[L02]PL LED 呼吸灯 + 固件固化

流水灯相当的简单,现在就来练手做一下呼吸灯,熟悉一下Verilog语法,工程还是上次的工程,应该就能轻松改出来,主要是增加各种模块,让每一个灯都呼吸上几次,然后切换到下一个灯.

功能分块设计,整体就不难了.

1. PWM时基.
2. 时基比较.
3. 改变CMP.
4. LED切换定时器.
5. 切换LED.

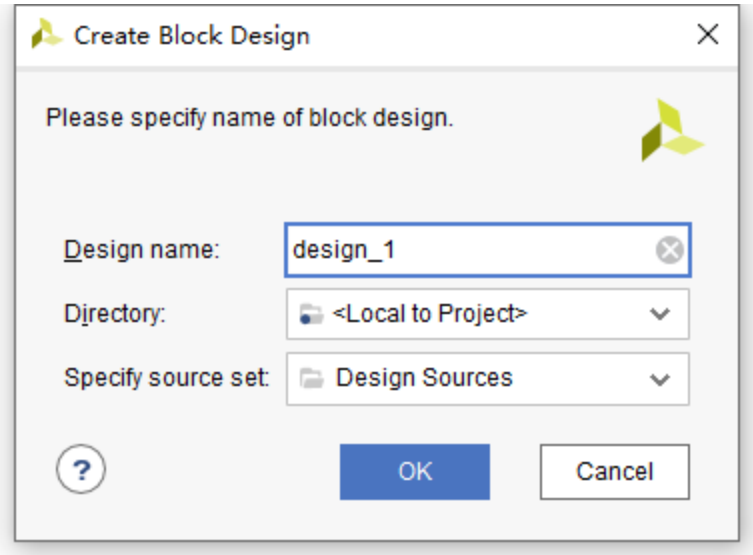
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/02/19 23:07:24
// Design Name:
// Module Name: led_stream
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module led_stream(output reg [3:0] led, // 显示4个LED
```

```
                  input clk,            // 时钟输入
                  input rst);           // 复位输入

reg [31:0] cnt; // 这是一个32Bit计数器(依然是用于LED切换)

reg [31:0] pwm_cnt; // PWM计数器
reg [31:0] pwm_cmp; // PWM比较数值
reg pwm_dir; // CMP改变方向 0 = 向上 1 = 向下
reg pwm_out; // PWM匹配时候拉低,不匹配拉高.
reg pwm_ovf; // PWM周期溢出,此寄存器用于改变PWM CMP,每次溢出改变一下.

reg [1:0] led_i; // 指示哪个LED开始亮起.

// PWM时基,会自由计数.
always@(posedge clk or negedge rst)
begin
    if (!rst)
    begin
        pwm_cnt <= 'b0;
        pwm_ovf <= 'b0;
    end
    else
    begin
        begin
        if (pwm_cnt == 25000) // 计算到顶,则不继续计算.
        begin
            pwm_cnt <= 'b0;
            pwm_ovf <= ~pwm_ovf;
        end
        else
        begin
            pwm_cnt <= pwm_cnt + 'b1;
        end
    end
end
end

// 时基比较,当遇到比较时候支持PWM电平,当计数值越过CMP时候,则拉低,否则拉高.
always@(pwm_cnt)
begin
    if (pwm_cnt < pwm_cmp)
    begin
        pwm_out <= 'b1;
    end
    else
    begin
        pwm_out <= 'b0;
    end
end

