APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL VERSION) Project 14 The use of block RAM IP cores based on the PL side

Almost all FPGA chips have a memory unit BLOCK RAM, and ZYNQ is no exception, and this article will focus on the use of BRAM IP cores

(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

Microprocessors such as single-chip microcomputers and STM32 have RAM to cache variables, FPGAs are no exception, and we also need to frequently cache some data (such as intermediate variables, temporary files of images, or matrix operation templates, etc.) in the design process of FPGAs, so how do we achieve data staging in FPGA design, there are many methods. We can store this data in DDR and SDRAM outside the chip, or inside the FPGA chip.

There are two ways to realize the data caching function inside the FPGA chip, one is hardware-based BLOCK RAM (it can be understood that the FPGA puts a RAM chip inside, which is equivalent to a physical storage module), and the other is DRAM, (Distributed RAM) spliced out by the logical resources of the FPGA

DRAM and BRAM have their own advantages and disadvantages, the most obvious difference is that BRAM does not occupy FPGA logic resources, DRAM needs to occupy LUTs and lookup tables, BRAM is complex to use (clock synchronization is required and some control signals are given), DRAM only needs to simply call the lookup table (can be stored and read without clock, equivalent to combinatorial logic), BRAM can store more

data, DRAM is subject to the large idle resources of FPGA logic resources (usually only a small amount of data such as a few bytes when DRAM is recommended), when there is more data such as caching a row of image data or storing convolution templates will use BRAM (several K to hundreds of K), when storing very large data such as the entire image (several M or even tens of M), it is recommended to use an external memory chip (SDRAM or DDR) Of course, how to use the method is not fixed, as long as the program of the project can run through, Even if you hang a memory stick outside, it's fine.

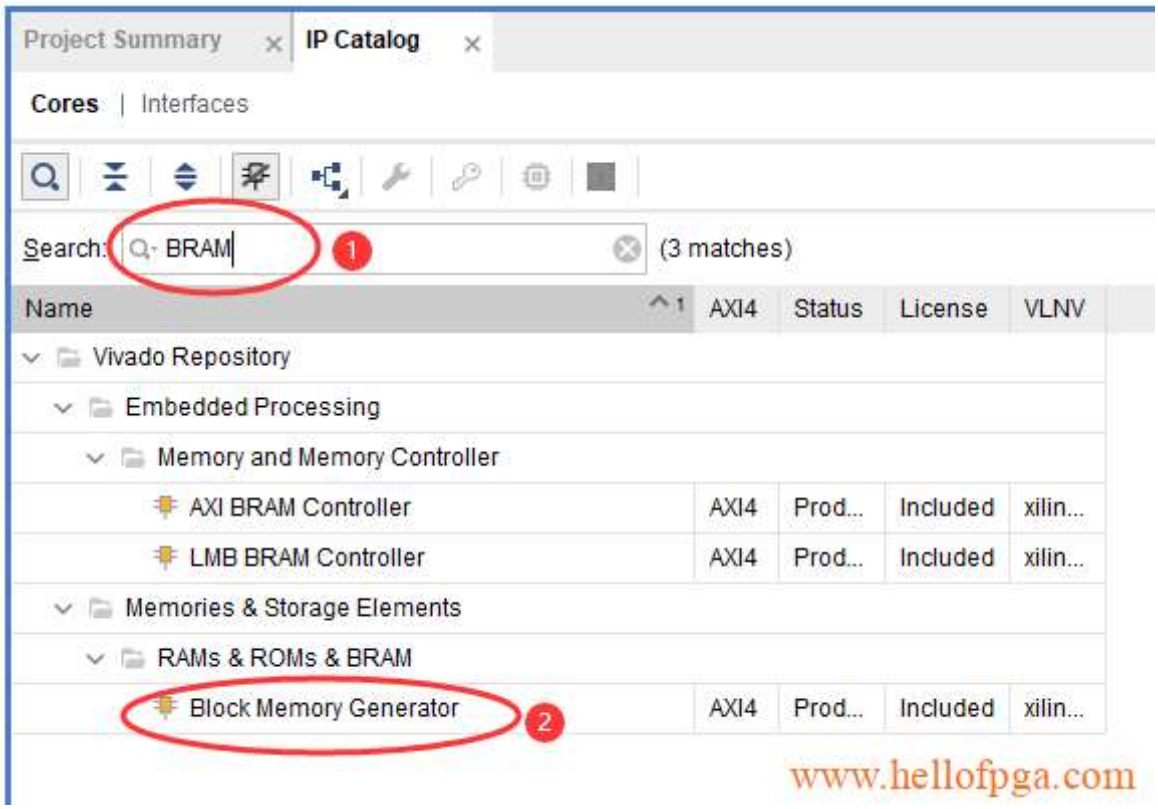
We can use BRAM to generate many different forms of RAM, such as ROM FIFO, etc.

