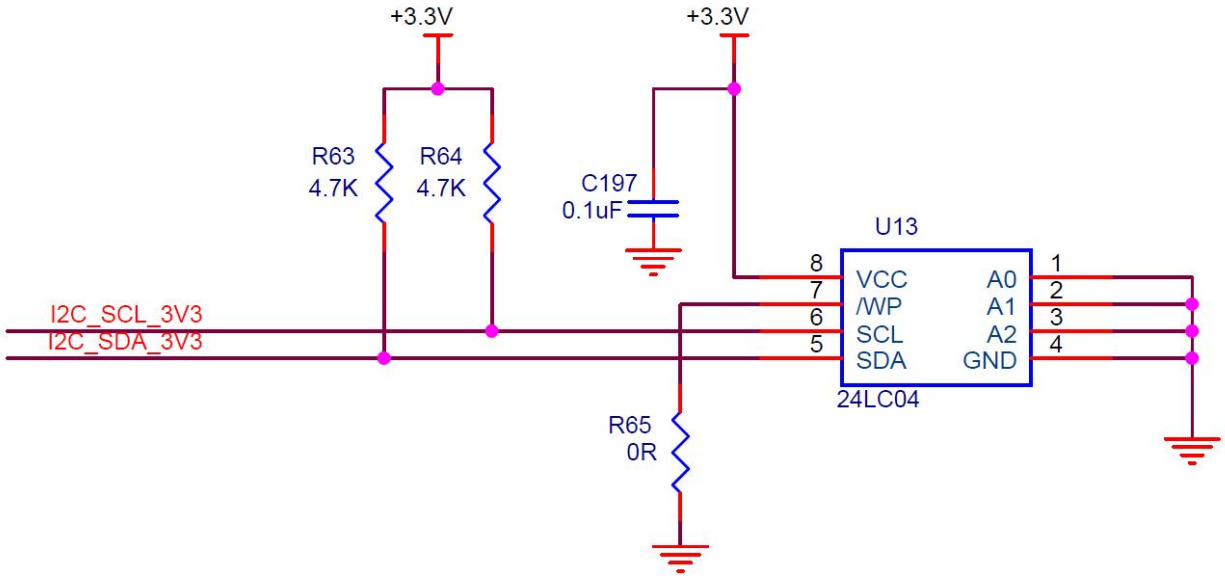
1 Experiment Introduction

This experiment uses the "I2C master" controller on the open source software "opencores" to control the EEPROM read and write of the I2C interface, and practice how to effectively use open source code to improve development efficiency.

2 Experiment Principle

2.1 Hardware Circuit

On the development board, the FPGA chip is connected to the EEPROM 24LC04 through the I2C bus. The two buses of I2C pull up a 4.7K resistor to 3.3V, so it will be pulled high when there is no output on the bus. The write protection of 24LC04 is not enabled. Otherwise, the FPGA will not be able to write data. Because A0~A2 are low on the circuit, the device address of 24LC04 is 0xA0.