// 改变CMP让PWM呼吸起来.
always@(posedge pwm_ovf or negedge rst)
begin
    if (!rst)
```

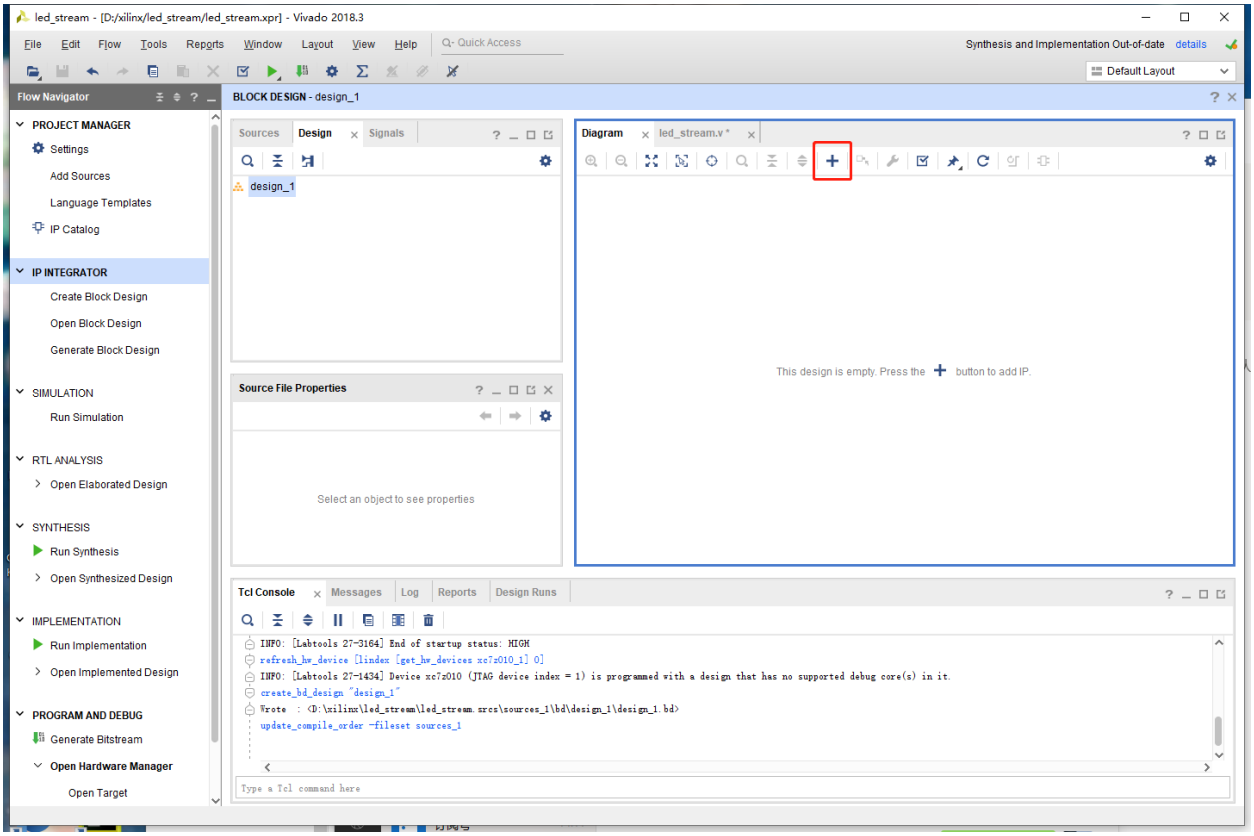
```
    begin
        pwm_dir <= 'b0;
        pwm_cmp <= 'b0;
    end
    else
    begin
        if (pwm_dir == 'b0)
        begin
            if (pwm_cmp == 5000) // 计算到顶,则不继续计算.
            begin
                pwm_dir <= 'b1;
            end
            else
            begin
                pwm_cmp <= pwm_cmp + 'd2;
            end
        end
        else
        begin
            if (pwm_cmp == 0) // 计算到底,则不继续计算.
            begin
                pwm_dir <= 'b0;
            end
            else
            begin
                pwm_cmp <= pwm_cmp - 'd2;
            end
        end
    end
end

// 还是以前那个流水灯的定时器.不过时间长了10倍.
always@(posedge clk or negedge rst)
begin
    if (!rst)
    begin
        cnt   <= 'b0;
        led_i <= 'b0;
    end
    else
    begin
        if (cnt == 250000000)
        begin
            cnt   <= 'b0;
            led_i <= led_i + 'b1;
        end
        else
        begin
            cnt <= cnt + 'b1;
        end
    end
end

