**GOWIN GW1NSR-LV4C Step by Step Tutorial**

**Cortex-M3 in FPGA Design Sample - SPI LCD control with RT-Thread OS**

*Background:*

*This Tutorial is translated from internet published article originally in Chinese.*

*RT-Thread is an open source embedded real-time operating system from China, by a number of professional developers in 2006 began to develop, maintain, in addition to real-time operating system kernel and similar FreeRTOS UCOS also includes a series of application components and drivers frameworks, such as the TCP/IP protocol stack, the virtual file system, POSIX interfaces, graphical user interface, FreeModbus master-slave protocol stack, CAN framework, dynamic modules, etc., because the system is stable, feature-rich new features are widely used in energy, power grids, fan high reliability industries and equipment, have been validated is a highly reliable real-time operating system.*

*GW1NSR-LV4C is GOWIN’s uSOC FPGA with embedded Cortex-M3 hard core.*

1. **Overview**

This tutorial describes the features and design resources of the GOWIN GW1NSR-4 FPGA device. With the information provided from the GOWIN Website, FPGA users can easily design and debug their projects themselves. GOWIN official website provides IDEs and programming download tools including synthesis, user guides, reference manuals, and other design resources. At the end of this tutorial, you will find all related online resources. In Chapter 3, a blinking LED project is explained step by step to show how to write a proper Verilog design. More importantly, it shows how to instantiate the IP cores in a design. In this design example, the on-chip hardcore CORTEX-M3 CPU (Cortex-M3) processor is utilized. By using the GOWIN IP core configuration tool, we can enable the SPI bus, UART and GPIO0 IPs to control the SPI LCD display. After finishing this chapter, I hope readers can instantiate any peripherals provided in the GOWIN EDA tools. In Chapter 5, how to deploy RT-Thread into this uSOC FPGA is discussed. I certainly hope after this tutorial, you can start your DIYs easily.

1. **GW1NSR FPGA Brief**

Verilog is the important hardware design language used in the digital IC front-end for IC verification. There are EDA tools call synthesizer which can translate the Verilog to RTL, another low-level hardware description language that can be used to produce IC chips or bitstream file to program FPGA. It is best way to learn Verilog through FPGA design process and the GOWIN MiniStar board with GW1NSR-LV4CQN48P is a perfect platform to learn Verilog.

An ARM Cortex-M3 hardcore processor and several DSP blocks are embedded in GW1NSR-LV4C FPGA, you can utilize them by coding your design with Verilog language. FPGAs are very useful for IO extensions because of their parallel process friendly nature, thus, can greatly help to improve the performance of the MCU. The advantage of having a CORTEX-M3 CPU integrated on-chip is that it can reduce the footprint of PCB, means low cost. Traditional products with MCU+FPGA have bigger size of the PCB and cost more. With current global MCUs shortage, GW1NSR-LV4C solution becomes more important as an alternative solution. Besides, it has higher performance than many MCUs. It is unique in which it can process serial logic on the CPU and meanwhile, process parallel logic in FPGA fabric. For more resources, see the GW1NSR FPGA Family data sheet.

Link: [http://cdn.GOWINsemi.com.cn/DS861E.pdf](http://cdn.gowinsemi.com.cn/DS861E.pdf)

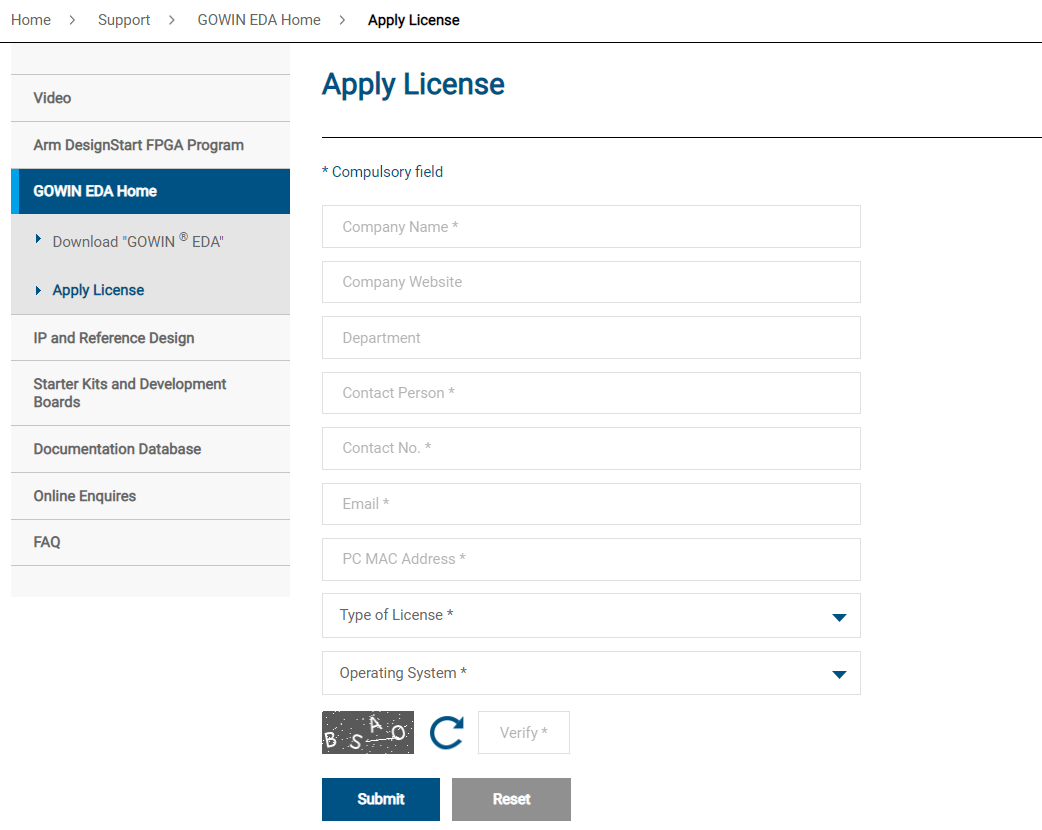
1. **Getting Started with Blinking LED** 
   1. **Development Environment**
      1. **Download and Install Software**

Link: [https://www.GOWINsemi.com/en/support/download\_eda/](https://www.gowinsemi.com/en/support/download_eda/)

****

* + 1. **Apply for license**

Link: [https://www.GOWINsemi.com/en/support/license/](https://www.gowinsemi.com/en/support/license/)



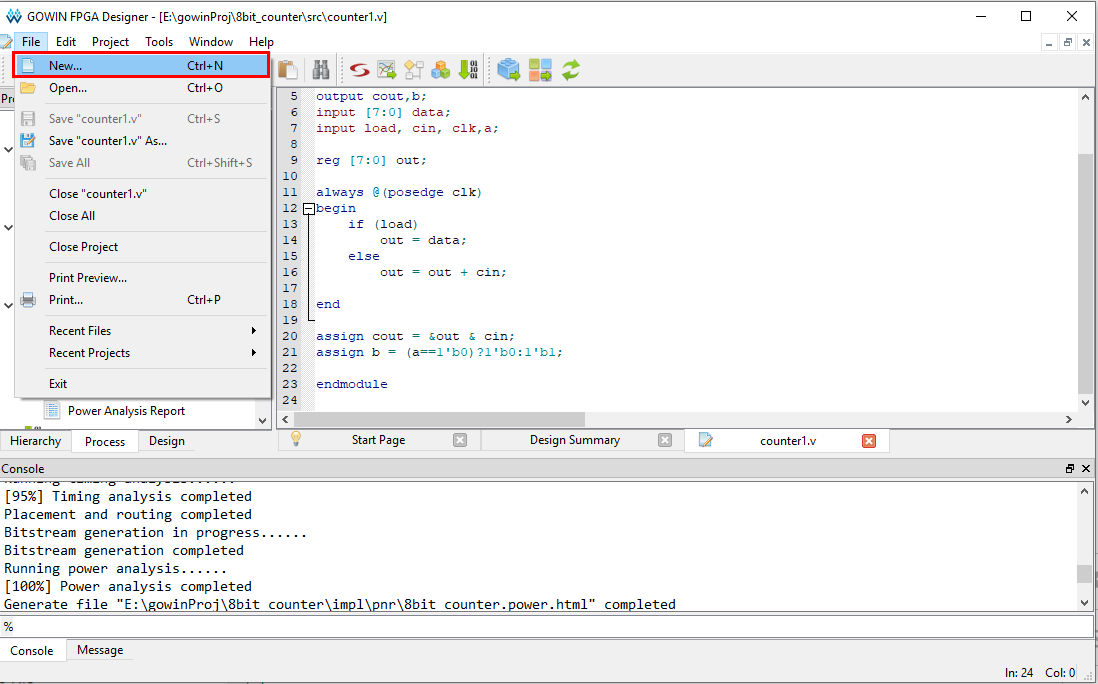
* + 1. **Install MiniStar USB driver**

See this article for reference.