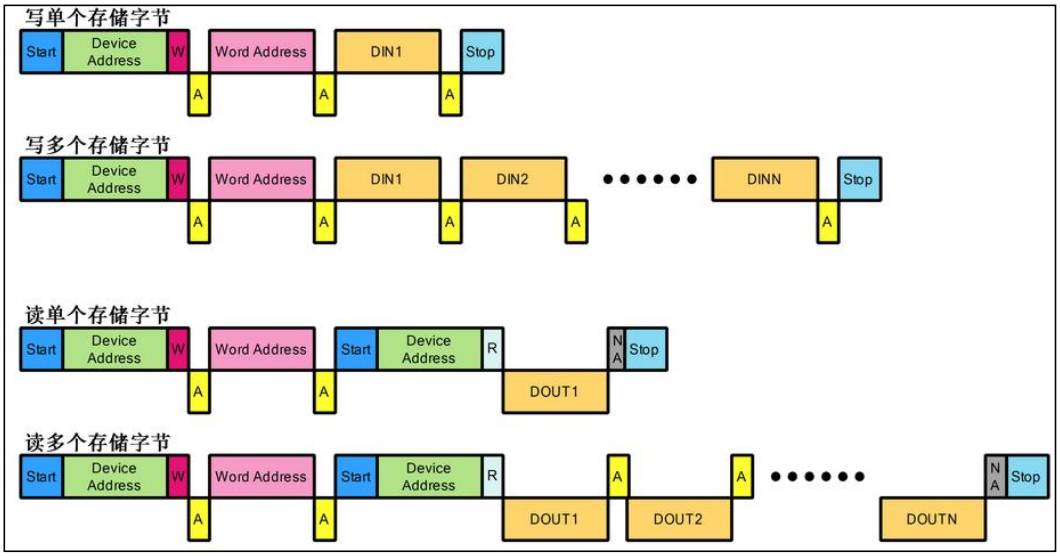


EEPROM Circuit on the AXKU040 FPGA Development Board

2.2 I2C bus protocol and Timing

The I2C standard rate is "100kbit/s", and the fast mode is "400kbit/s". It supports multi-machine communication and supports multiple master modules, but only one master is allowed at the same time. The serial bus is formed by the data line SDA and the clock SCL; each circuit and module has a unique address.

Here, AT24C04 is taken as an example to illustrate the basic operation and timing of I2C read and write. The operation of I2C device can be divided into writing a single storage byte, writing multiple storage bytes, reading a single storage byte and reading multiple storage bytes. The various operations are shown below.



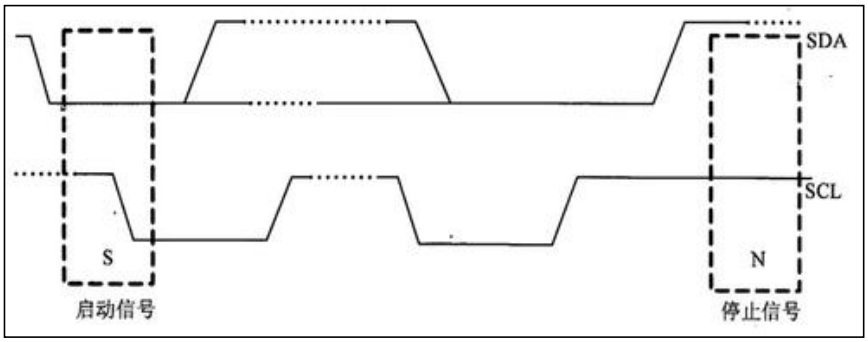
The following is an analysis of several signal states and timings that occur during I2C bus communication.

① Bus idle state

When the two signal lines of SDA and SCL of the I2C bus bus are simultaneously at a high level, the idle state of the bus is specified. At this time, the output stage FET of each device is in the off state, that is, the bus is released, and the level is pulled up by the respective pull-up resistors of the two signal lines.

② Start

While the clock line “SCL” is held high, the level on the data line “SDA” is pulled low (ie, a negative transition), defined as the enable signal of the “I2C” bus bus, which marks the beginning of a data transfer. The start signal is actively established by the master. The I2C bus must be idle before the signal is established, as shown in the following figure.



③ Stop Signal

While the clock line SCL is held high, the data line SDA is released, causing SDA to return to a high level (ie, a positive transition), referred to as a stop signal of the I2C bus, which marks the termination of a data transfer. The stop signal is also actively established by the master. After the signal is established, the I2C bus will return to the idle state.