// 切换LED
```

```
always@(pwm_out)
begin
    case(led_i)
        2'b00:
        begin
            led[0] <= pwm_out;
            led[1] <= 'b0;
            led[2] <= 'b0;
            led[3] <= 'b0;
        end
        2'b01:
        begin
            led[0] <= 'b0;
            led[1] <= pwm_out;
            led[2] <= 'b0;
            led[3] <= 'b0;
        end
        2'b10:
        begin
            led[0] <= 'b0;
            led[1] <= 'b0;
            led[2] <= pwm_out;
            led[3] <= 'b0;
        end
        2'b11:
        begin
            led[0] <= 'b0;
            led[1] <= 'b0;
            led[2] <= 'b0;
            led[3] <= pwm_out;
        end
    endcase
end

endmodule
```

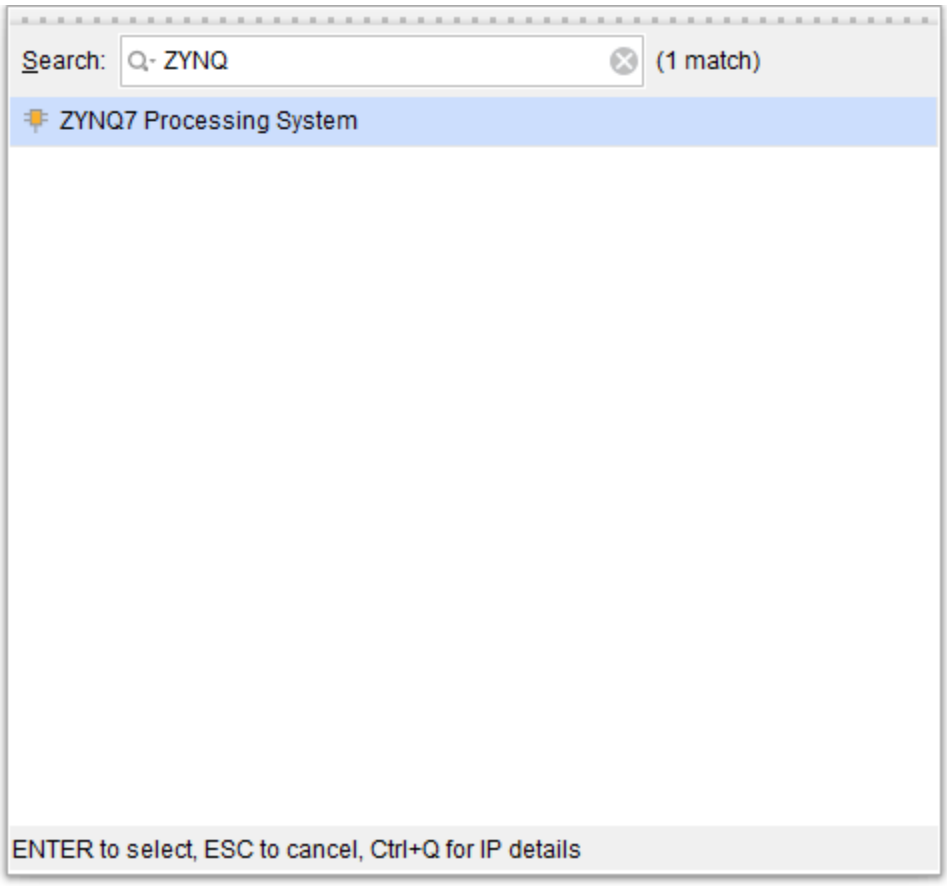