Project creation

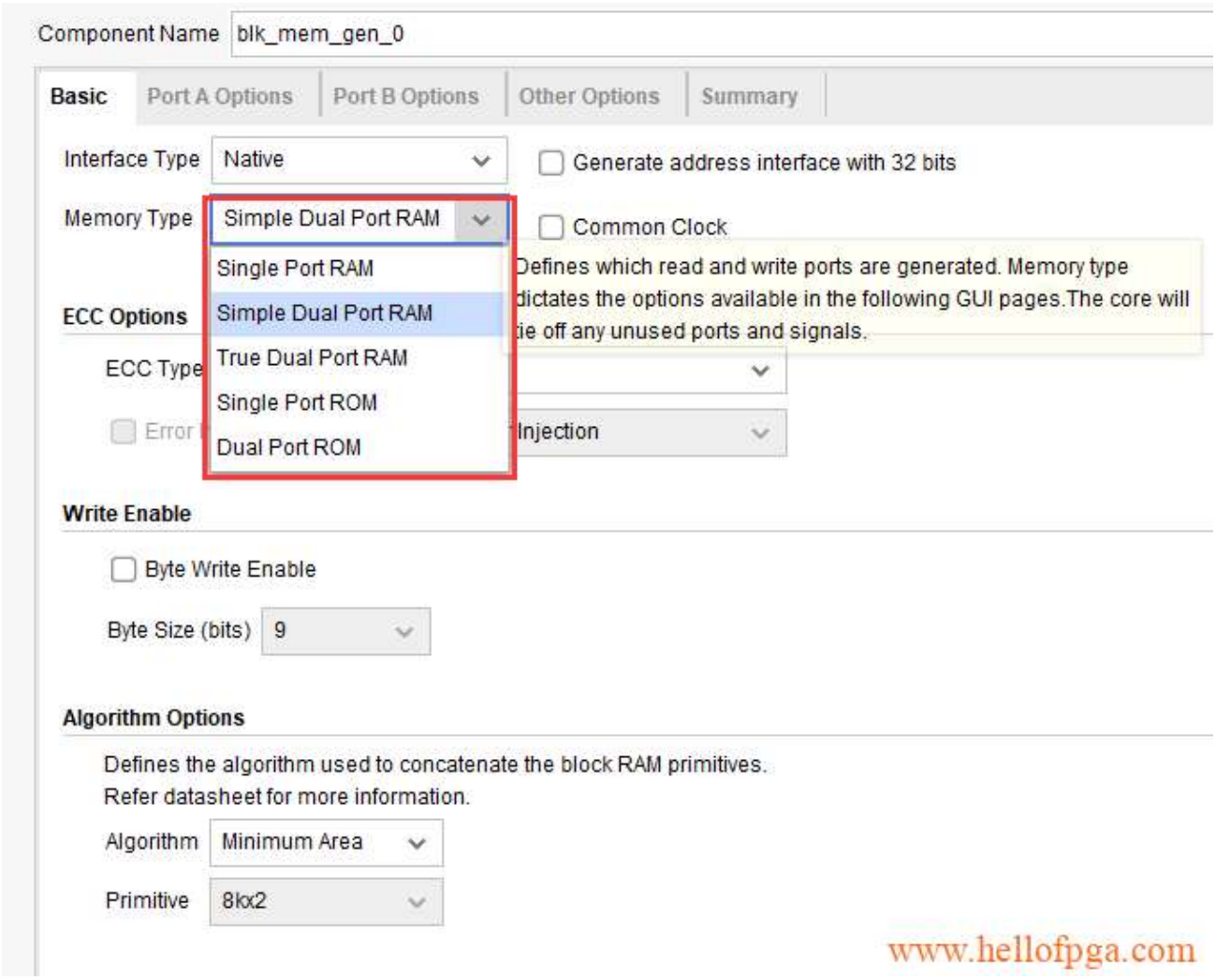
1) First create a project normally (you can refer to the previous example),

2) Add PLL IP cores. Check "IP Catalog" in the Vivado software

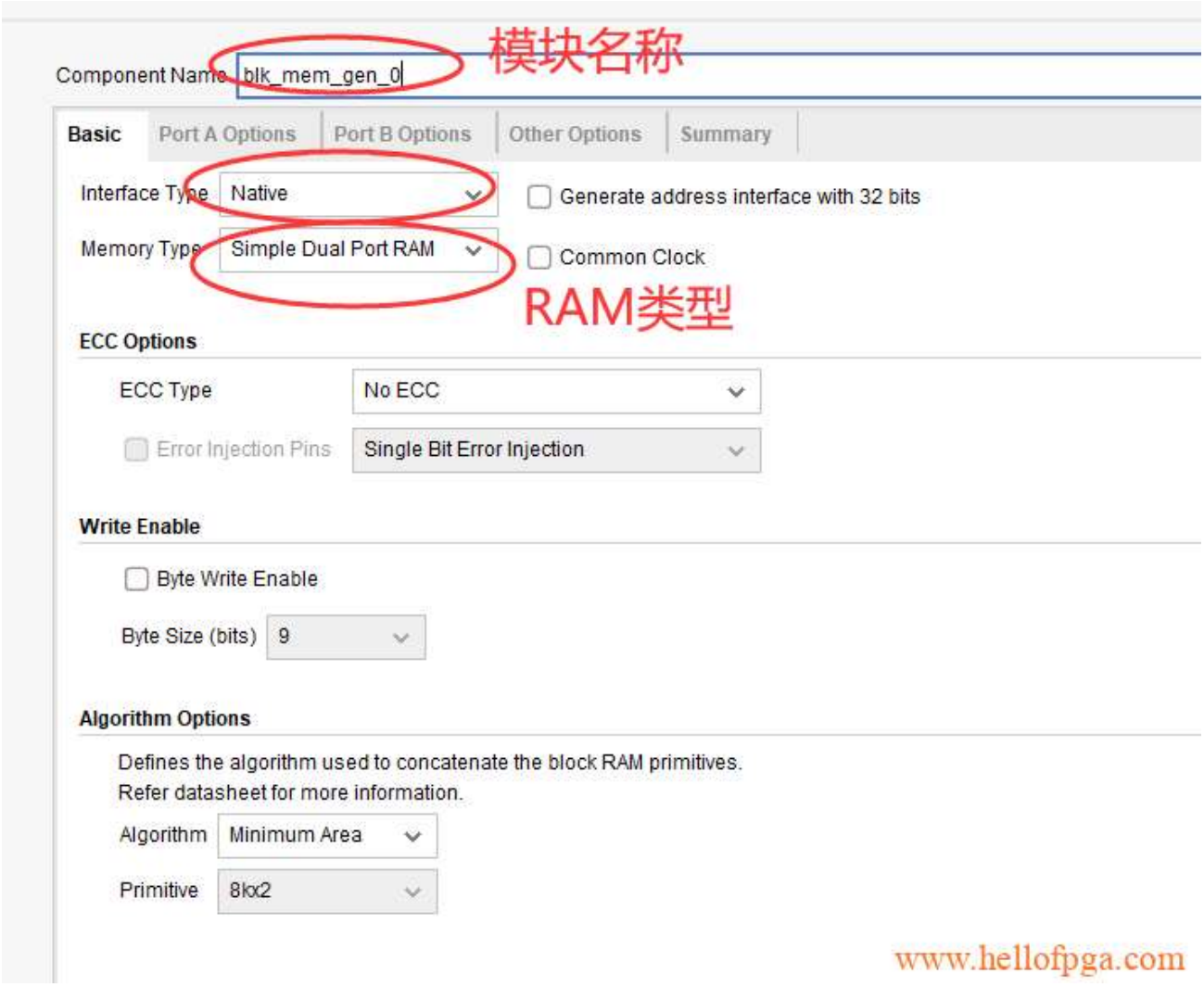
Search for BRAM in the pop-up window and double-click Block Me in the search results. Enter the Clock Module Setup Wizard