④ Data Bit Transfer

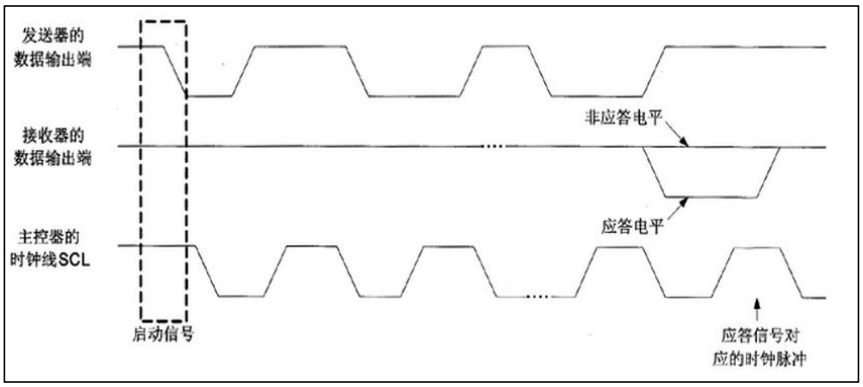
Each bit of data transmitted on the I2C bus has a clock pulse corresponding (or synchronous control), that is, each bit of data is serially transmitted bit by bit on the SDA in cooperation with the SCL serial clock. During data transfer, the level on SDA must remain stable while SCL is high. The low level is data 0 and the high level is data 1. The level change state on SDA is allowed only while SCL is low.

⑤ Response Signal （ACK and NACK）

All data on the I2C bus is transmitted in 8-bit bytes. Each time the transmitter transmits a byte, the data line is released during clock pulse 9, and a response signal is fed back by the receiver. When the response signal is low, it is specified as a valid acknowledge bit (ACK is abbreviated as an acknowledge bit), indicating that the receiver has successfully received the byte;

When the response signal is high, it is specified as a non-acknowledgement bit (NACK), which generally means that the receiver has not received the byte successfully. The requirement for the feedback valid acknowledge bit, ACK, is that the receiver pulls the SDA line low during the low period before the ninth clock pulse and ensures a stable low level during the high period of the clock.

If the receiver is the master, after it receives the last byte, it sends a “NACK” signal to inform the controlled transmitter to end the data transmission and release the SDA line so that the master receiver sends a stop signal.



3 Programming

Although the I2C timing is simple, there are a lot of problems with poor writing. On the open source website http://opencores.org/ we can find a lot of very good code, most of which provide detailed documentation and simulation. Appropriate use of open source code can not only improve our development efficiency, but also learn other people's development ideas. Since most of the code is

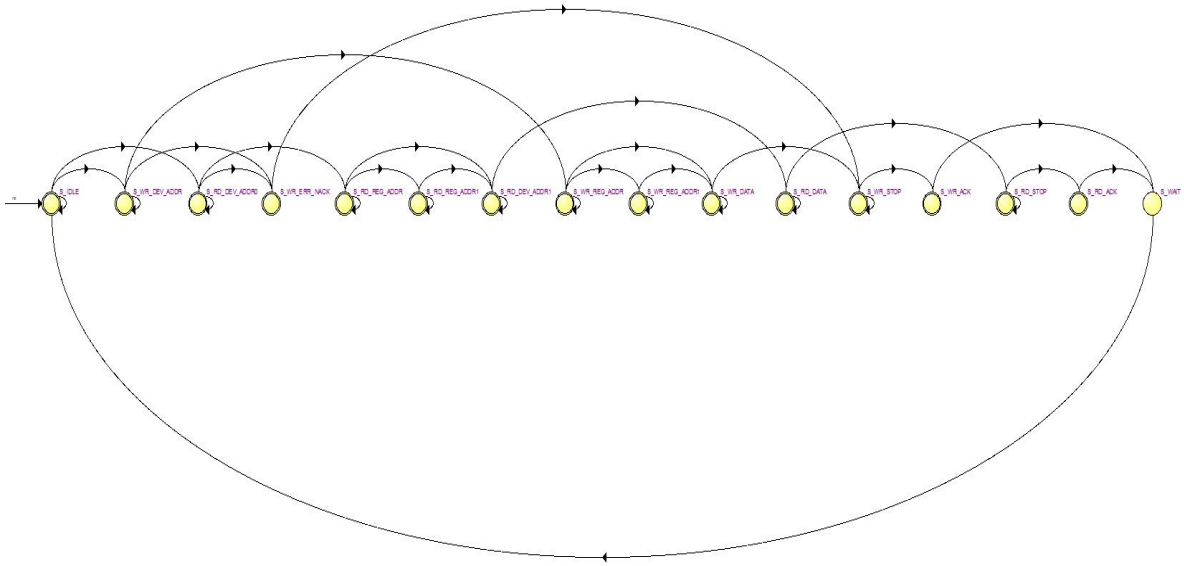
repeatedly modified and re-refined after a long time, some code may be difficult to understand. When you can't understand other people's code well, the best way is to simulate.