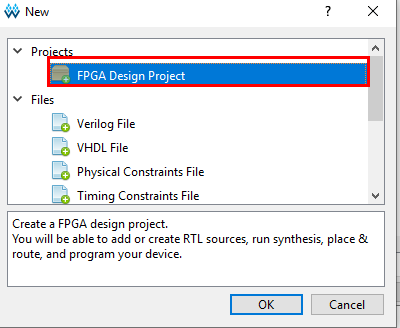
Link:<https://blog.csdn.net/csdnyueguoyu/article/details/99577971?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162312085716780269896555%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&amp;request_id=162312085716780269896555&amp;biz_id=0&amp;utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-99577971.first_rank_v2_pc_rank_v29&amp;utm_term=ft2232d&amp;spm=1018.2226.3001.4187>

* 1. **Create a New Project and Verilog File**
     1. **Create a New Project**

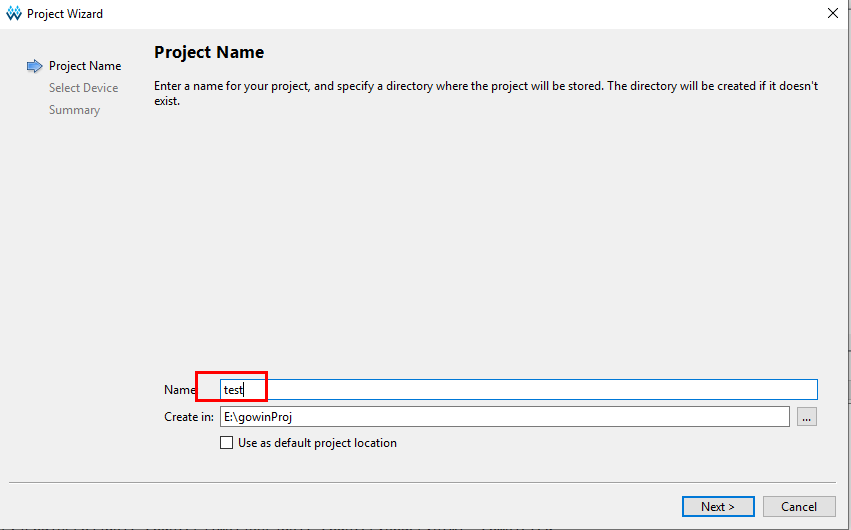
Click “File” -> “New”.

****

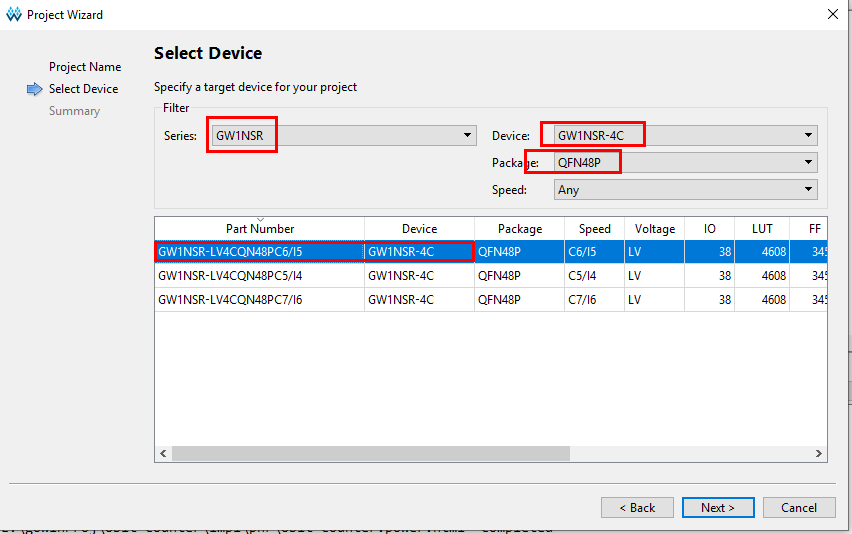
Select “FPGA Design Project”.

****

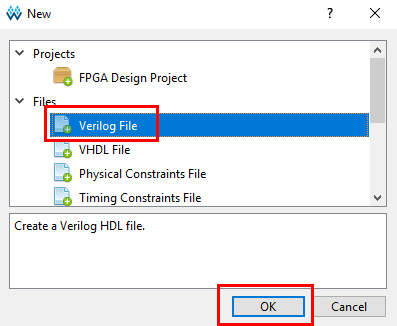
Import project name.

****

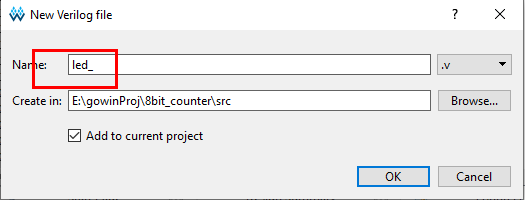
Select device.



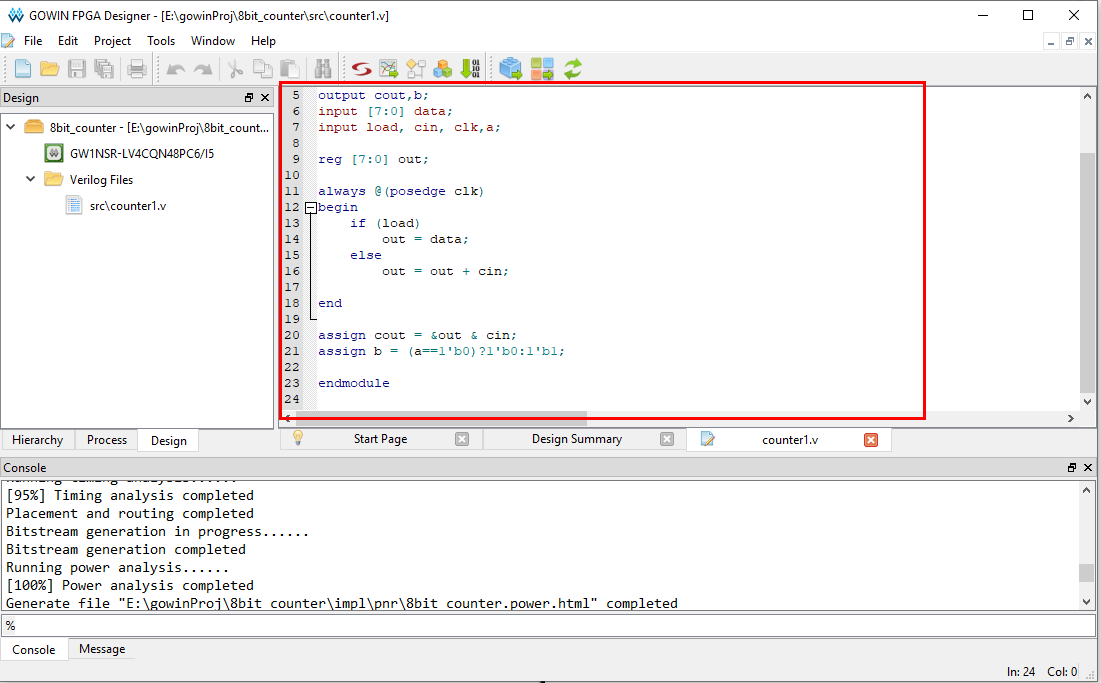
* + 1. **Create & Compile Verilog File**

****

The module name should be consistent with the file name.