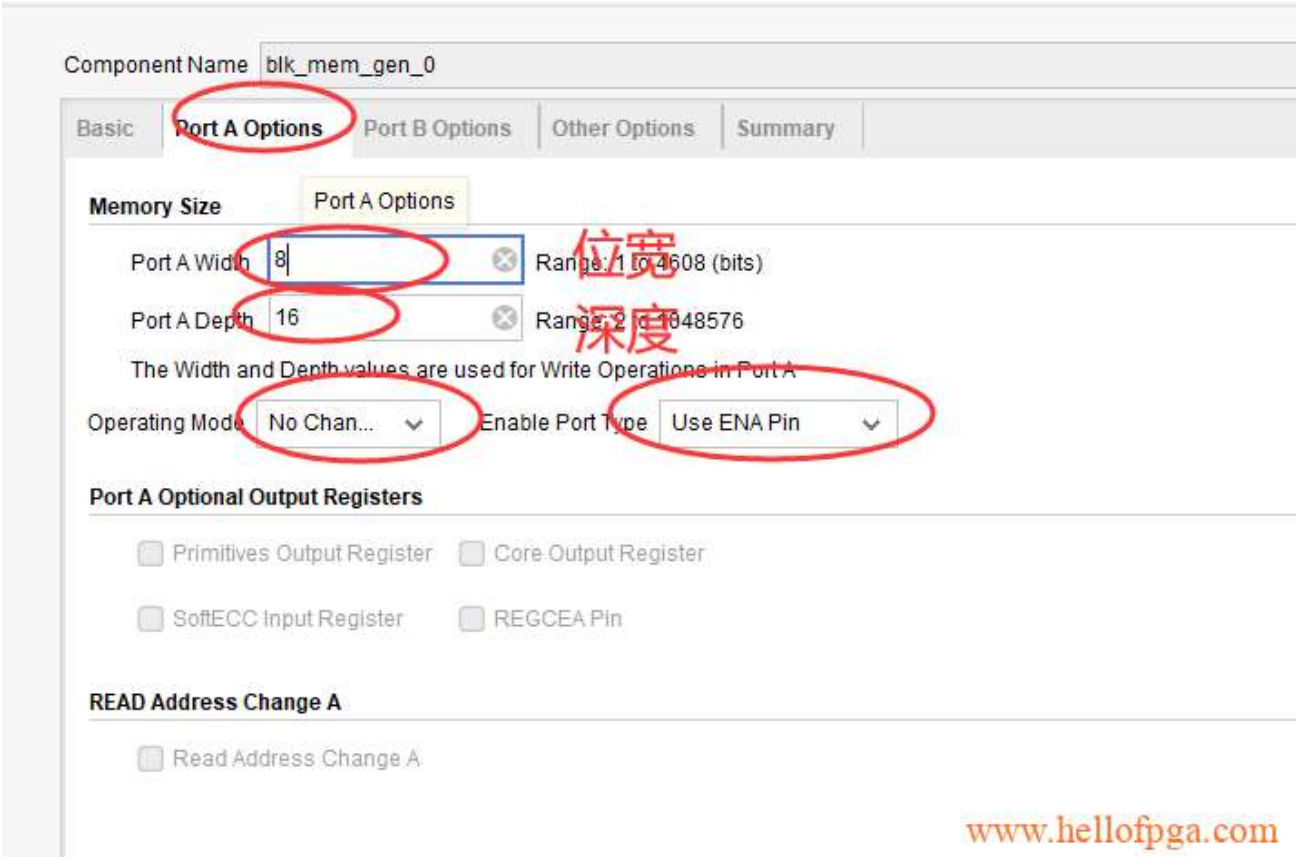
The BRAM inside Xilinx 7 series FPGAs can be configured as **true dual-port** RAM (True Dual-Port ram, which can be responsible for reading and writing at the same time), **Simple Dual-Port RAM** (one can only be responsible for reading and one can only be responsible for writing), **single-port RAM** (only one port), and ROM, as shown in the figure below.



The various types of RAM are used similarly, and we will choose a **simple dual-port RAM** for the demonstration.