固件固化之前要创建一个bd文件,从IP INTEGRATOR→Create Block Design,创建一个名字默认就行.



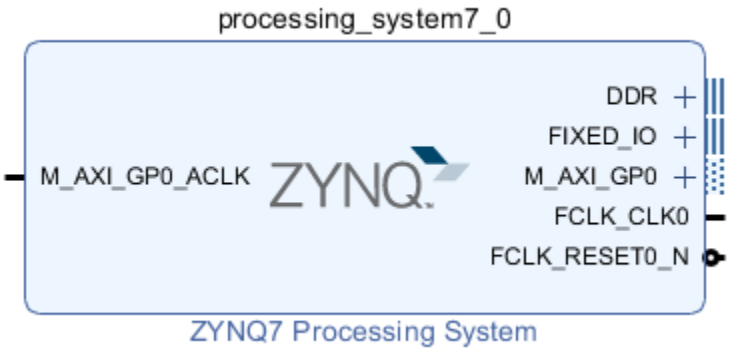
进入文件后新建IP.



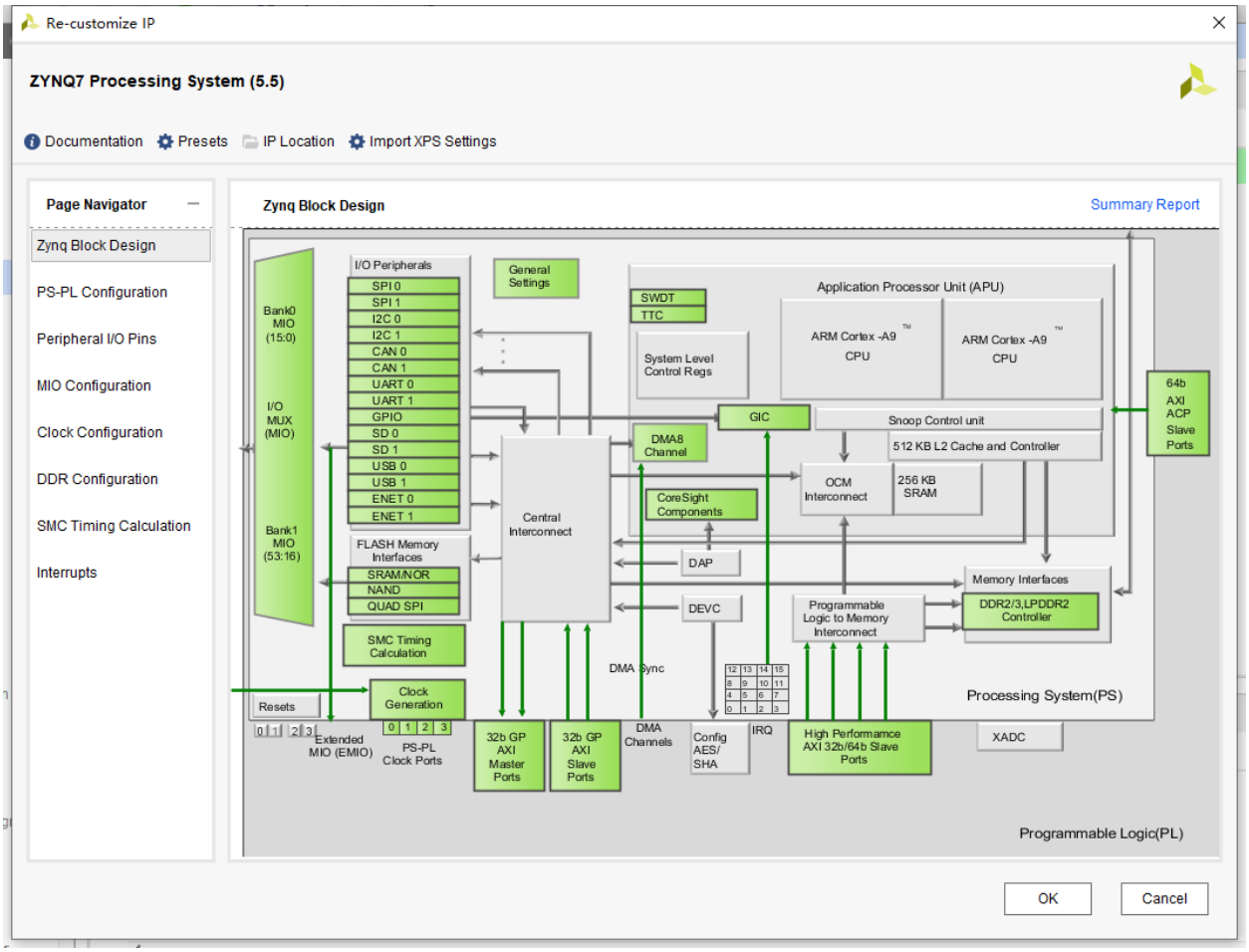
添加一个ZYNQ片上系统.


Search:
ZYNQ
(1 match)
ZYNQ7 Processing System
