are two modules inside this file: des and `des_o`. `des_o` contains the DES implementation, and `des` is the wrapper top module that instantiates `des_o`. Plaintext and key are given in des. There is a RAM module called ram1 instantiated, which is a 64-bit wide, 32-word deep RAM. It will store all the encryption results of the 16 rounds. All the submodules are given except the `ram1` module.

Import these Verilog files by going to **Project** > **Add/Remove Files in Project…** window. Click on **…** next to **File name** text box. Locate the downloaded files, select them all, and click **Open**. Then, click **OK**. You should create the `ram1` module from 'RAM: 1-PORT' of the installed IP with exactly the same name 'ram1' with exactly the same size. Make sure it can be captured by the In-System Memory Content Editor in the last step of Parameter Settings. Refer to the Appendix: RAM Instantiation on Quartus software for visual instructions of instantiating this module. Module `divi` is added to slow down the clock so that an error will not be caused due to a fast clock. You can even make it slower.

Inside your Quartus project directory, locate and open `expt4_1.qsf` file. Add the following lines to the end of that file:

```
set_location_assignment PIN_88 -to clk_in
set_location_assignment PIN_30 -to rst
```

These lines maps `clk_in` and `rst` ports from the `des` top module to pins 88 and 30 on the FPGA. Alternatively, you can do this mapping using the 'Pin Assignment' tool in Quartus.

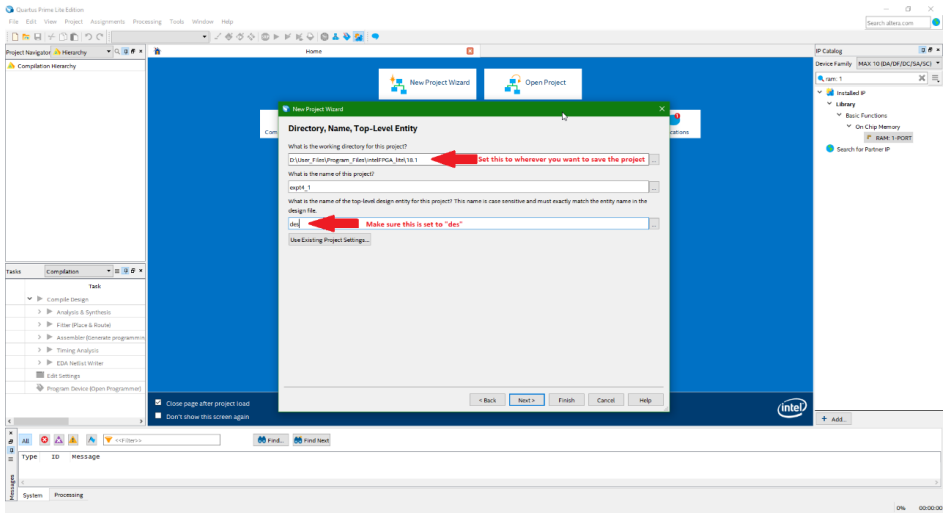


Figure 5 New Project Wizard

Part II: Insert a combinational Trojan

Insert a combinational Trojan into the DES circuit you implemented in Part I. The trigger condition of the Trojan is when the output of the F function (Figure 6) satisfies for some value of the least significant 4 bits (see next section). When the Trojan is triggered, the LSB of the input key (NOT round keys) for the DES is inverted. When the trigger condition is not true, the key becomes the original input key.

Part III Insert a sequential Trojan

Insert a sequential Trojan into the DES circuit you implemented in Part I. Use the same clock as the DES circuit. The trigger condition of the Trojan is when the least significant 2 bits of the F function output in order go through some order of three values at the negative edge of the clock (see next section). After the Trojan is triggered, the LSB of the input key (NOT round keys) for the DES will always be inverted.

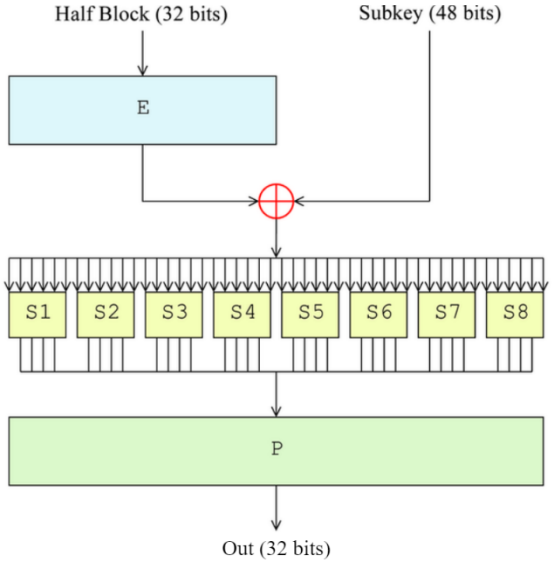


Figure 6 Feistel function (F function) of DES

Measurement, Calculation, and Question

Answer the following questions.