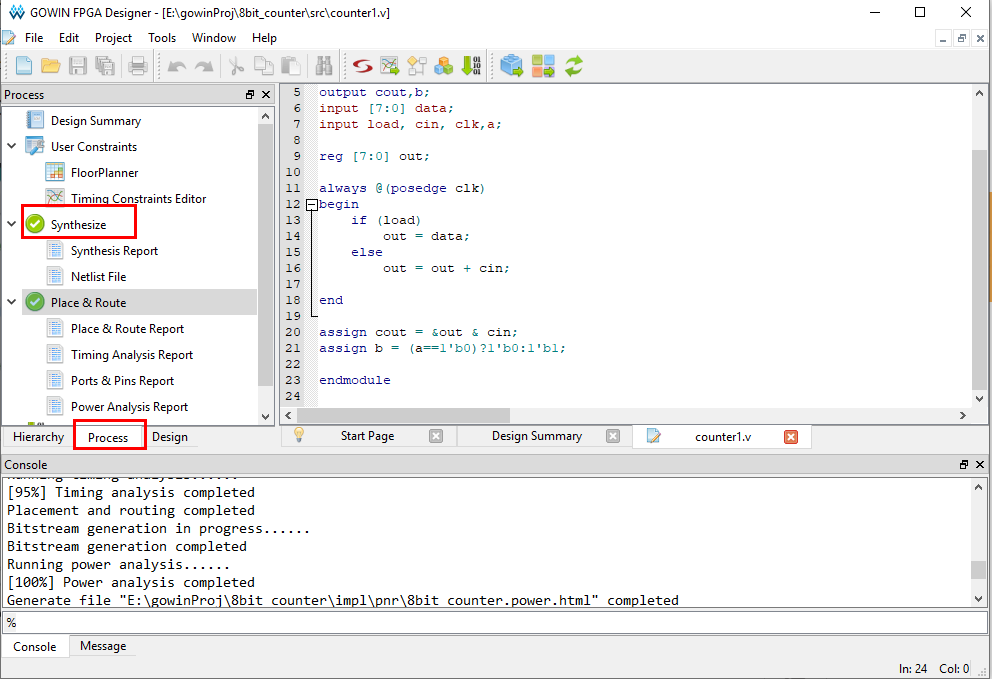
****

Compile Verilog codes.

****

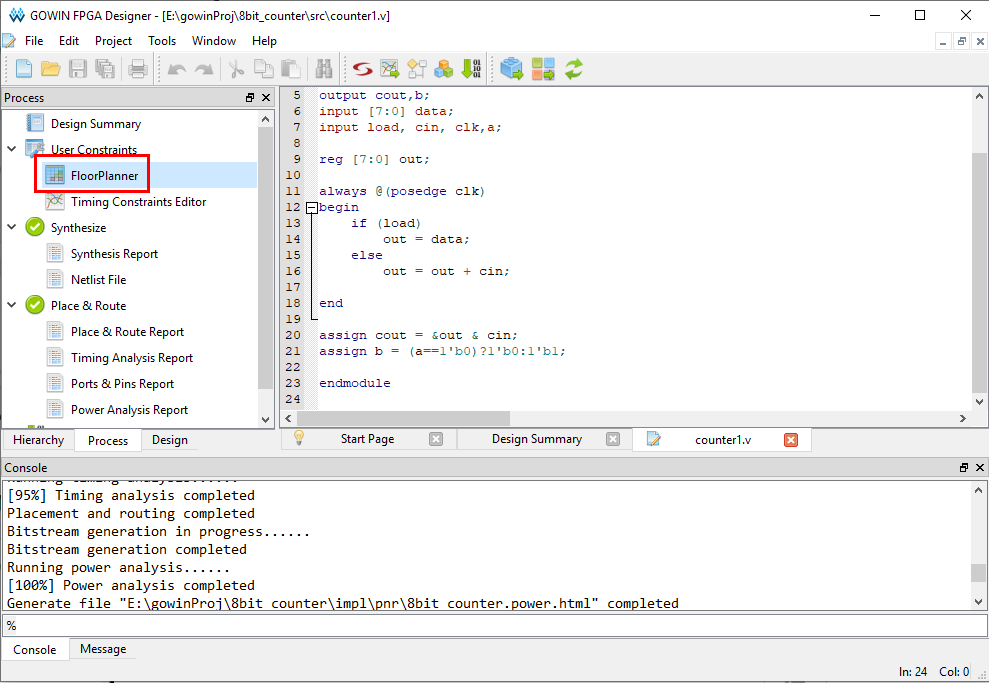
1. //author：qwz
2. //time：2021.6.8
3. //version：v0.1
4. //desc: clk signal connected to the crystal on the hardware, asynchronous reset (reset on the falling edge of the reset pin). for every clock pulse clk, counter register cnt+1. when counting up to 1000\_0000, clear the counter, shift the 2-bit LED to left by one.
5. ​
6. module led\_flow(input clk,//clock
7. input rst\_n,//reset
8. output reg [1:0] led  //2-bit led
9. );
10. ​
11. reg [23:0] cnt;// Clock pulse counter
12. ​
13. always @(posedge clk or posedge rst\_n) begin
14. if(rst\_n)
15. cnt <= 24'd0 ;// clear the counter on the falling edge of reset
16. else
17. cnt <= (cnt == 24'd1000\_0000)?(24'd0):cnt+1'b1;// clear the counter when the count value reaches 1000\_0000, otherwise counter value +1
18. end
19. ​
20. always @(posedge clk or posedge rst\_n) begin
21. if(rst\_n)
22. led <= 2'b01 ;// after reset, led[0] is on, led[3:1 ] off
23. else  if(cnt == 24'd1000\_0000)
24. led <= {led[0],led[1]};// when the count value reaches 1000\_0000, shift the 2-bit led
25. else
26. led <= led;// when the count value does not reache 1000\_0000, no shifting
27. end
28. ​
29. endmodule
    1. **Synthesis**

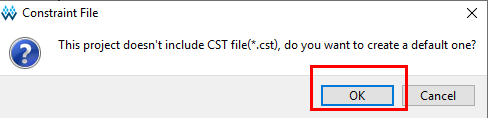
Left-click “Process”, and then right-click “Synthesize” -> “run”.