Gamepads	Stats		GPL		
General-Purpose I/O (GPIO) Core	Stats	wbc			
GPIB (IEEE-488) controller	Stats		GPL		
Hardware Assisted IEEE 1588 IP Core	Stats	wbc	LGPL		
HDB3/B3ZS Encoder+Decoder	Stats		BSD		
HDLC controller	Stats	wbc			
HyperTransport Tunnel	Stats				
I2C controller core	Stats	wbc OCCP	BSD	B.3	
I2C Master Slave Core	Stats		BSD		
I2C master/slave Core	Stats	wbc			
I2C Multiple Bus Controller	Stats	wbc	BSD		
I2C Repeater	Stats		LGPL		
I2C Slave	Stats		GPL		
I2C Traffic Logger	Stats				
i2cgpio	Stats		LGPL		
i2c_to_wb	Stats	wbc	LGPL		
I2S Interface	Stats	wbc	GPL		
I2S to Paralell ADC/DAC controller	Stats		GPL		
I2S to Parallel Interface	Stats		GPL		
I2S to WishBone	Stats		LGPL		

According to the “IP core” document, the “i2c_master_byte_ctrl” module mainly performs one byte of reading and writing. We only need to complete the device address, register address, data, etc. according to the requirements of I2C read and write.

The “i2c_master_top module” repackages the i2c_master_byte_ctrl module and completes reading and writing of a register. Since different device registers may be 8bit or 16bit, the i2c_addr_2byte signal controls whether the register address is 8 or 16 bits.

The i2c_master_top module state machine, if it is a write register operation, first writes a byte device address (write operation), then writes a 1-byte or 2-byte register address, and then writes a byte of data; If it is a read operation, first write a byte of the device address (write operation), then write a 1-byte or 2-byte register address, complete the address write, write the device address again (read operation), and then read One byte of data. In any case, the programming is to meet the chip timing requirements, so it is best to read the chip's data sheet carefully before reading the program.



i2c_master_top State Machine

Signal Name	Direction	Description
clk	in	Clock input
rst	in	Asynchronous reset input, high reset
clk_div_cnt	in	I2C clock division factor, equal to the system clock frequency / (5 * I2C clock frequency) - 1. For example, 50Mhz system clock, 100Khz I2C, configured as 99,400Khz I2C, configured as 24.
scl_pad_i	in	I2C clock data input, this experiment can be ignored
scl_pad_o	out	I2C clock output
scl_padoen_o	out	I2C clock output enable, low effective, I2C external pull-up resistor, if the output high impedance state, it will be pulled high, in this experiment, high output output high resistance
sda_pad_i	in	I2C data input
sda_pad_o	out	I2C output data
sda_padoen_o	out	I2C data output enable, low effective. In this experiment, the output is high impedance at the high level output.。
i2c_addr_2byte	in	Whether the register address is 8 or 16 bits, 1:16 bits, 0:8 bits
i2c_read_req	in	I2C register read request
i2c_read_req_ack	out	I2C register read request response
i2c_write_req	in	I2C register write request
i2c_write_req_ack	out	I2C register write request response
i2c_slave_dev_addr	in	I2C device address, 8bit, the lowest bit is ignored, and the valid data bit is the upper 7 bits.

i2c_slave_reg_addr	in	Register address, 8-bit address, the lower 8 bits are valid
i2c_write_data	in	Write data register
i2c_read_data	out	Read register data
error	out	Device no response error

i2c_master_top Module Port

The i2c_eeprom_test module finishes reading and writing the EEPROM. The EEPROM device address is A0. The data of the address 00 is read out in the program, and then displayed by the LED. When the KEY2 is pressed, the number is added and written to the EEPROM again and displayed. In the I2C controller, most of the code's features also have a lot of comments in the comments.