Part I: Implement a DES

1) Use the plaintext and key provided and add them in the des.v Verilog file:

   `desIn` = 64'hA42F891BD376CE05
   `key64` = 64'h0123456789ABCDEF

   If you choose to use other open source VHDL code, also use the same plaintext and key. Store all the encryption results for the 16 rounds in an implemented RAM and show them in In-System Memory Content Editor ('Editor window' below). Turn in a screenshot.
2) Which module is creating the key for each round? By how?
3) Which module is doing Feistel function?
4) How many Logic Elements are used?

Part II: Insert a combinational Trojan

1) Turn in your code when the trigger condition is 4'b0110. Only turn in the Verilog files that have been changed and Verilog files (if there are) that are created by you.
   a. When the trigger condition is 4'b0110, will the Trojan be triggered? How many times is it triggered in the 16 rounds? Turn in a screenshot of the Editor window.
2) When the condition is 4'b1001, repeat answering question 1) again.
3) When the condition is 4'b1010, repeat answering question 1) again.

Part III: Insert a sequential Trojan

1) Turn in your code when the trigger condition is 2'b01→2'b10→2'b11. Only turn in the Verilog files that have been changed and Verilog files (if there are) that are created by you.
   a. How many states are needed in total? How many additional registers have you implemented?
   b. When the trigger condition is 2'b01→2'b10→2'b11, will the Trojan be triggered? How many times is it triggered in the 16 rounds? Turn in a screenshot of the Editor window.
2) When the condition is 2'b01→2'b10→2'b11, repeat answering question 1) again.
3) When the condition is 2'b11→2'b01→2'b00, repeat answering question 1) again.

Optional Follow-up

Part IV: Change the trigger condition

Change the trigger condition of the Trojan you inserted in Part II to be acceleration value from the accelerometer.

Use the hex file you downloaded for Experiment 3 (Bus Snooping Attack), make the Atmel AVR microcontroller to run the code when the FPGA is running DES. The microcontroller will keep sending acceleration data to the FPGA through the 8 interconnections.

1) If the longitudinal direction of header P3 is x and the longitudinal direction of header P4 is y, on which direction is the sensor sensing the acceleration?
2) Add an 8-bit input for the DES to accept the acceleration data. When the board is more than 45-degree up tilted (in the right direction), the Trojan will be triggered, and the payload is the same with the Trojan in Part II. Turn in your Verilog description (or VHDL code). No screenshot is needed.

☞ Lab Report Guidelines and Demonstration

Deliverables:

1. In your report, give answers to ALL the questions.
2. In part I, give a screenshot after compiling the DES code.
3. Give a photo, or a screenshot to prove your DES works well.
4. Give all the code that is required.
5. Attach screenshots as required.

Demonstration:

Please take videos of the demonstration and include them to your submission.

1. For part I, show your encryption result.
2. For part II, show the result when the combinational Trojan is triggered.
3. For part III, show the result when the sequential Trojan is triggered.

References and Further Reading

[1] http://securityaffairs.co/wordpress/17875/hacking/undetectable-hardware-trojan-reality.html

[2] Bhunia, Swarup, et al. "Hardware Trojan attacks: threat analysis and countermeasures." Proceedings of the IEEE 102.8 (2014): 1229-1247.