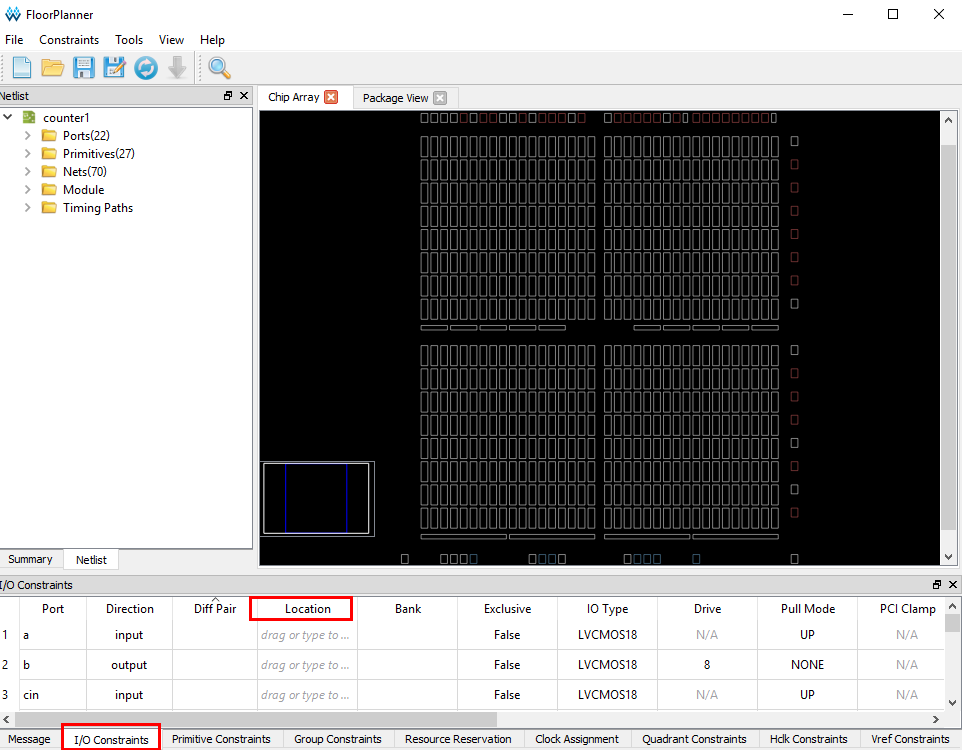
****

* 1. **Pinout**

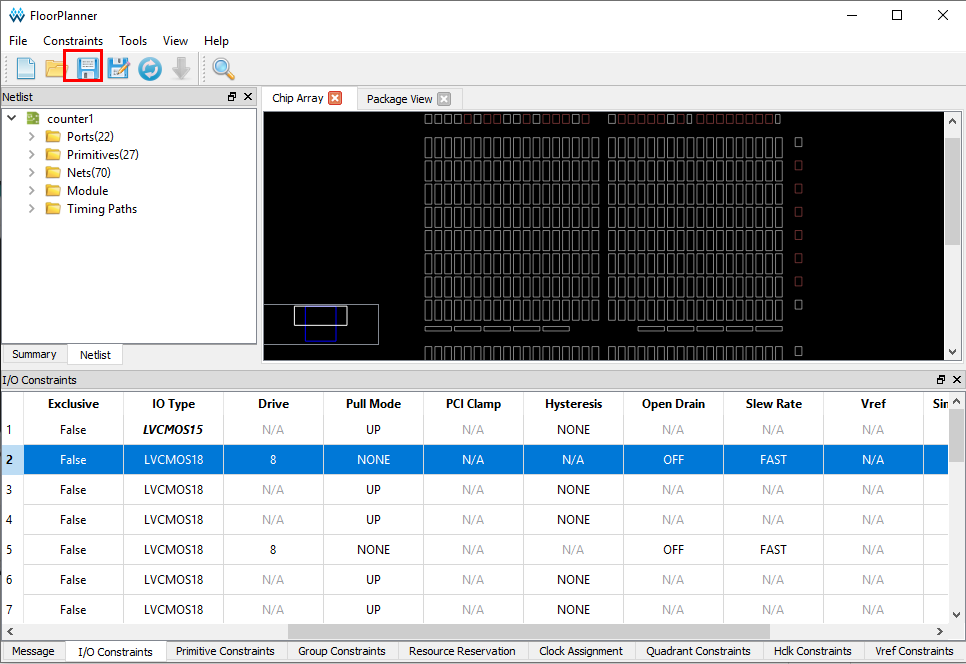
Double click “floorplanner”.

****

****

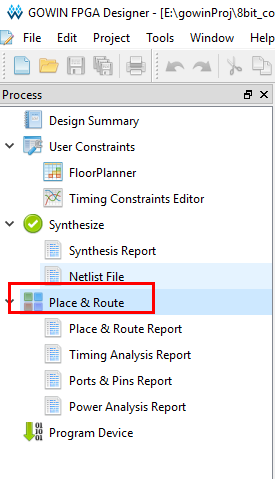
****

Click save after configuration.

****

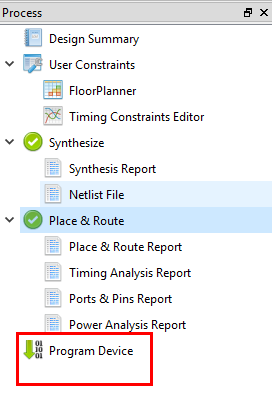
* 1. **Placement & Routing**

Right-click “place & Route” -> “run”.

****

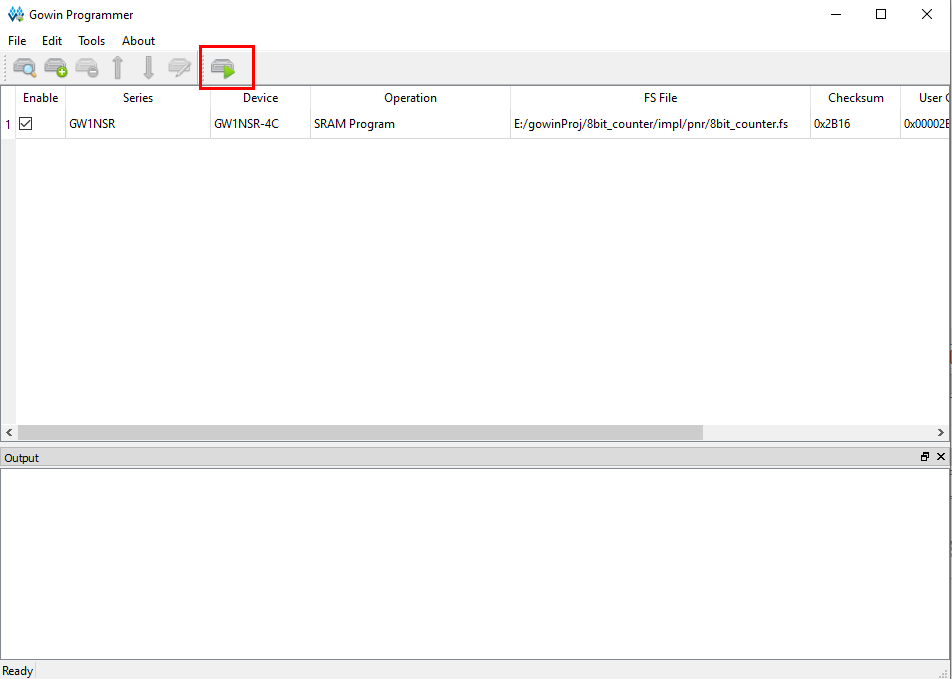
* 1. **Download**

Double click “Program Device”.

****

Click to download.

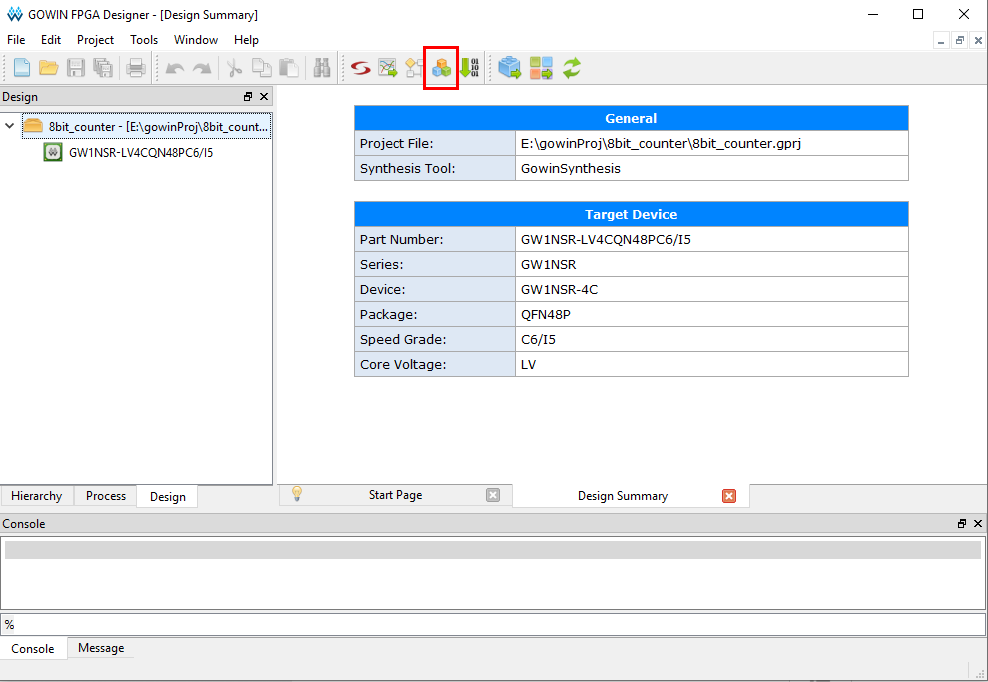
After the download is complete, the two LEDs on the board will keep flashing in turn.