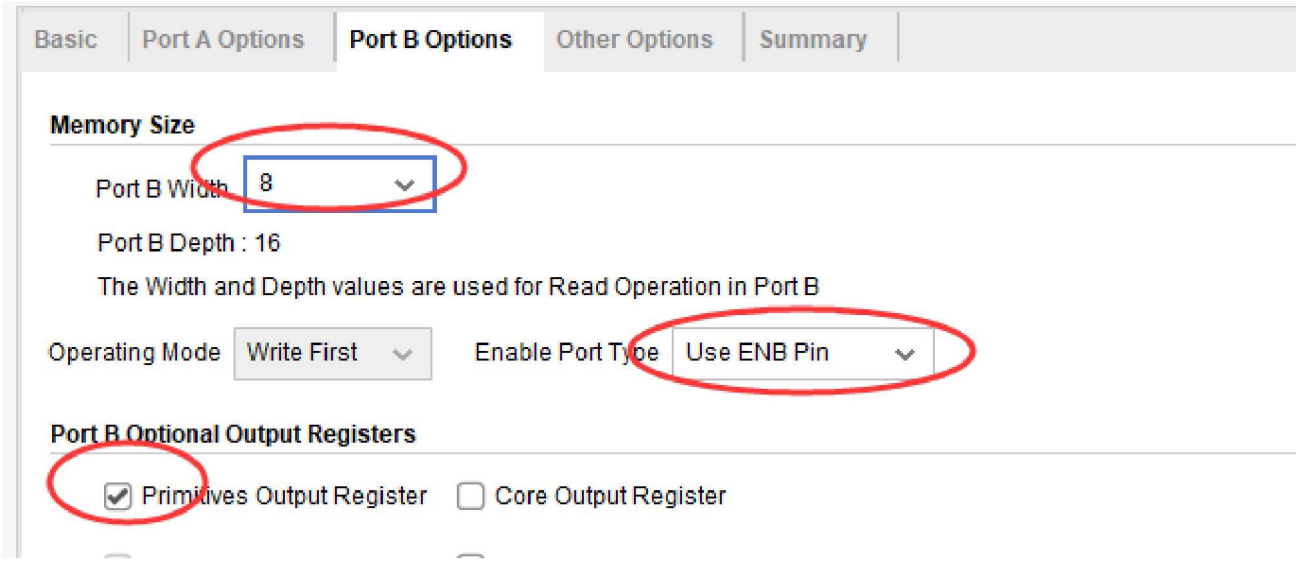


Next, set up port A (here we create a RAM with 8-bit width and 16-bit depth) The depth represents how many 8-bit wide RAM cells there are, and 16 is used here for demonstration purposes



There are three modes for the Operating Mode option, read-first, write-first, and hold-and-hold, because the simple dual port RAM we use belongs to two ports, one for writing and one for reading, so it's good to select the hold mode here

Next is the B port setting, the bit width and depth system will match the A port by default, and there is no need to modify it here. There is an output register option here Primitives Output Register needs to pay attention to the system is open by default, so that the data read out by the B port of BRAM will pass through a register, in the design of the high-speed pipeline, you can improve the timing performance, so that the data and output clock are synchronized, but this will delay the data output by BRAM by a beat, so you need to pay special attention (mentioned below), and then select OK to generate the IP module

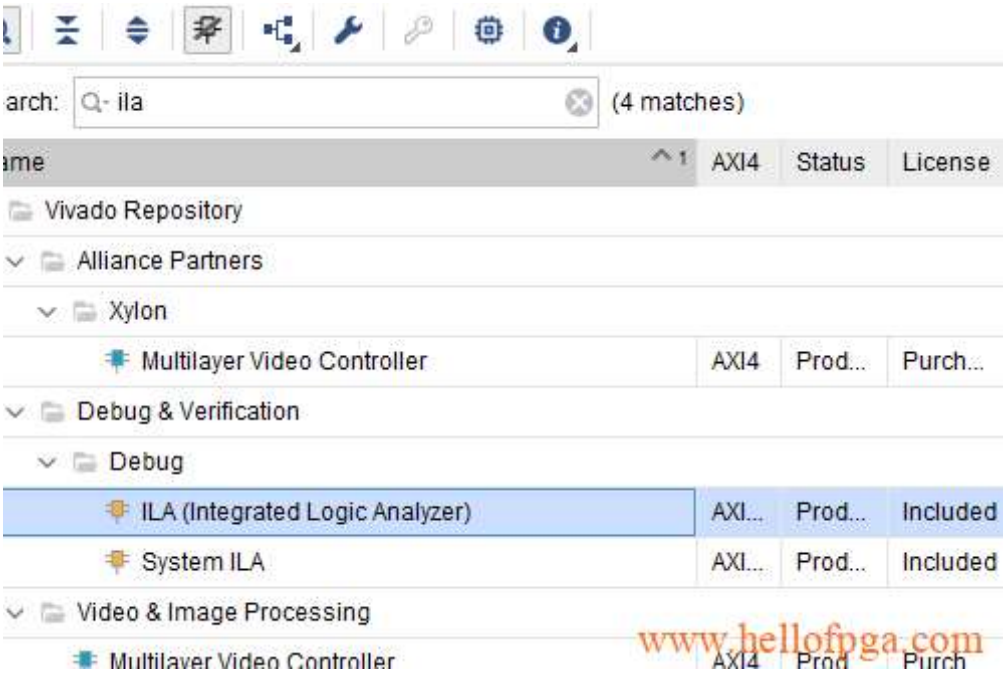


Among the other options, you can import the initialization file to assign initial values to RAM, which we don't need here, so leave it blank