[3] http://www.emvlab.org/descalc/

[4] https://www.pantechsolutions.net/matlab-code-for-des-algorithm

[5] http://www.tutorialspoint.com/cryptography/data_encryption_standard.htm

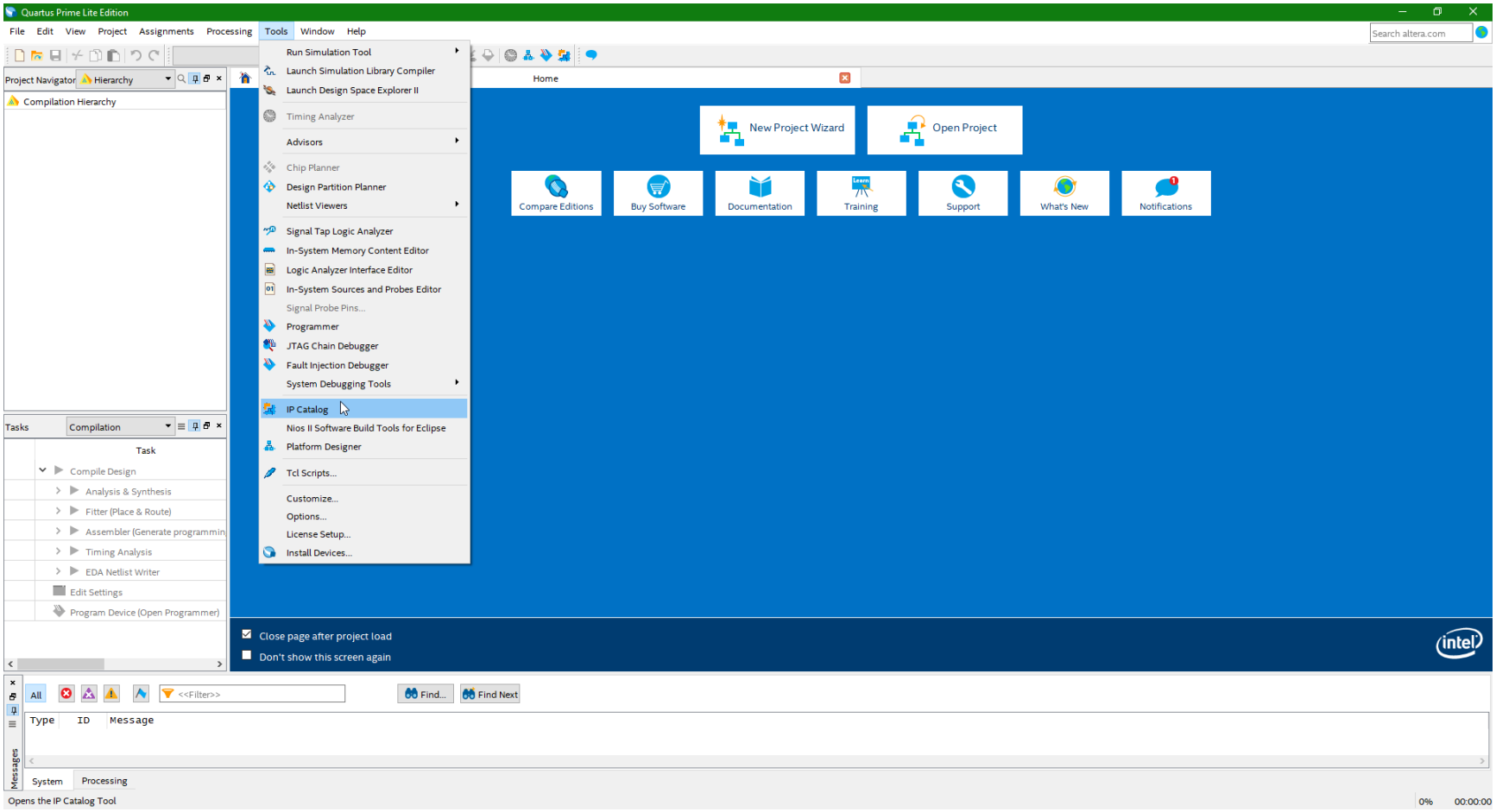
[6] https://en.wikipedia.org/wiki/Data_Encryption_Standard