****

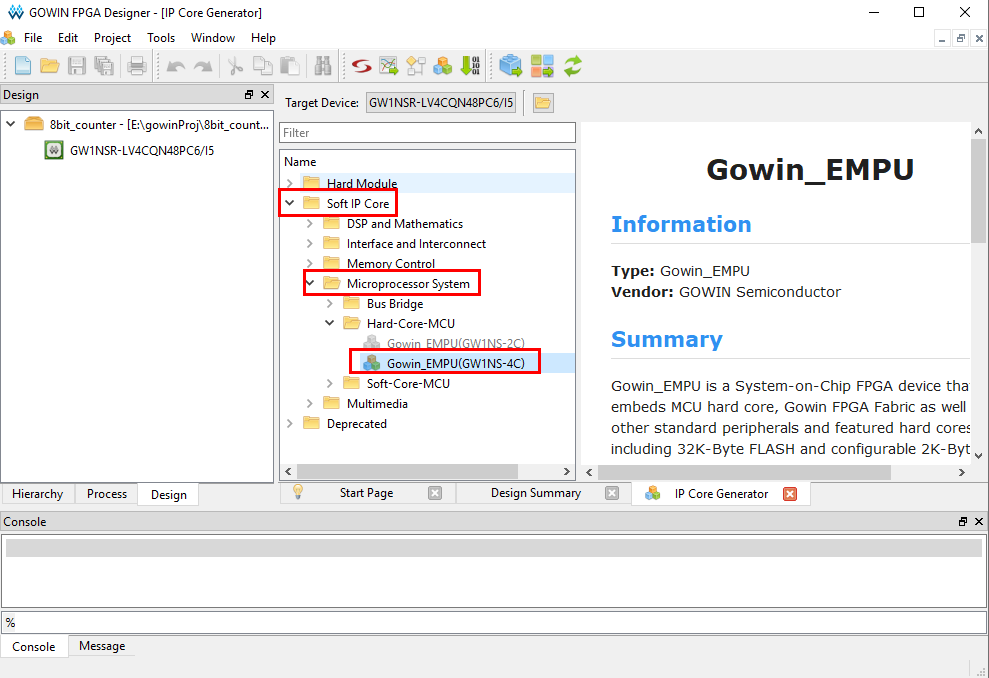
1. **Practice: IP core applications -- On-chip Hard-core Processor**

Please refer to Chapter 3 for the method of creating a new project. The main content of this chapter is to use the IP core generator to enable the on-chip hard core’s SPI peripheral to drive the LCD.

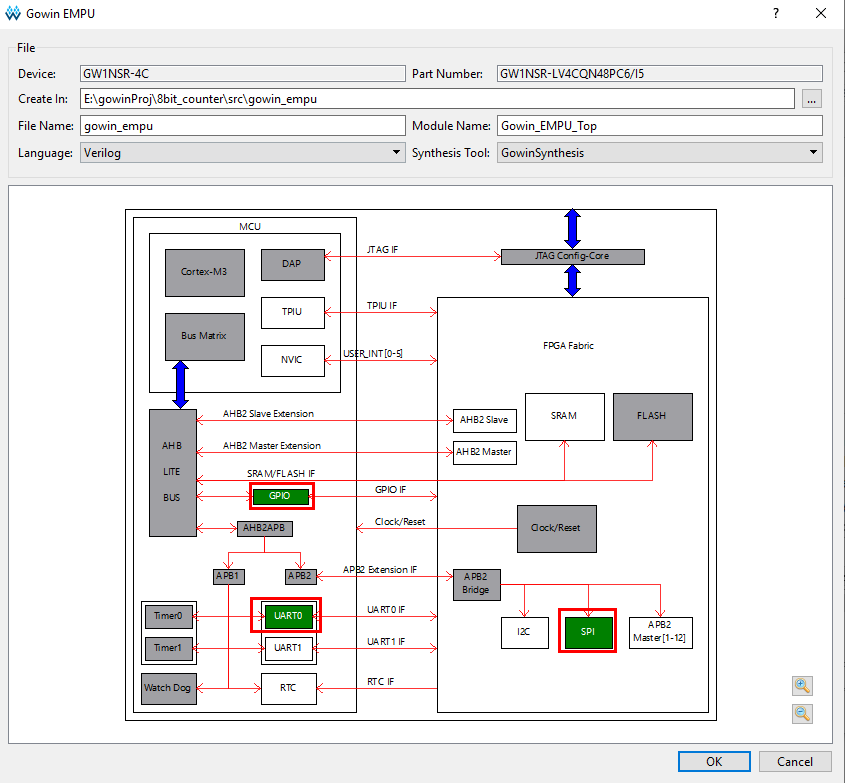
Open IP core library.



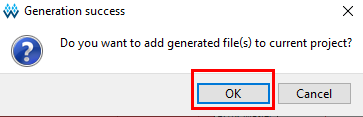
Click “Soft IP Core” -> “Microprocessor System” -> “GOWIN\_EMPU(GW1NS-4C)”.



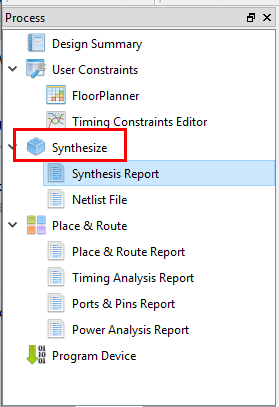
Double click “GPIO”, “UART0”, “SPI” to enable them.



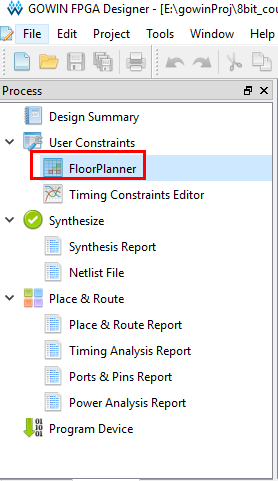
Click “OK” to add generated file to current project.



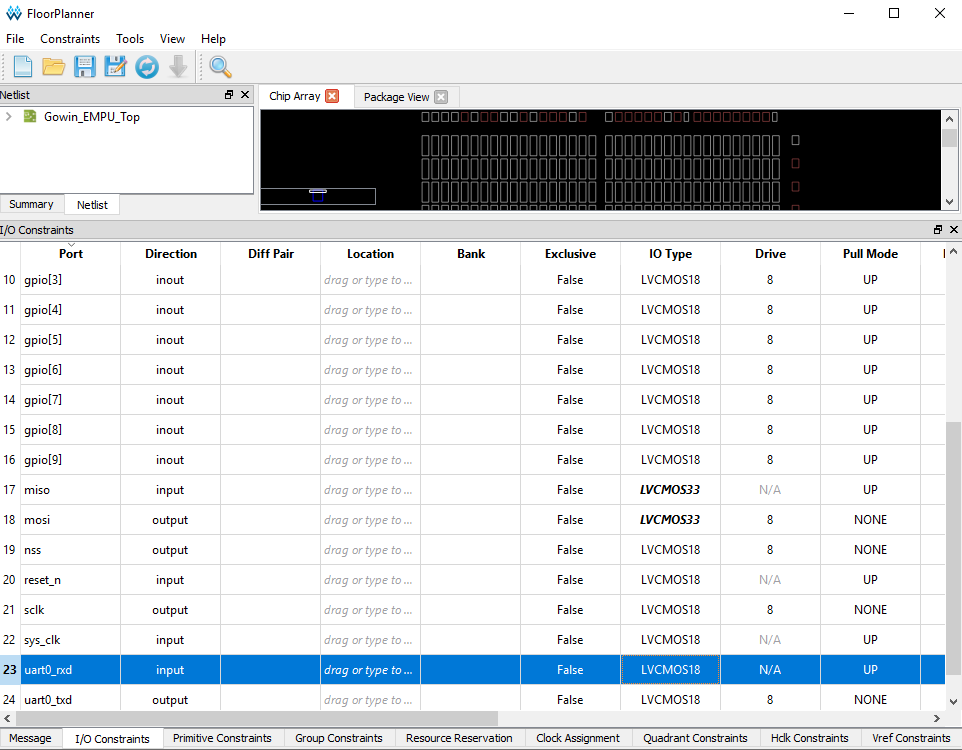
Double click “Synthesize”.