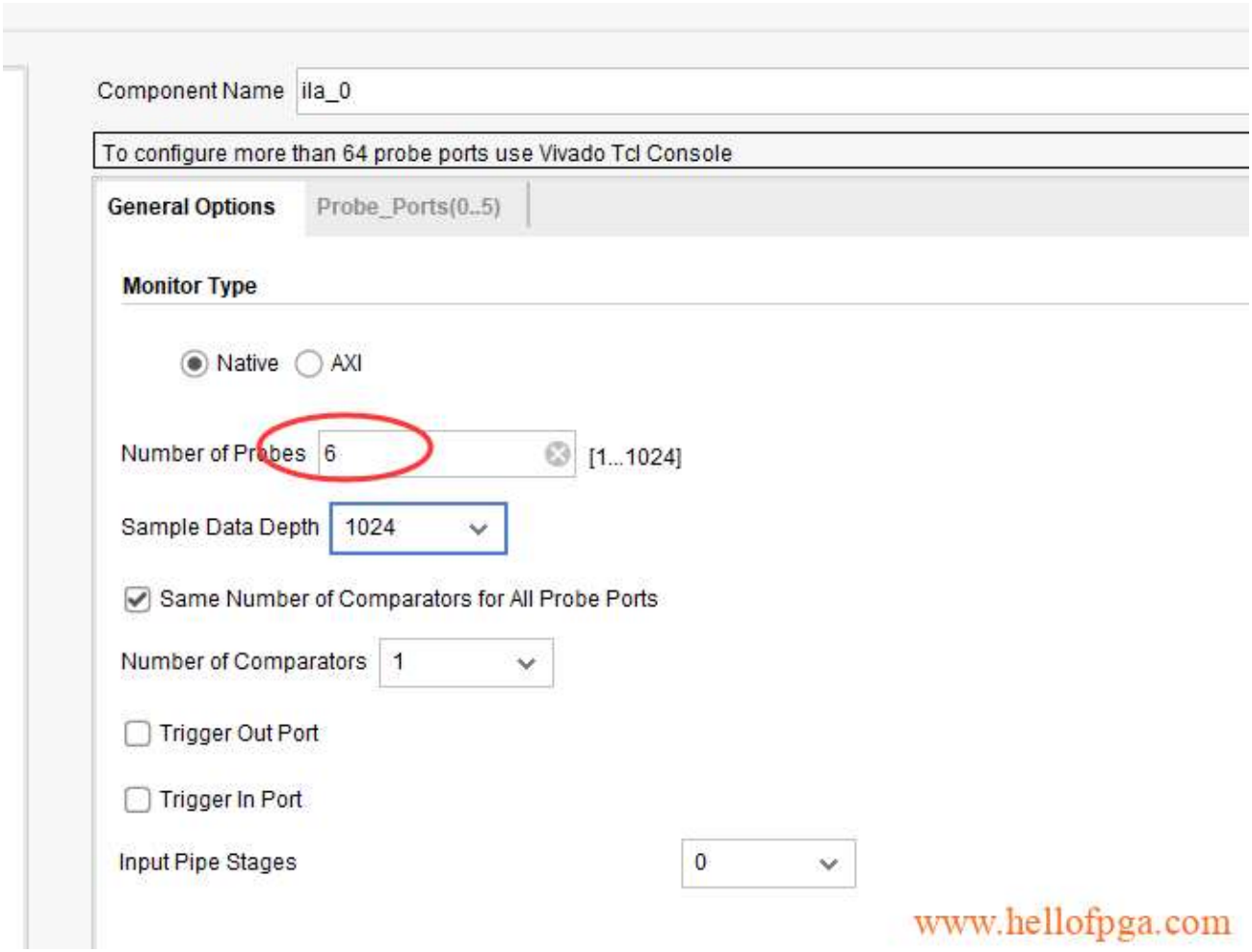
After that, click OK to save and generate BRAM resources

In order to make the call result of BRAM more intuitive, here we add an ILA module (ILA module is equivalent to the logic analyzer function inside the FPGA), and the specific use of ILA can refer to the next project (Engineering 2022)
http://www.hellofpga.com/index.php/10/12/<>/ila/

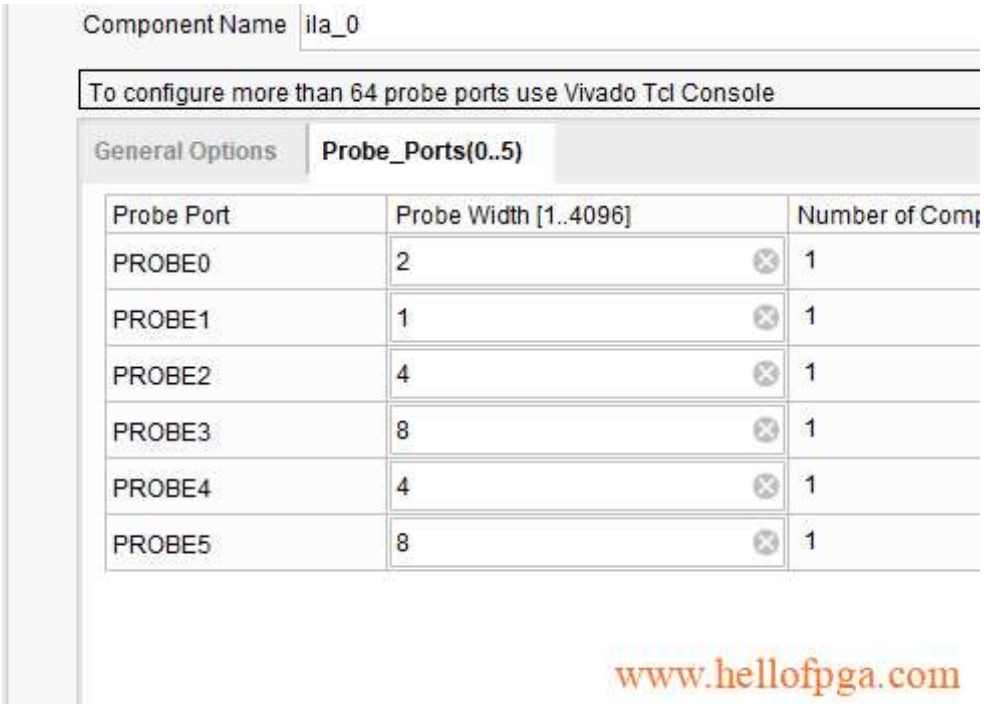
Add the ILA module to the IP menu



Modify the number of probes to 6 (here we are going to grab 6 signals)



On the second page of ILA, set the bit width of each probe separately



Then click OK to save and generate the ILA resource

programming

Next is the programming part of this experiment

The program design is divided into 3 stages

- The first stage (mode=0) initializes the registers in the system (including the address of the bram call, registers such as read and write enable) (mode 0)
- The second stage (mode=1) (the address of port A is self-increased from 0-15, and the corresponding 0-15 is stored in Bram, and after storing 16 numbers, the read and write function of port A is switched to read, that is, no data is written)
- The third stage (mode=2) (the address of port B is incremented from 0-15, and the output data of port B is read, and when the self-increment is increased to 15, the value of mode is assigned to 3)
- The fourth stage (mode=3) does not do anything Assign a value of 0 to mode Let the four stages loop up **(why loop here, because if you don't increase the loop, you haven't used ILA to start capturing the waveform at the moment of power-on, and the FPGA has completed all the read and write operations, you can't see the complete process, so you need to loop the whole process here for easy observation)**