4 Experiment Result

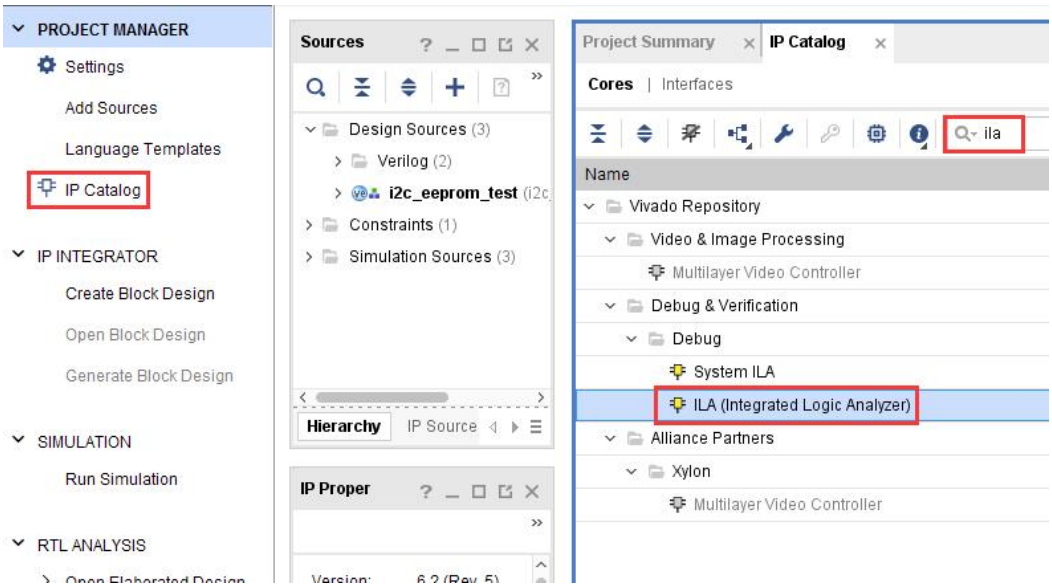
After downloading the experiment program, you can see that the LED displays a binary number. This number is the data stored in the 00 address in the EEPROM. The data is random. At this time, the KEY2 is pressed, the number is increased by one, and the EEPROM is written and downloaded again. The program can see the updated data directly.