[7] https://en.wikipedia.org/wiki/Feistel_cipher

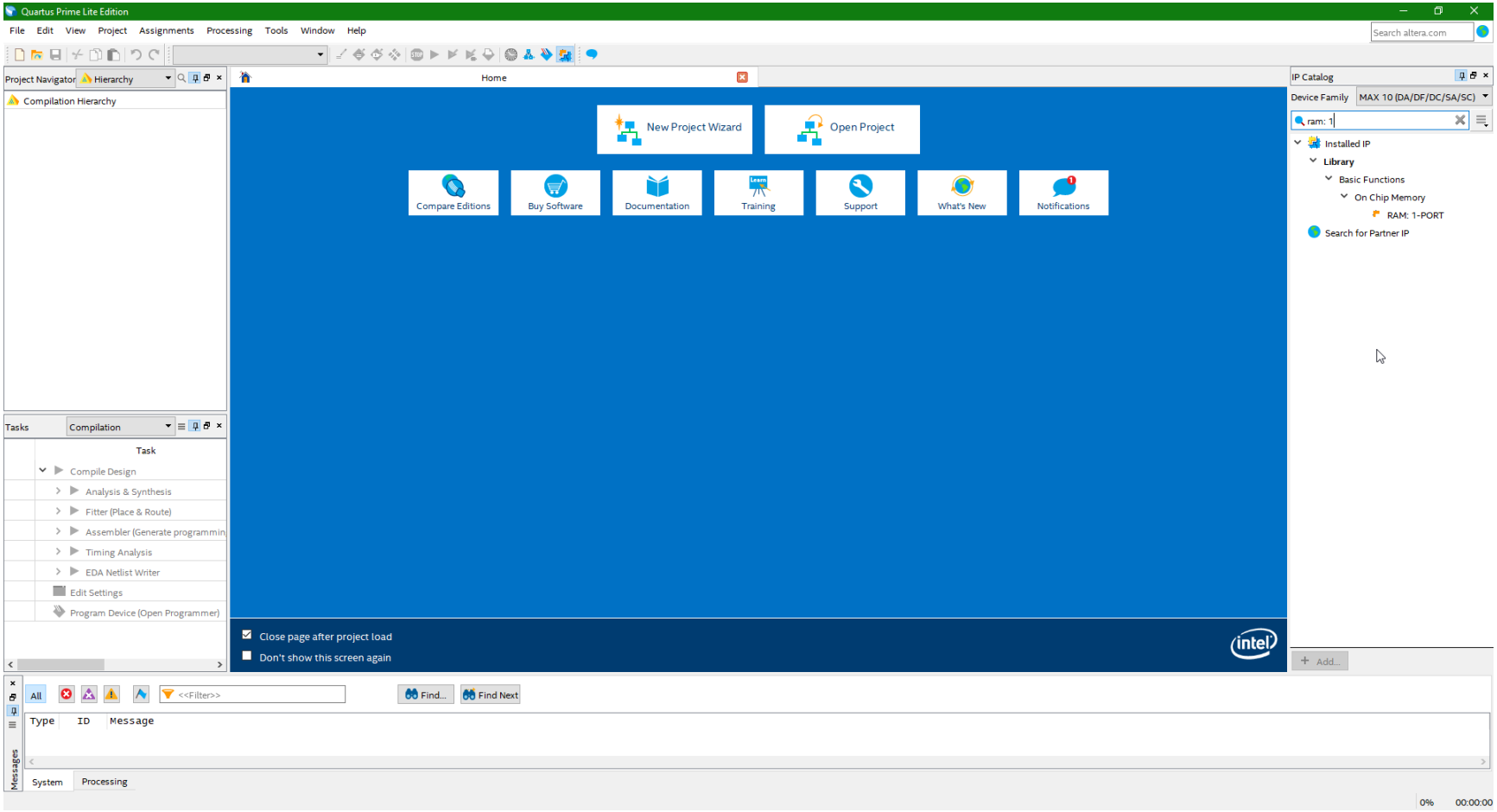
Appendix: RAM Instantiation on Quartus software

The following are the steps to instantiate the RAM module on the Quartus software:

1. Go to **Tools** > **IP Catalog**



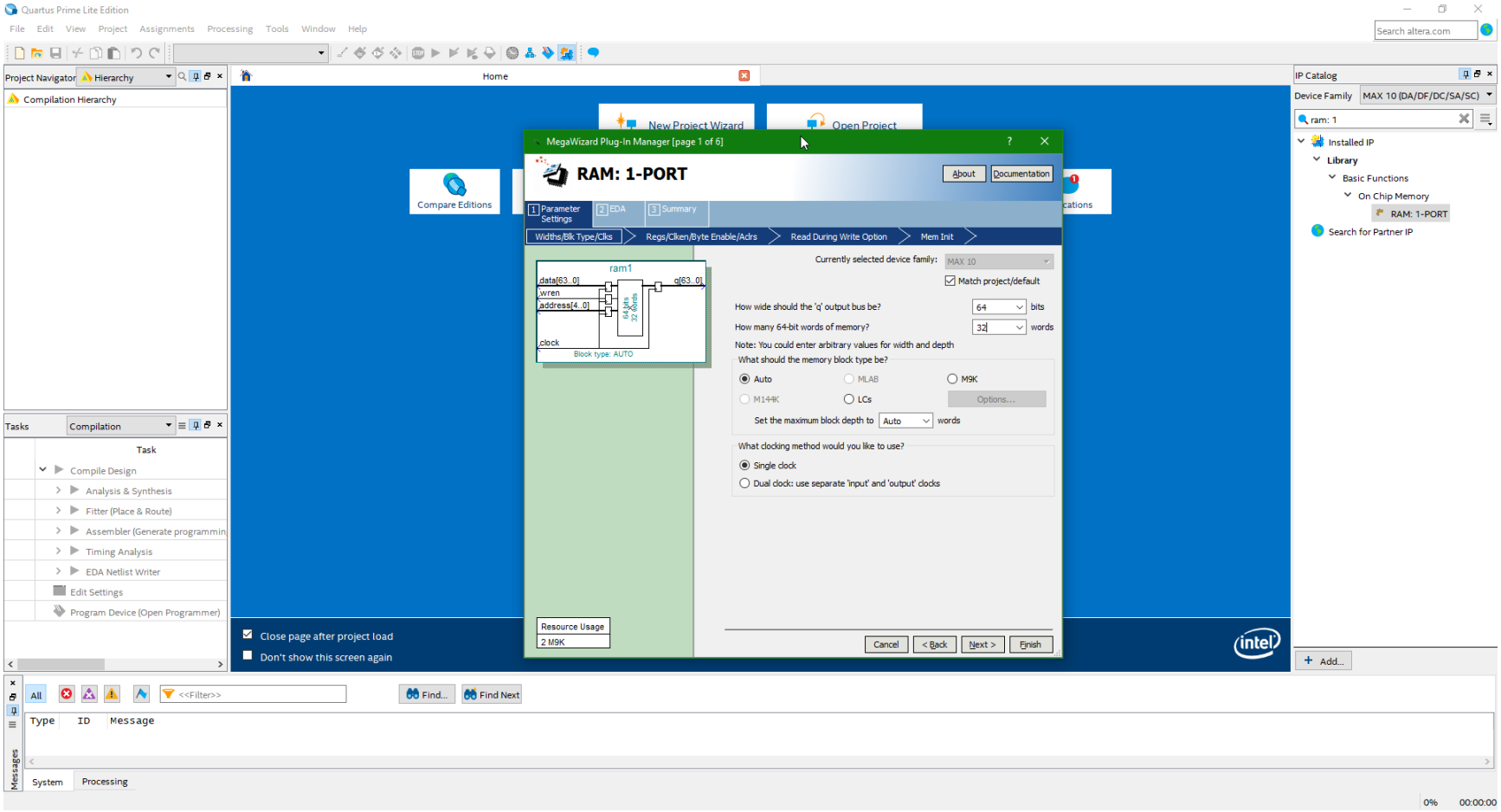
2. The IP Catalog sidebar will show up on the right side of the window. Search for ‘ram: 1’ and click on ‘RAM: 1-PORT’.



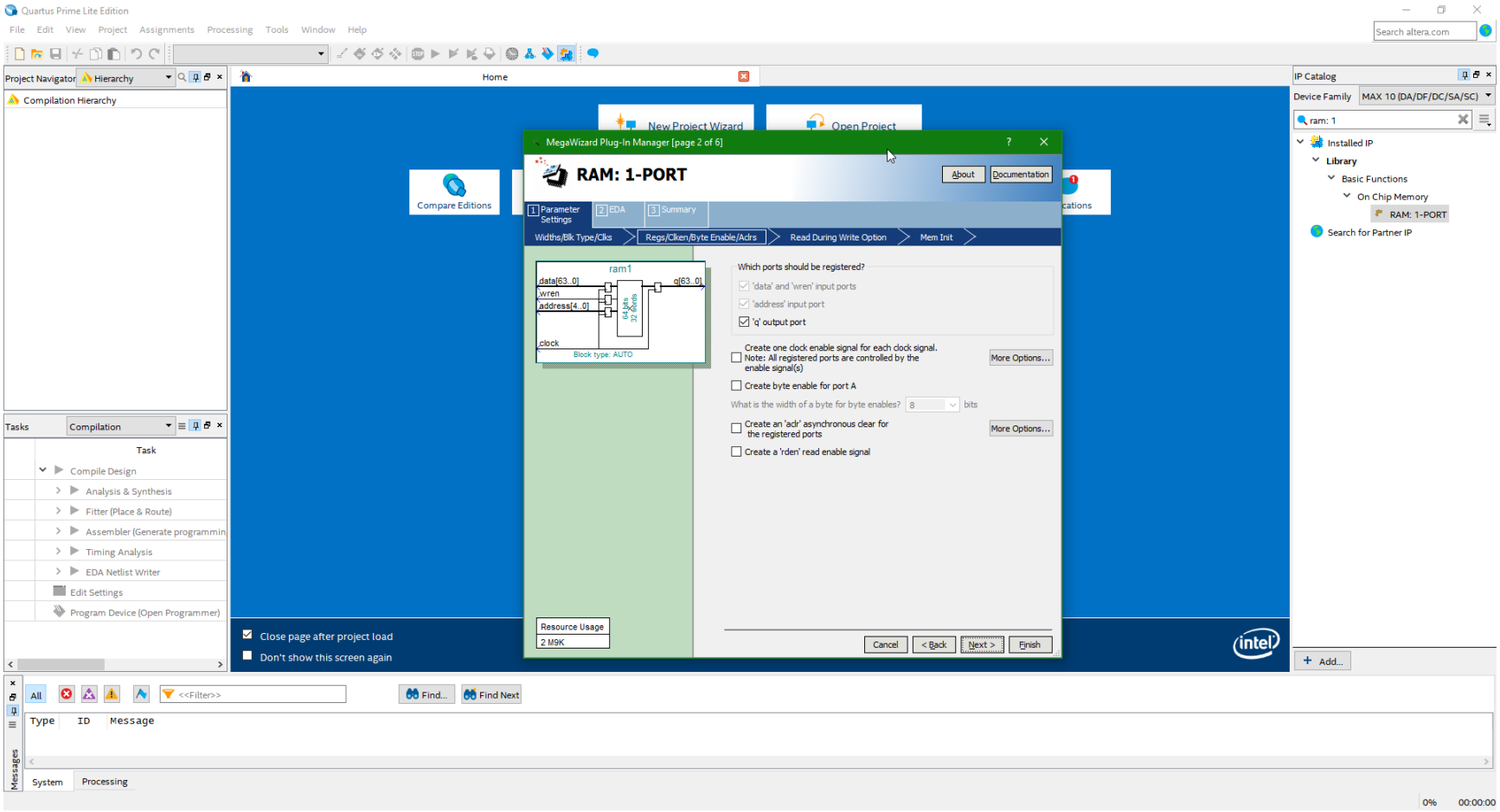
3. The MegaWizard Plug-In Manager window will open. Set the following then click **Next**:

   How wide should the 'q' output bus be? **64 bits**

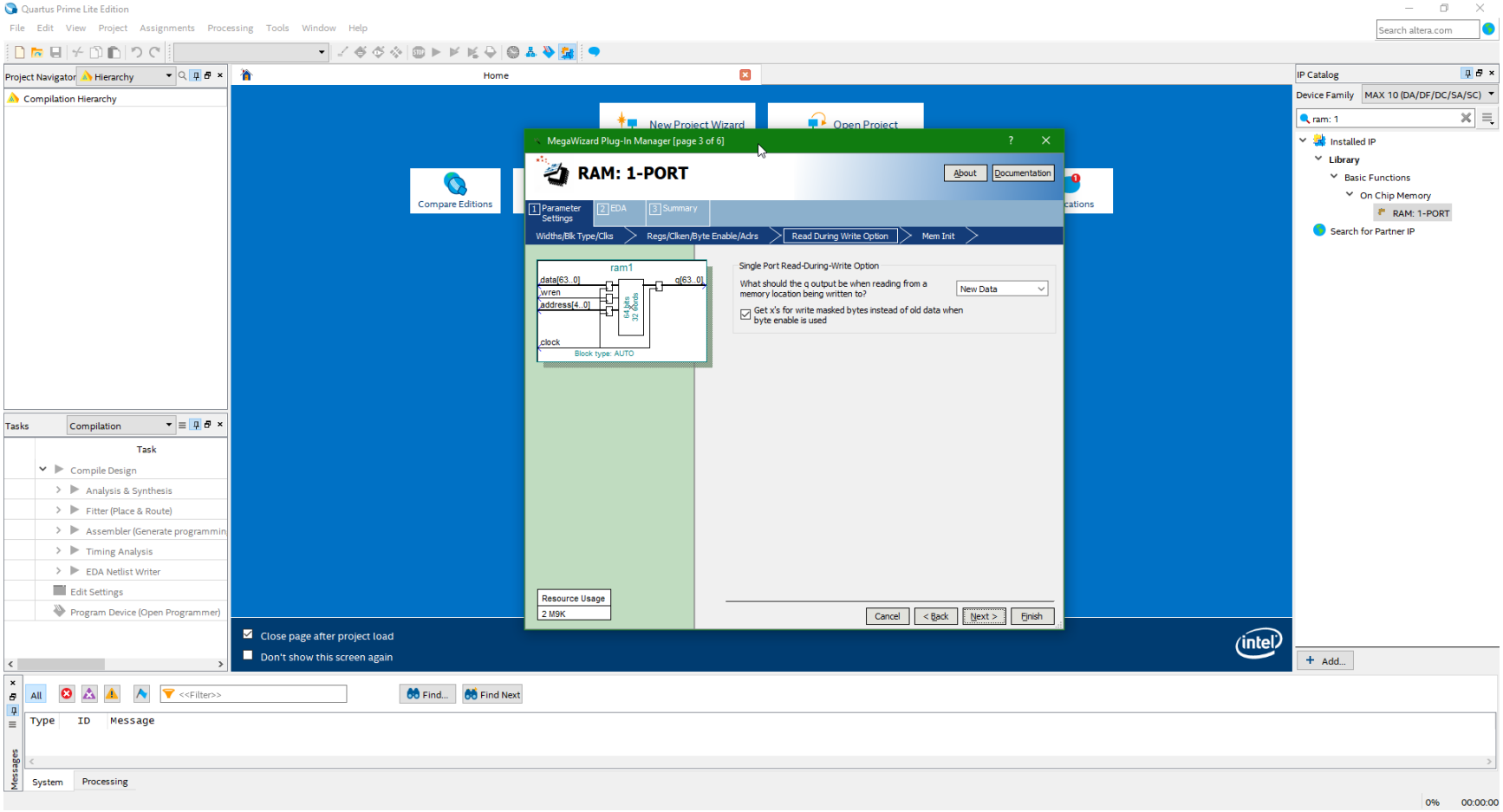
   How many 64-bit words of memory? **32 words**