Below is the complete procedure

```
`timescale 1ns / 1ps
module BRAM_TEST(
    input CLK
);

reg [3:0]addr_a;
reg [3:0]addr_b;
reg  wr_en_a;
reg  [1:0]mode=2'd0;
reg  [7:0]din_a;
wire  [7:0]dout_b;

always@(posedge CLK)begin
    if(mode==0)begin
        addr_a<=4'd0;
        din_a<=8'd0;
        addr_b<=4'd0;
        wr_en_a<=1'b1;
        mode<=2'd1;
    end
    else if(mode==1)begin
        if(addr_a==4'd15)begin
            mode<=2'd2;
            wr_en_a<=1'b0;
        end
        else begin
            addr_a<=addr_a+1'b1;
            din_a<=din_a+1'b1;
        end
    end
    else if(mode==2)begin
        if(addr_b==4'd15)begin
            mode<=2'd3;
        end
        else addr_b<=addr_b+1'b1;
    end
    else mode<=2'd0;
end


blk_mem_gen_0 u_bram (
  .clka(CLK),
```

```
    .ena(1'b1),
    .wea(wr_en_a),
    .addra(addr_a),
    .dina(din_a),
    .clkb(CLK),
    .enb(1'b1),
    .addrb(addr_b),
    .doutb(dout_b)
);

ila_0 ila_u (
        .clk(CLK),
        .probe0(mode),
        .probe1(wr_en_a),
        .probe2(addr_a),
        .probe3(din_a),
        .probe4(addr_b),
        .probe5(dout_b)
);

endmodule
```

There is only one always block in this program, which performs the functions of the above 4 stages according to the different values of the mode register, of which the values of din_a and addr_a are always the same; The content that represents the deposit of each address is the address number of the address, that is, the number between 0-15

Add a constraint file

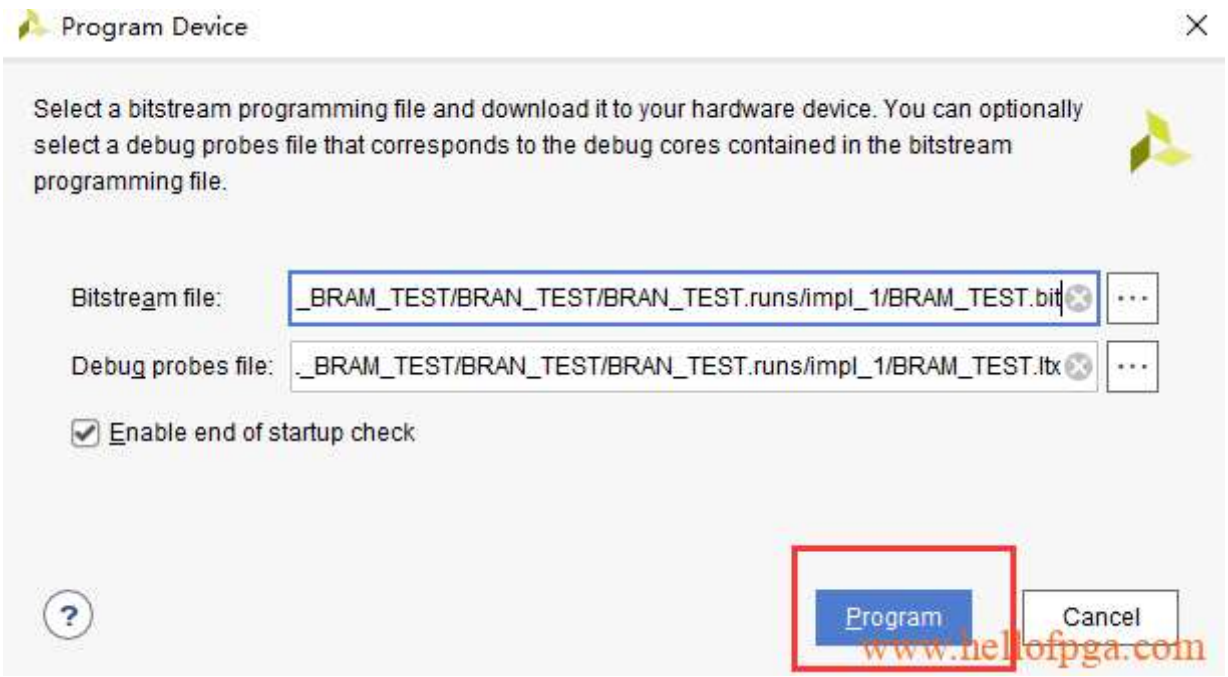
```
set_property PACKAGE_PIN M19 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
```

After that, it is compiled and synthesized

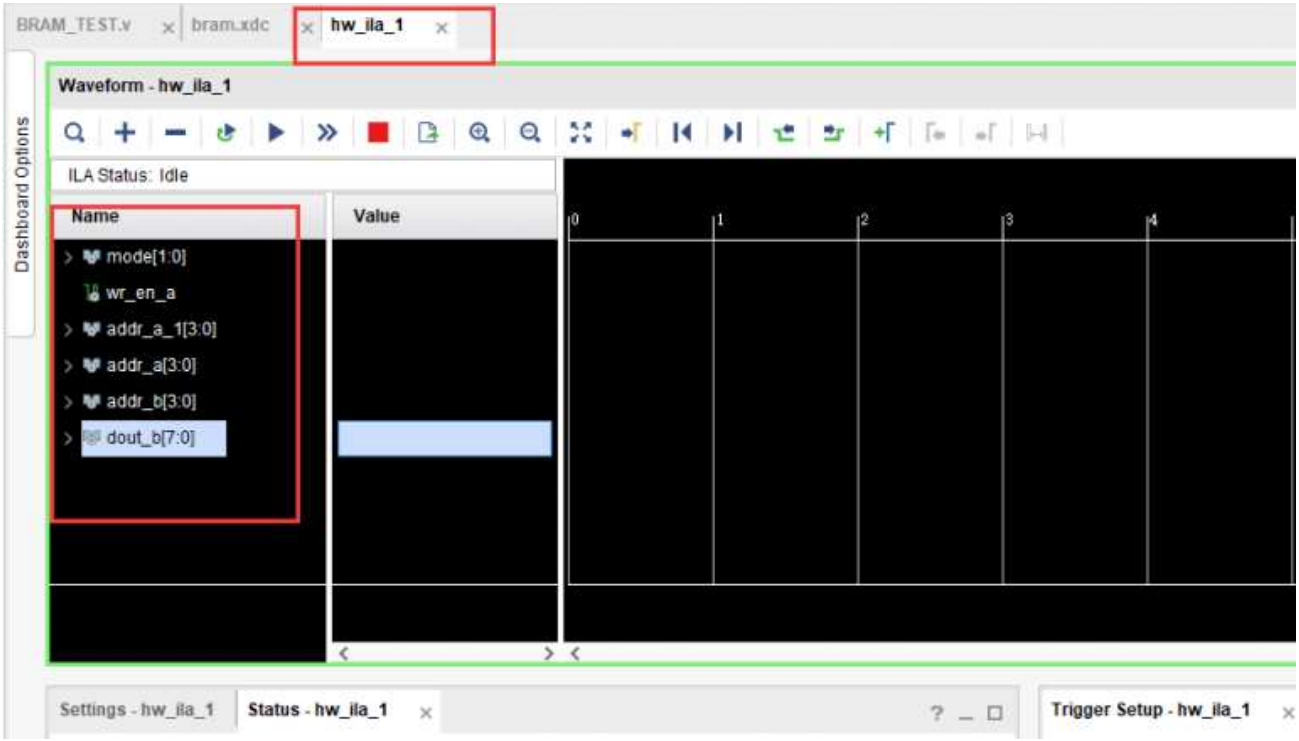
Online ILA observation waveform to see the test results