Double click “FloorPlanner”.

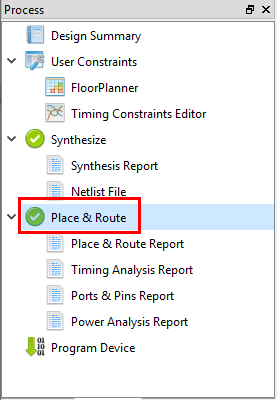


Configure SPI, uart and GPIO pins.



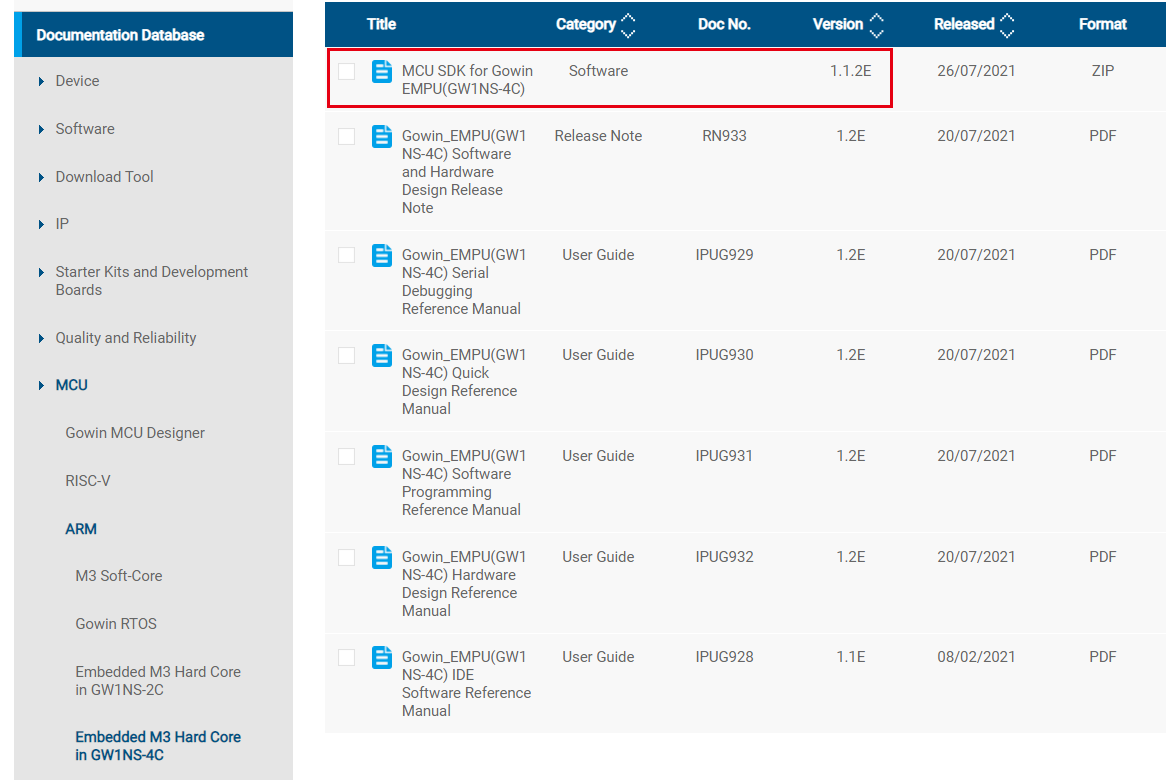
Note that the IO type should be LVCMOS33 because 3.3V is required to drive the SPI chip.

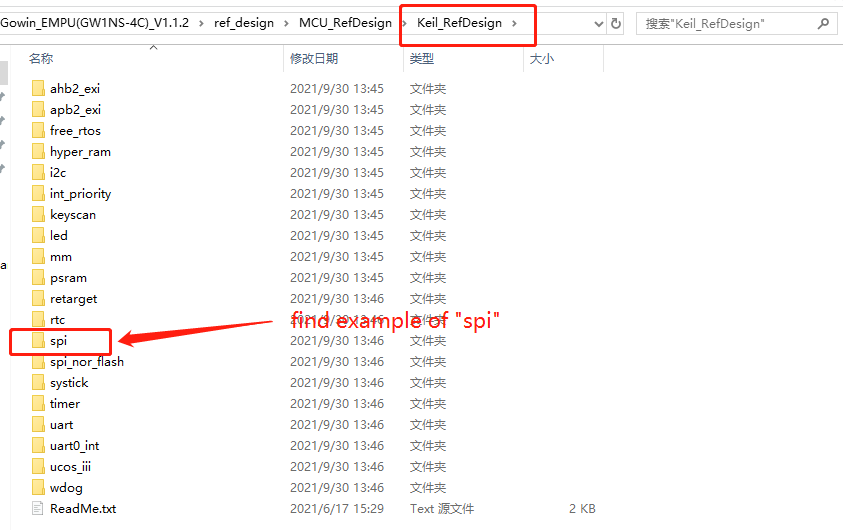
Click “Place & Route”.

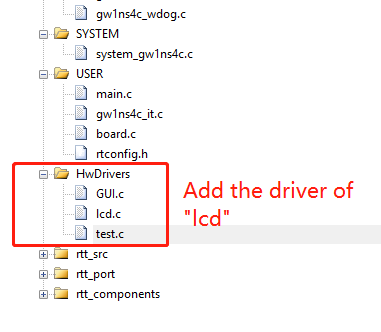


This is the end of the FPGA configuration. Now Let’s start the CORTEX-M3 CPU programming.

Download the firmware package from [https://GOWINsemi.com/en/support/database/](https://gowinsemi.com/en/support/database/)







The most important thing here is to understand the “write register “and “write data” functions of SPI. What you need to do is as follows. For the detailed driving code, please refer to the project file at the end of the tutorial.

Use the SPI write function SPI\_WriteData(data) of the GOWIN MCU firmware library to define the chip select macro for the SPI and the command/data enable macro for LCD driving.