ENTER to select, ESC to cancel, Ctrl+Q for IP details

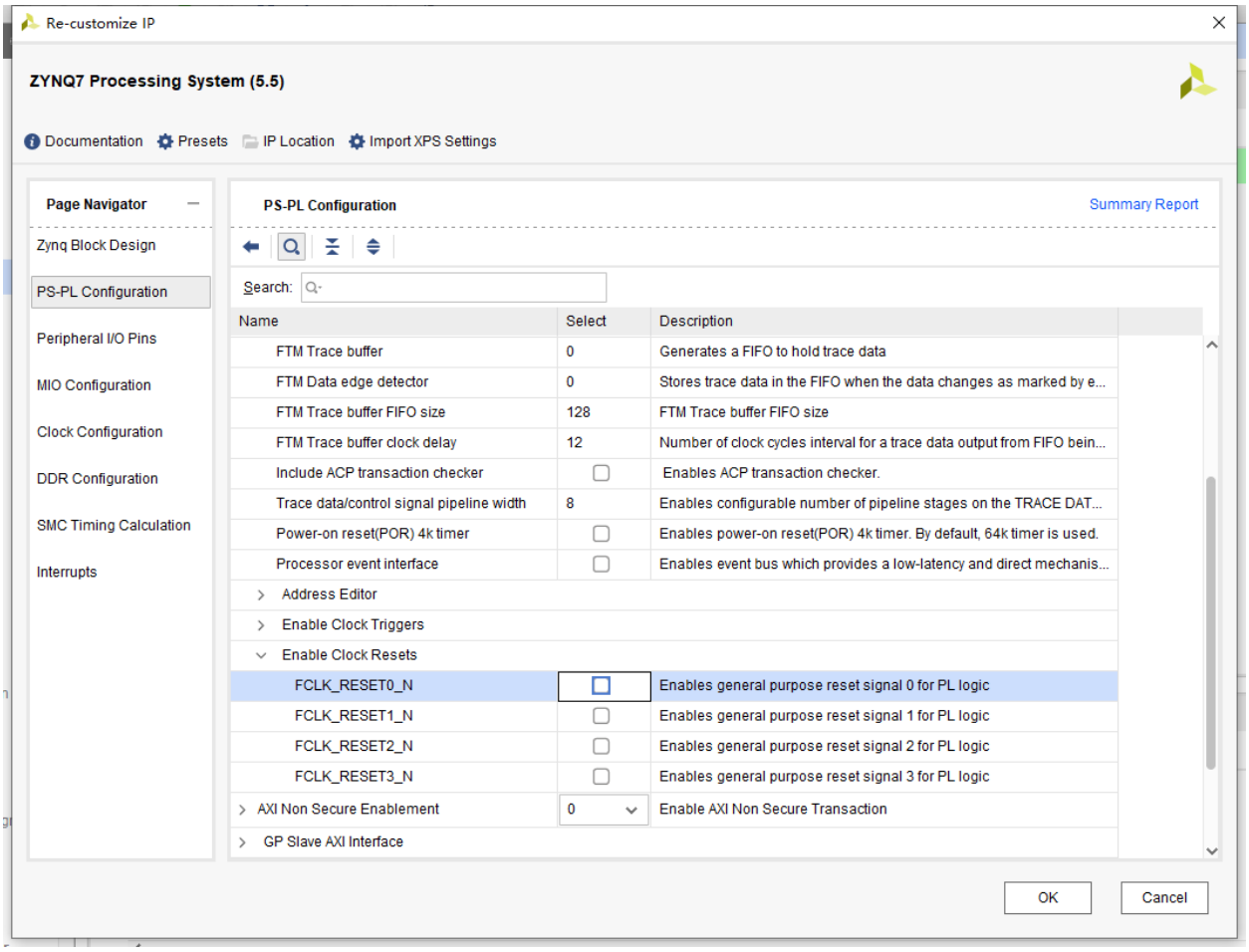
双击模块.


processing_system7_0
M_AXI_GP0_ACLK
ZYNQ
DDR
FIXED_IO
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System

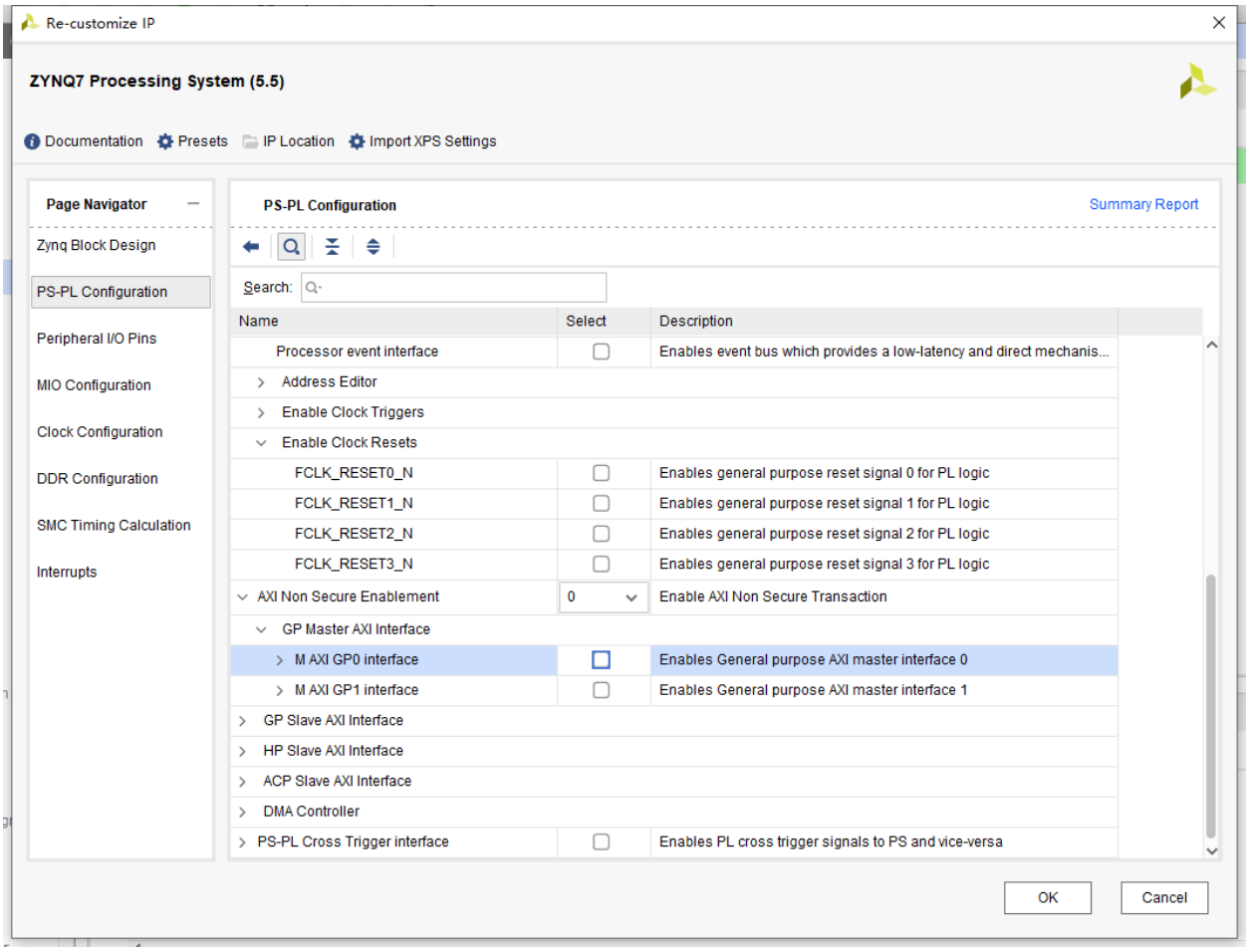
进入IP配置.