PROGRAM THE FPGA

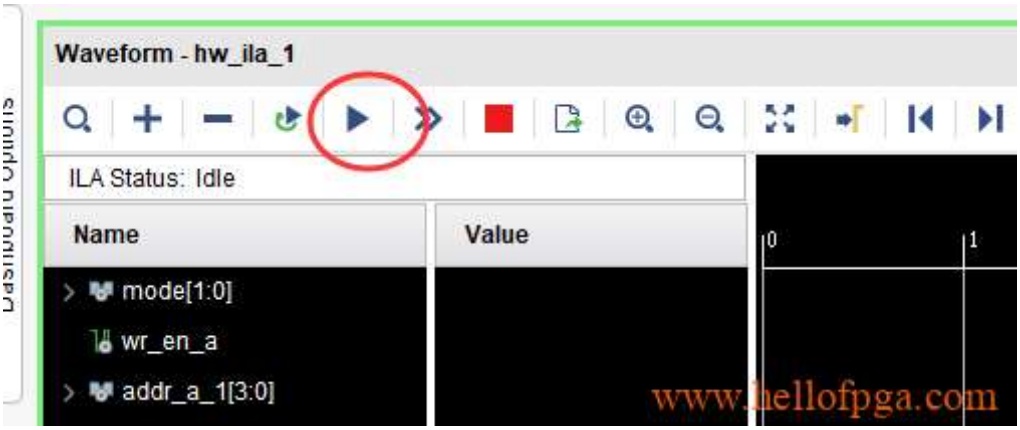
Keeping the default options, click Program



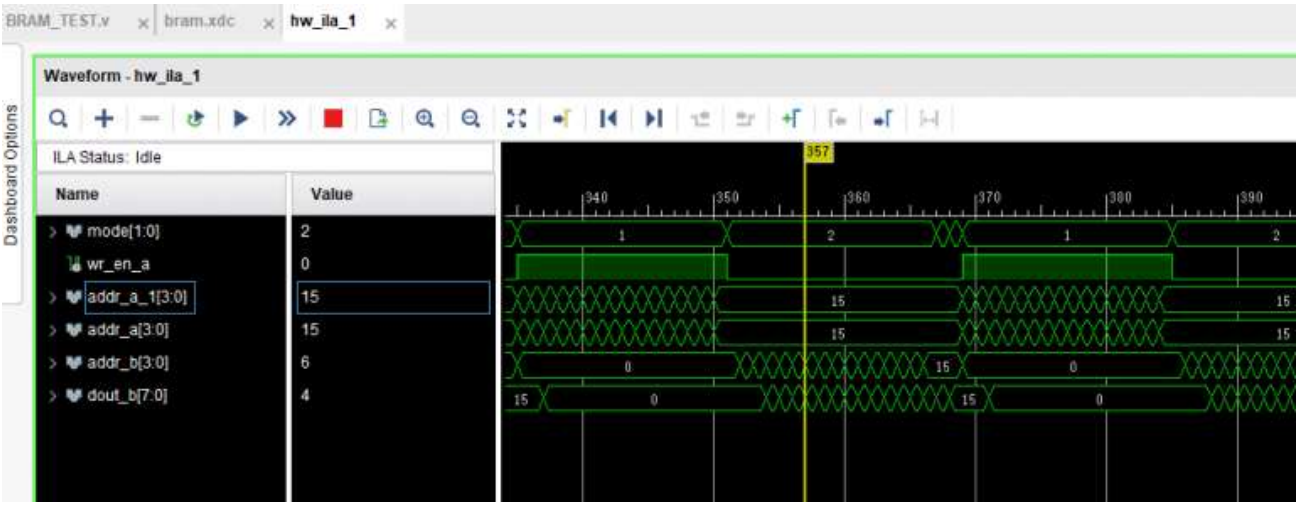
After that, an ILA window will be automatically added (containing all the signals observed with the probe in our previous program)