The code migration is quite easy.

void LCD\_WR\_REG(uint8\_t data)

{

LCD\_CS\_CLR;

LCD\_RS\_CLR;

//HAL\_SPI\_Transmit(&hspi1,&data,1,100);

SPI\_WriteData(data);

LCD\_CS\_SET;

}

void LCD\_WR\_DATA(uint8\_t data)

{

LCD\_CS\_CLR;

LCD\_RS\_SET;

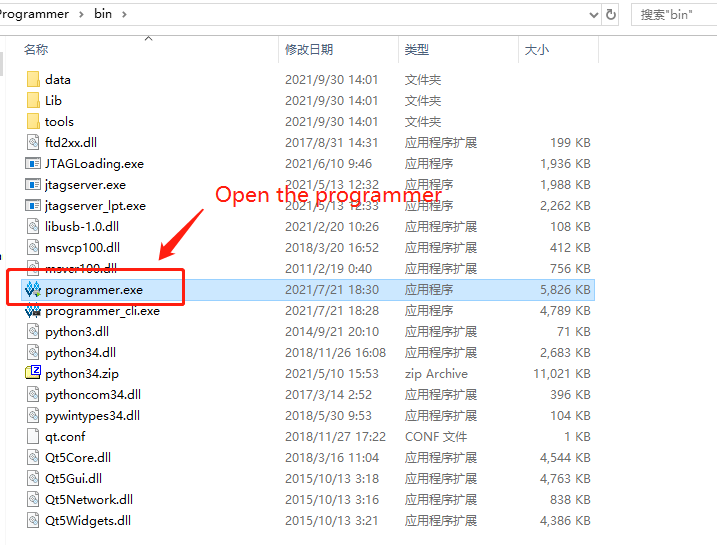
//HAL\_SPI\_Transmit(&hspi1,&data,1,100);

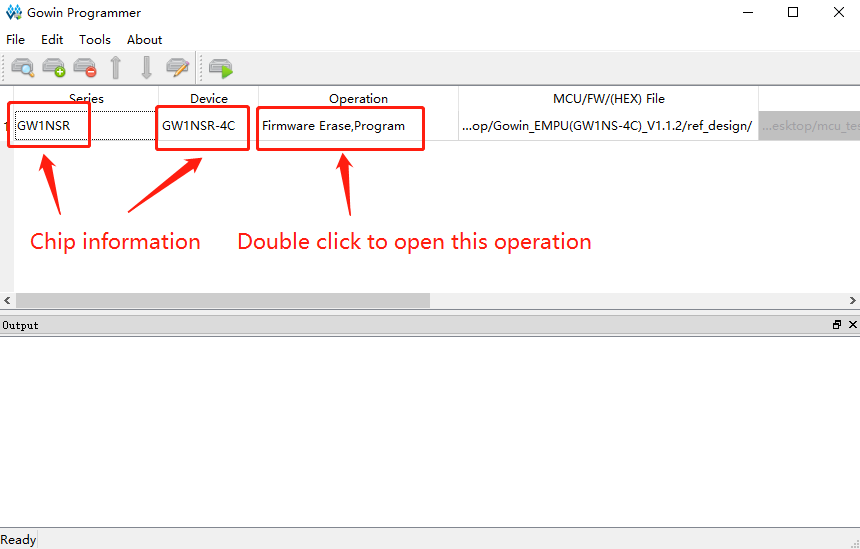
SPI\_WriteData(data);

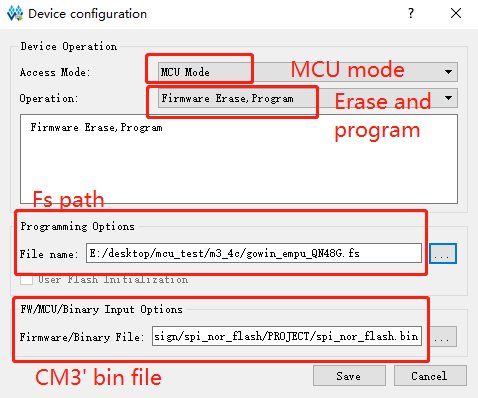
LCD\_CS\_SET;

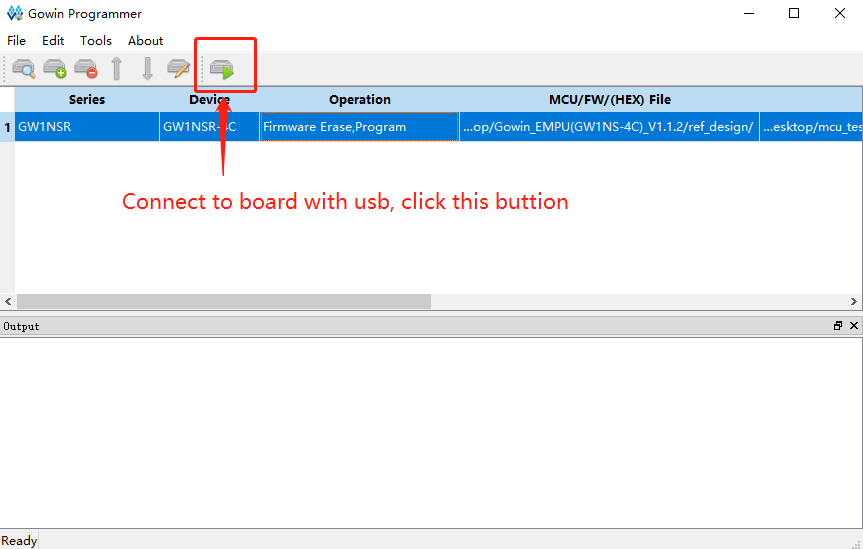
}

Now we have completed the work for the FPGA and CORTEX-M3 CPU. The fs file is generated in FPGA project synthesis, and the bin file is generated in CORTEX-M3 CPU project compilation. Then we just need to download it.







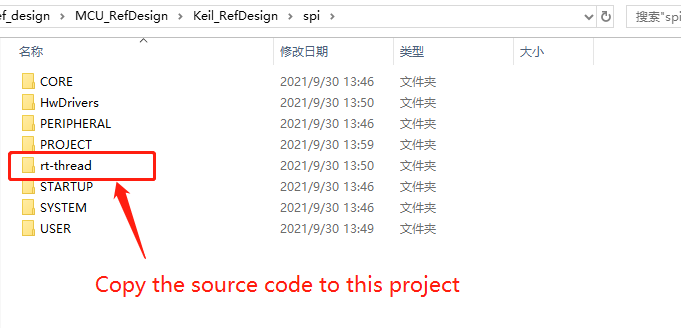


Now the LCD should start to display the “test function” in screen, indicating that the driver is normal, and Cortex-M3 CPU is working normally.

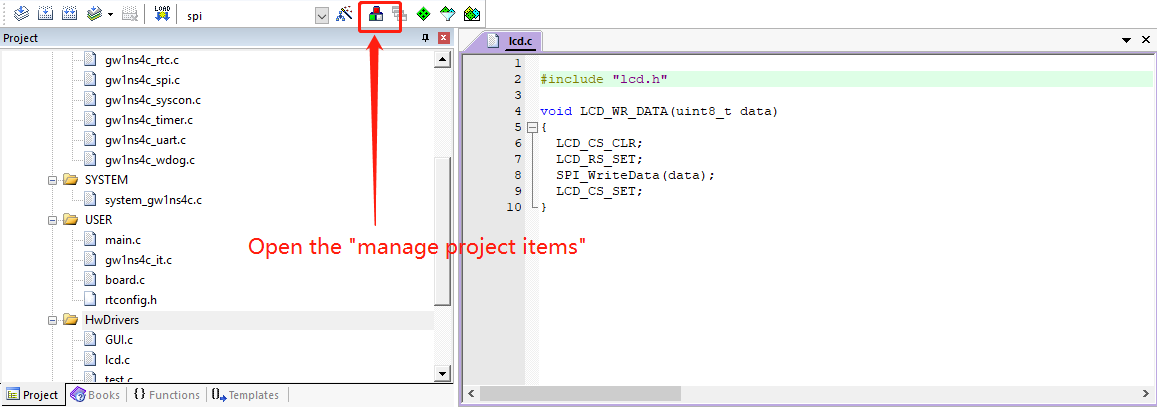
1. **Migrate the Operating System RT-Thread to CPU**
   1. **Download the RT-Thread nano 3.1.3 kernel source code and copy it to the project directory.**

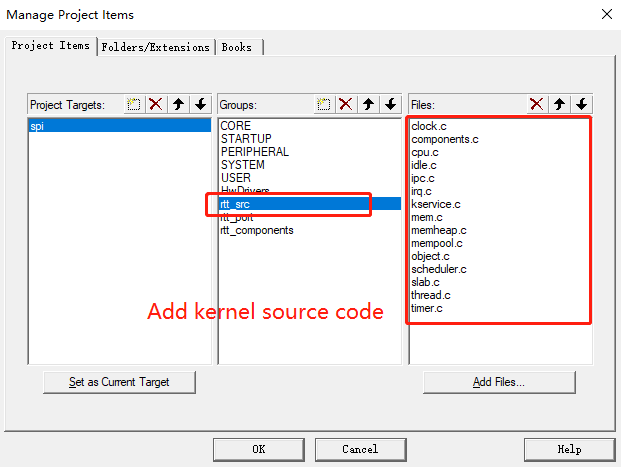
Download URL：<https://www.rt-thread.org/page/download.html>