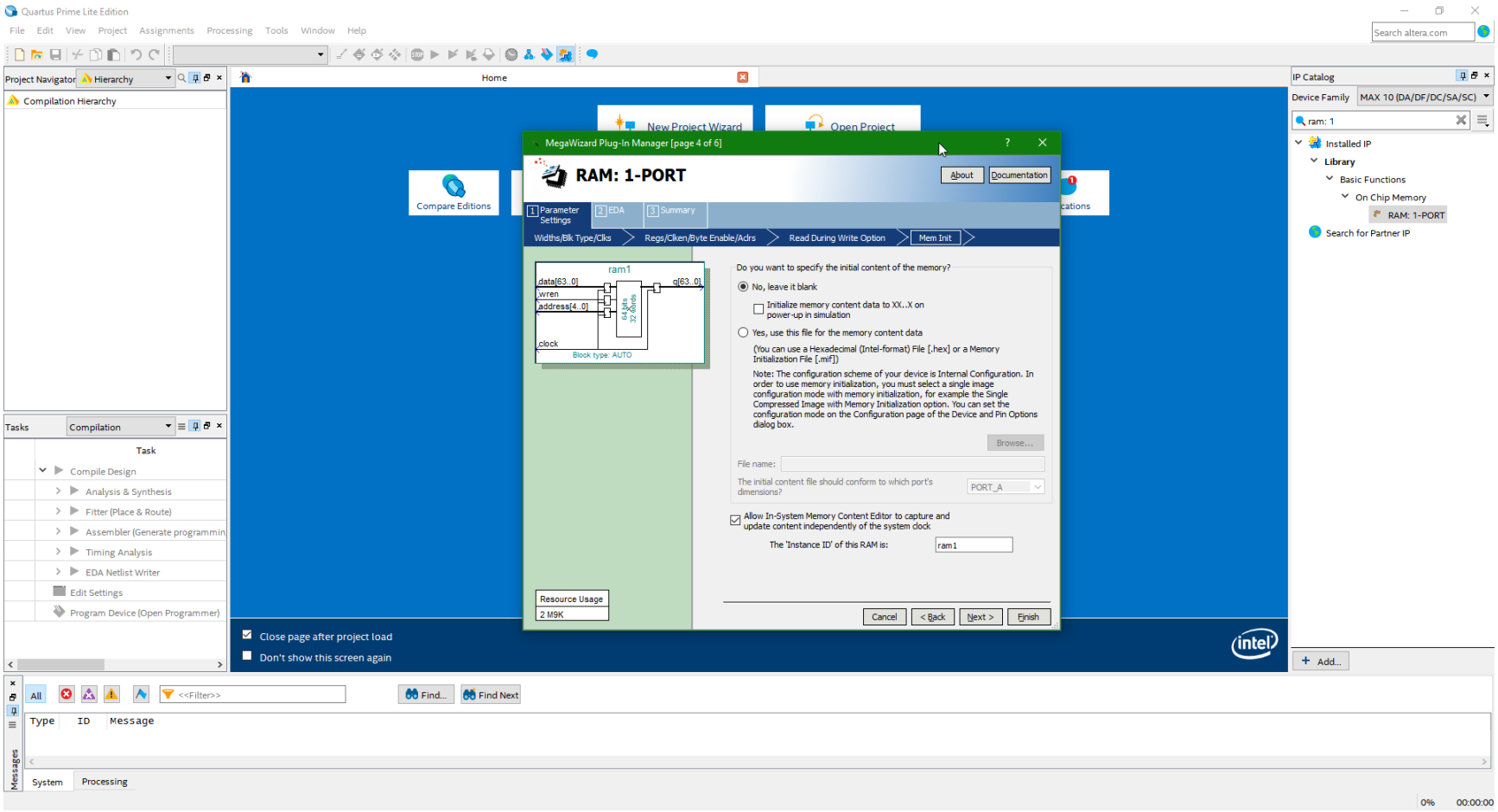
4. Check the box for **'q' output port** and click **Next**.