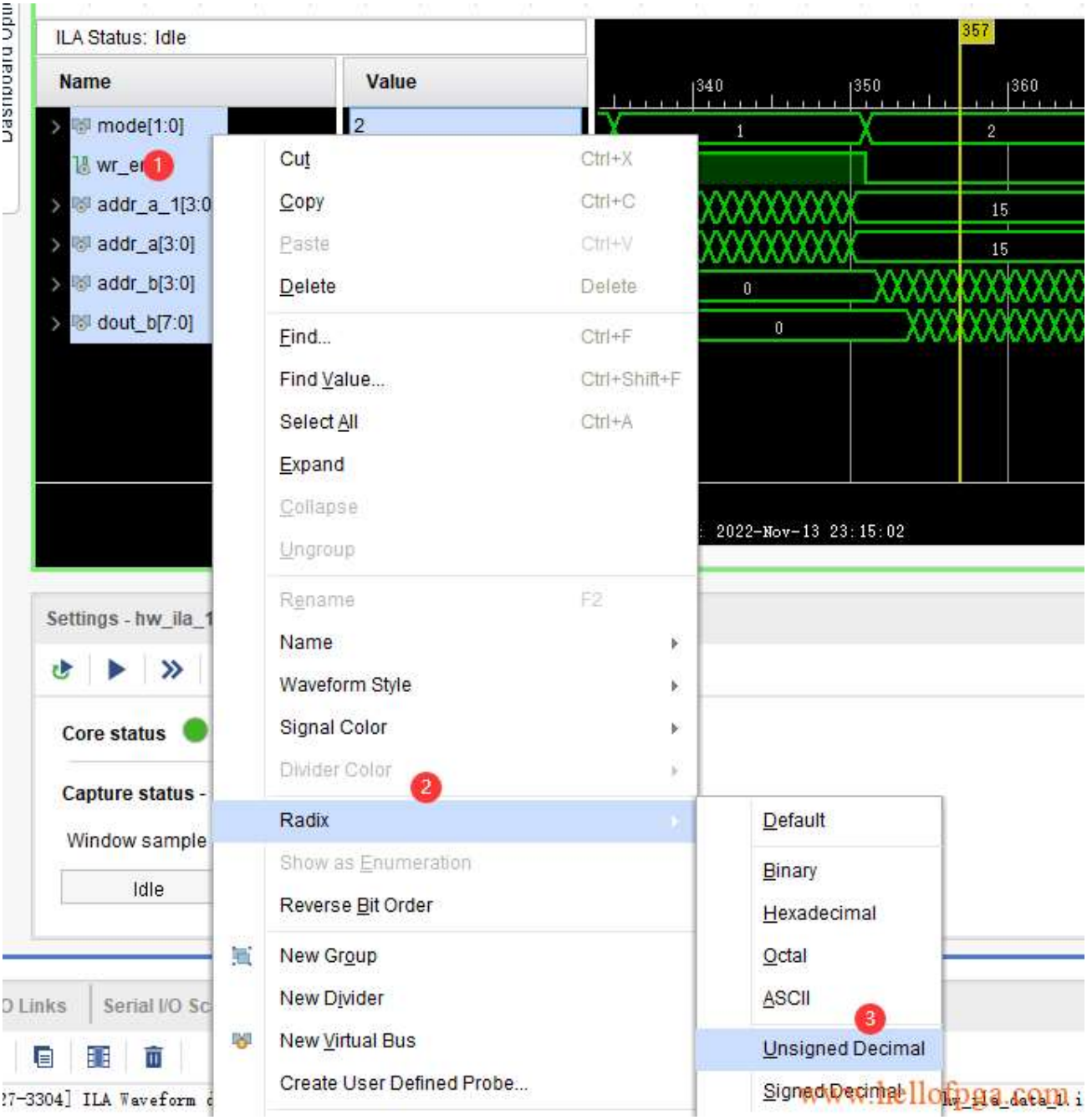
Click the arrow to start grabbing the signal



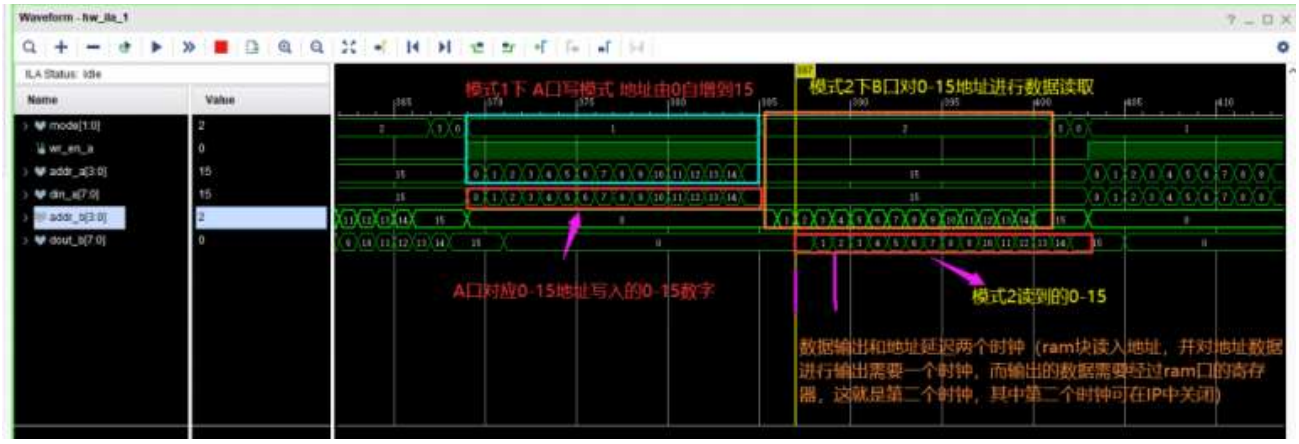
After that, the signal we want to observe is captured



If the data is hexadecimal, you can select all the observed signals and switch the data base



The following figure is the waveform result captured, here are a few places to remark, the output output is with the address has two clock beat delay (RAM block reads into the address, the process of outputting address data requires clock synchronization, here is a clock, and the output data needs to pass through the register of the RAM port, here is the second clock) The second clock delay can be canceled by turning off the output register in the interface of the previous setting bram, but it is not conducive to the stability of the system in the case of high-speed access.



The program is relatively simple, when the BRAM module is actually called, in addition to using ILA to verify, it can also be verified by simulation. In addition, the BRAM module needs to pay special attention to the output delay during use (here it is easy to ignore the delay of the output data in the actual project, resulting in actual bugs)

Here is the complete project of this experiment:

14_PL_BRAM_TEST_XC7Z020 **Download**

SMART ZYNQ SP & SL