AXKU040 FPGA Development Board

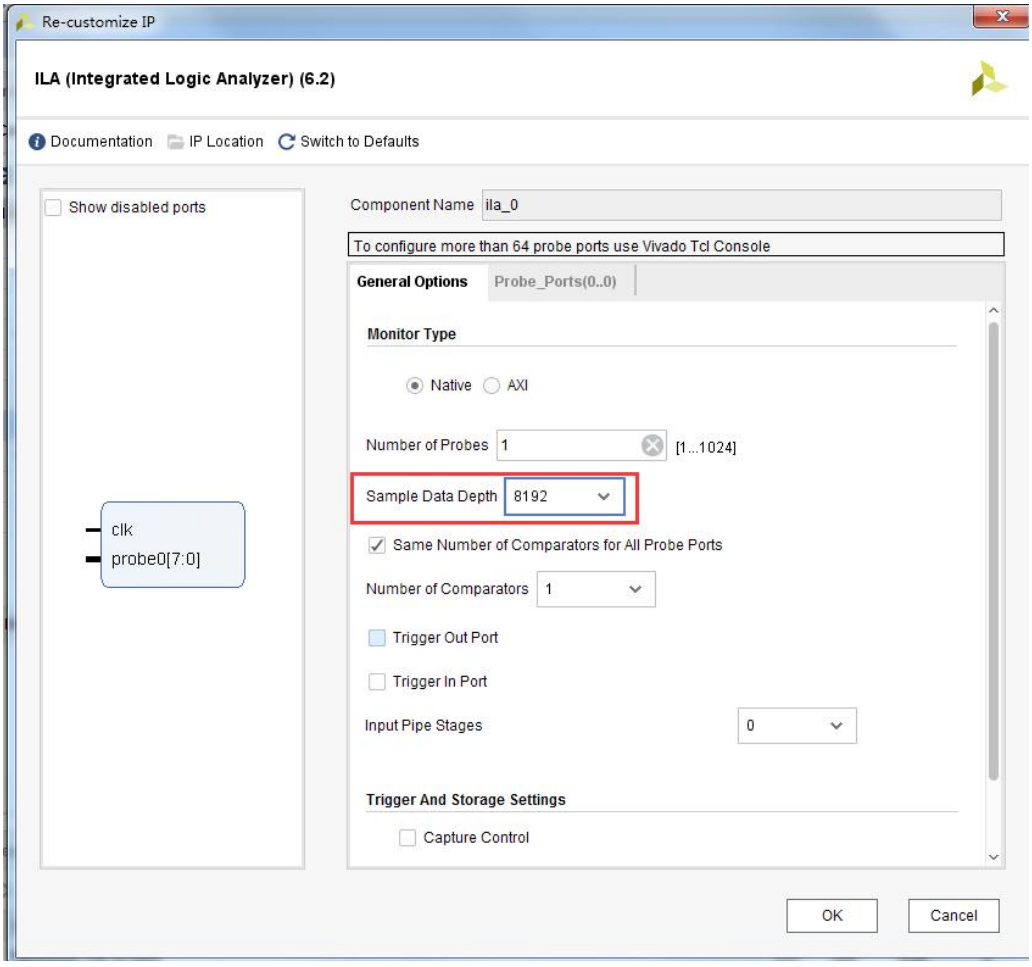
5 Use vivado logic analyzer ila to observe signals

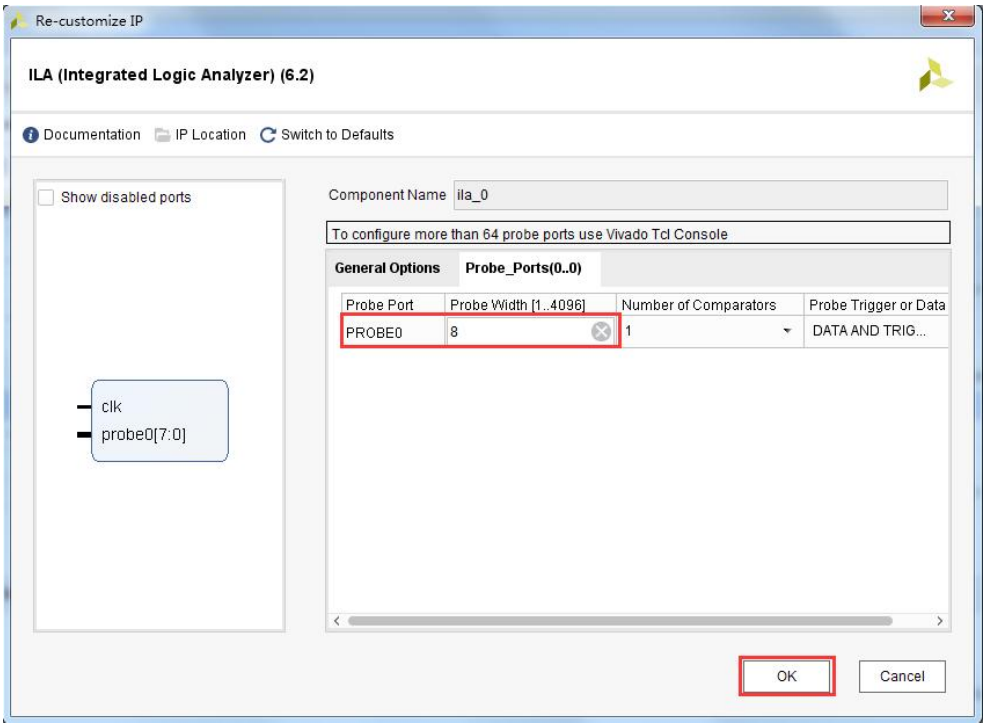
Using “vivado ila” can be very intuitive to see the changes in the various signals when the program is running on the development board. Add an “ila core” to this routine to observe the changes of the data lines when the program is running.