5. Leave the settings as is and click **Next**.

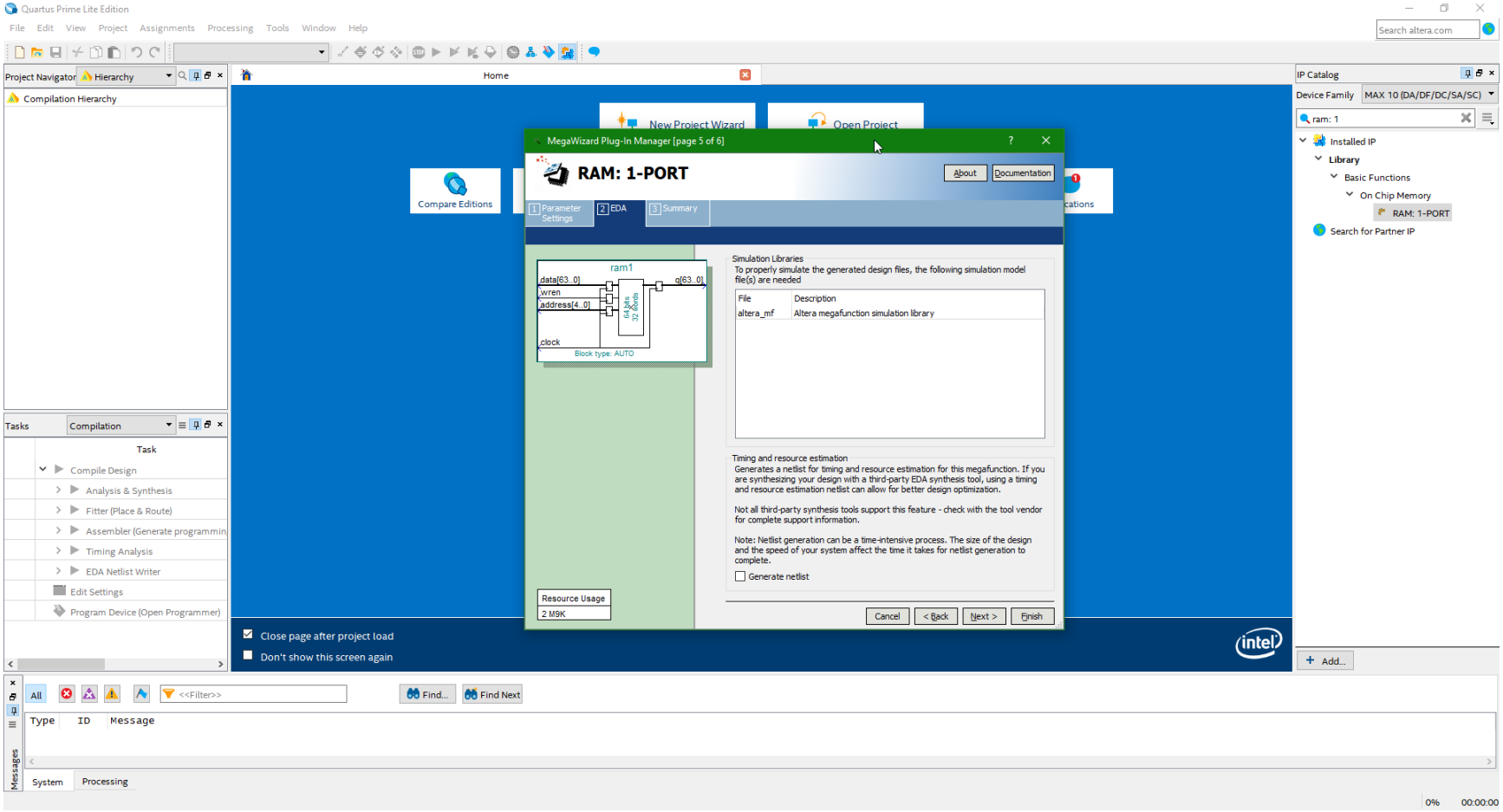


6. Check the box for **Allow In-System Memory Content Editor to capture and update content independently of the system clock**.
   The 'Instance ID' of this RAM is: ram1



Then, click **Next**.

7. Leave the settings as is and click **Next**.



8. Check the boxes for `ram1_inst.v` and `ram1_bb.v`. Then, click **Finish**.