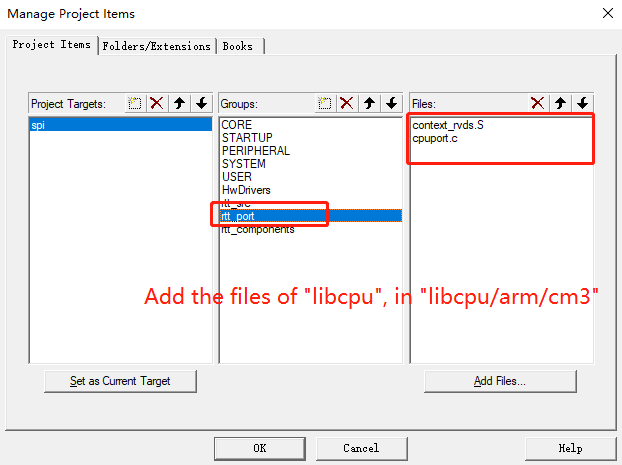


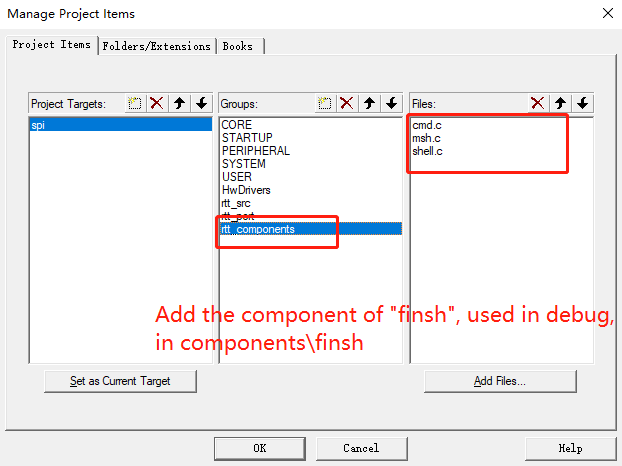


* 1. **Add Source Files in the keil Project**

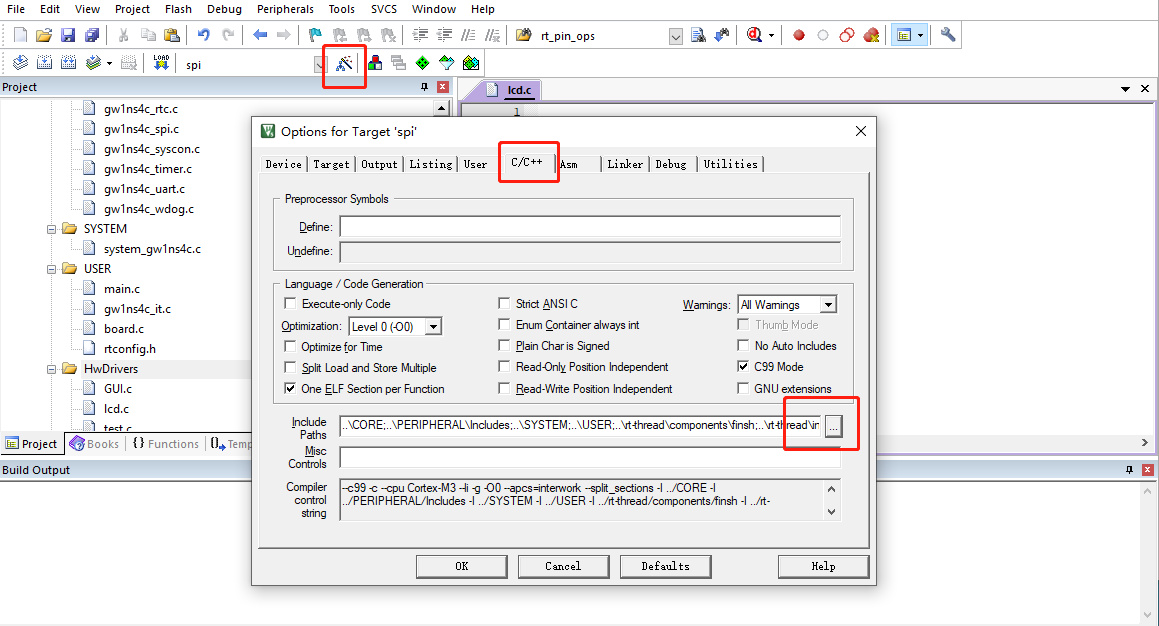
****

****

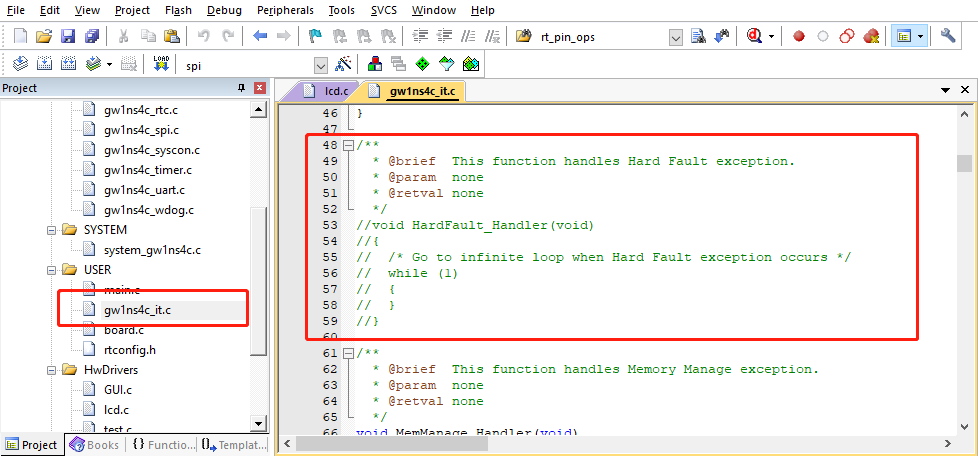
****

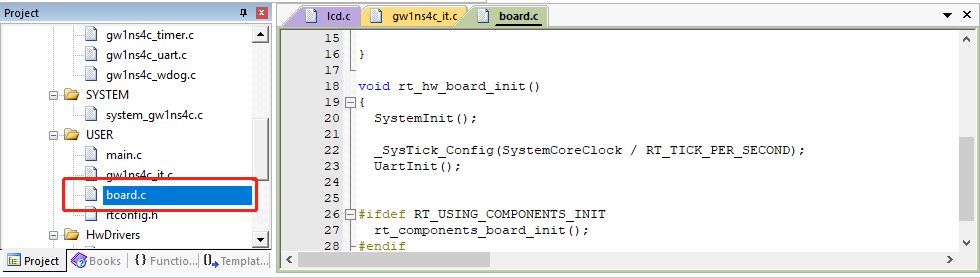
****

* 1. **Add the Header File**

****

* 1. **Comment out ystick, hardfault, pendsv Interrupt**

****

****

After completing the above steps, compile the project, download it to the development board. Then RT-Thread will start to run on the development board.

1. **Conclusion**This is my first time to work on a domestic FPGA in China, I was happily surprised that the IP cores are quite abundant. When I explored GOWIN official website, I found that some reference manuals of IP cores, such as RISC-V and ARM CPU cores, were released early in 2018. That means GOWIN has a petty long period developed and perfected their FPGA solutions, as a result, GOWIN provide easy to use IDE software and easy to understand reference manuals for their users. This MiniStar development board is very suitable for students and beginners. And I hope that GOWIN can provide more such easy-to-use boards, help advancing the Verilog learning and the IC research and development.