Press the red mark and select "ILA" in the pop-up dialog box to double-click:



In the following interface pop-up, the following settings are made. Here, the sampling depth is 8192 and the bit width is 8 (can be customized). When finished, click OK:

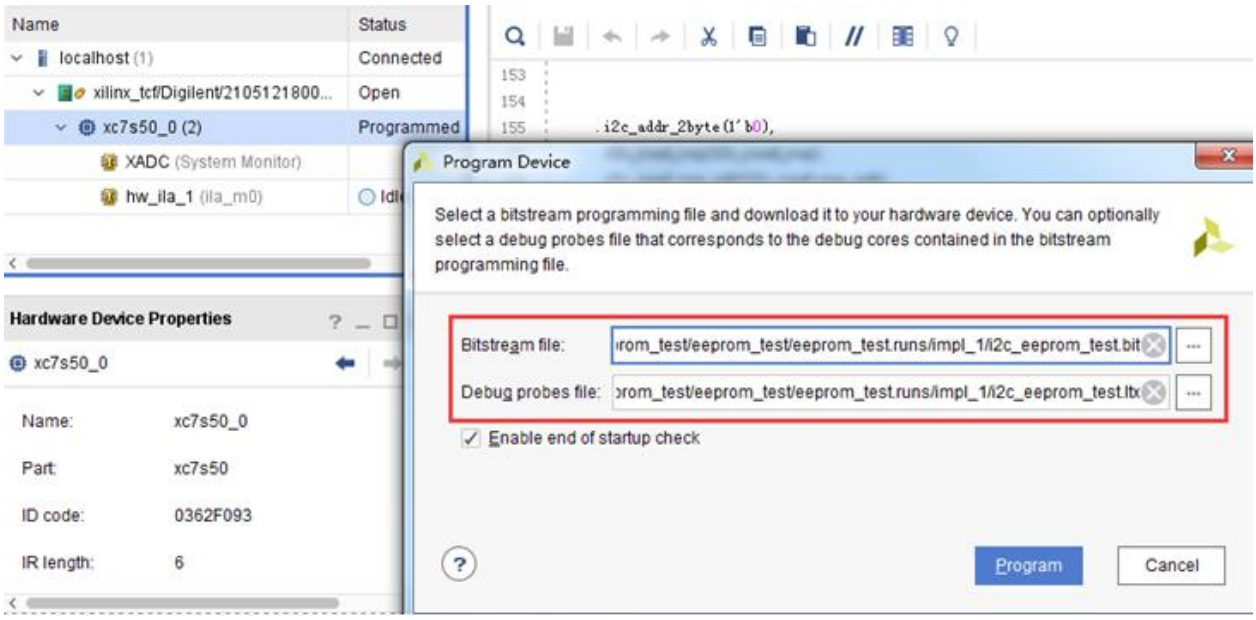




Instantiate in the top-level file, the code is as follows:

```
ila_0 ila_m0 (
        .clk(sys_clk), // input wire clk
        .probe0(read_data) // input wire [7:0] probe0
);
```

After finish, Compile synthesis and download ".bit" file



The following interface click key will pop up. Each time you press “KEY2” to run once, you can see that the data is incremented by 1.