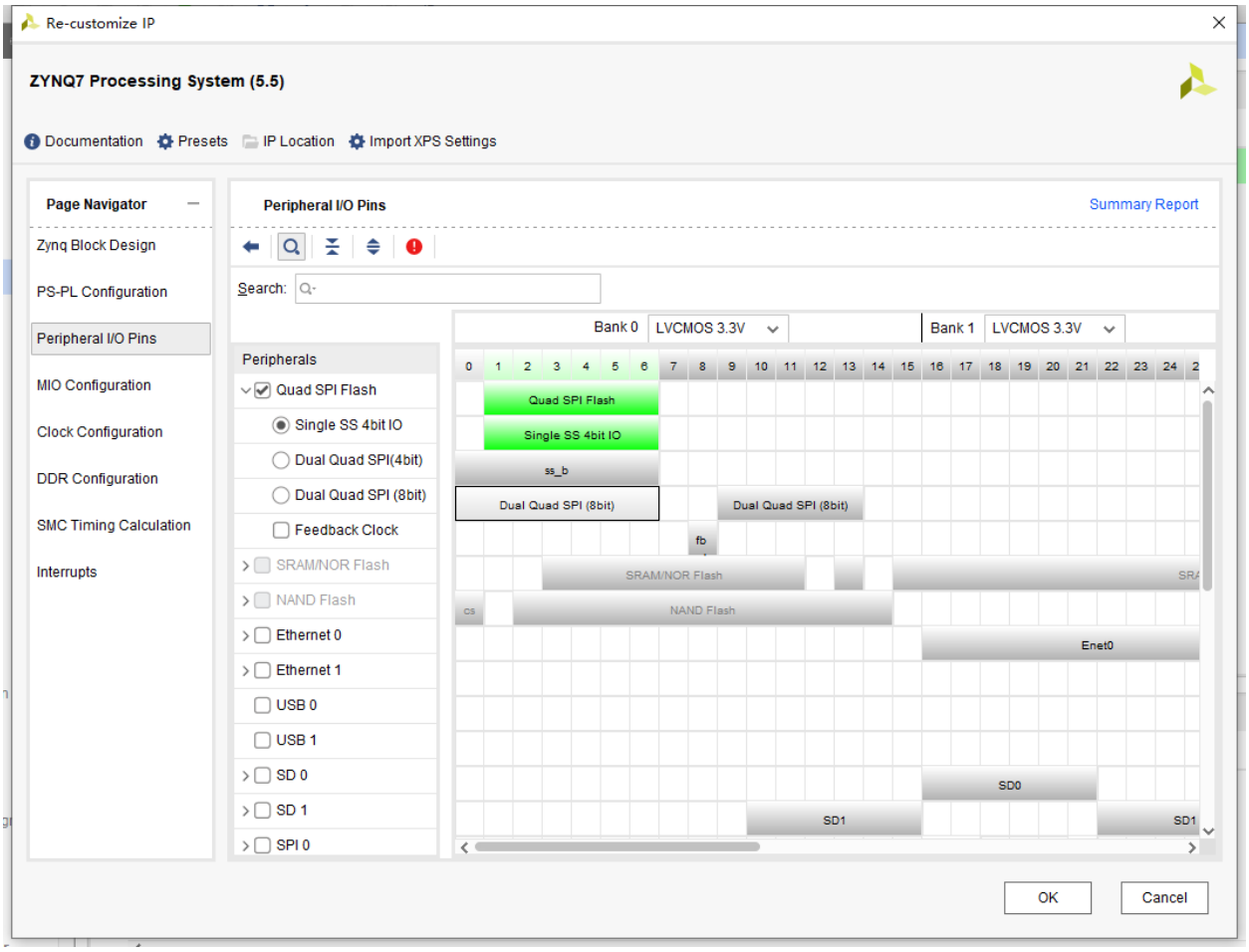
进入PS-PL configration→General→Enable Clock Resets,禁用FCLK_RESET0_N.



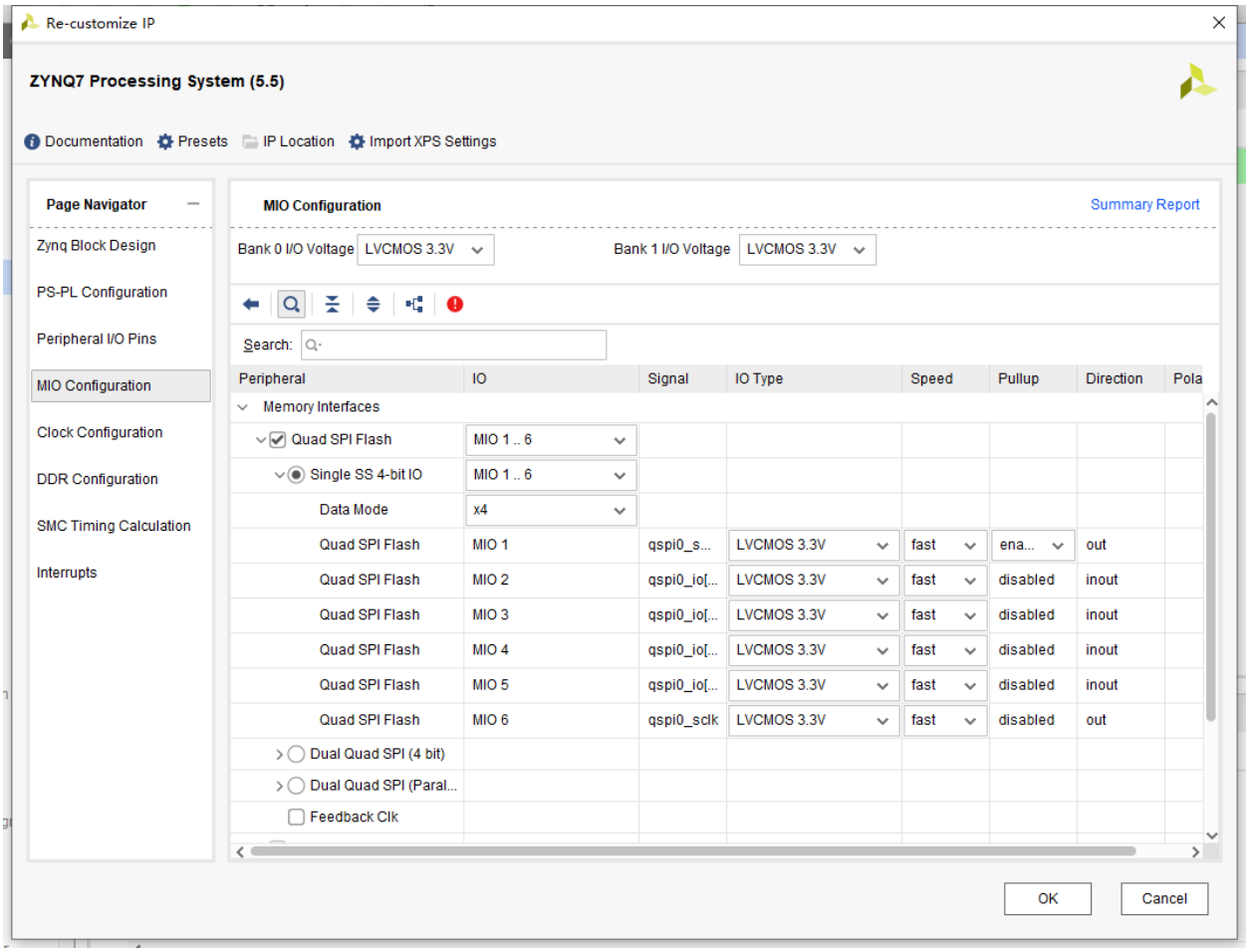
进入PS-PL configration→AXI Non Secure Enablement→GP Master AXI Interface,禁用M AXI GP0 Interface.



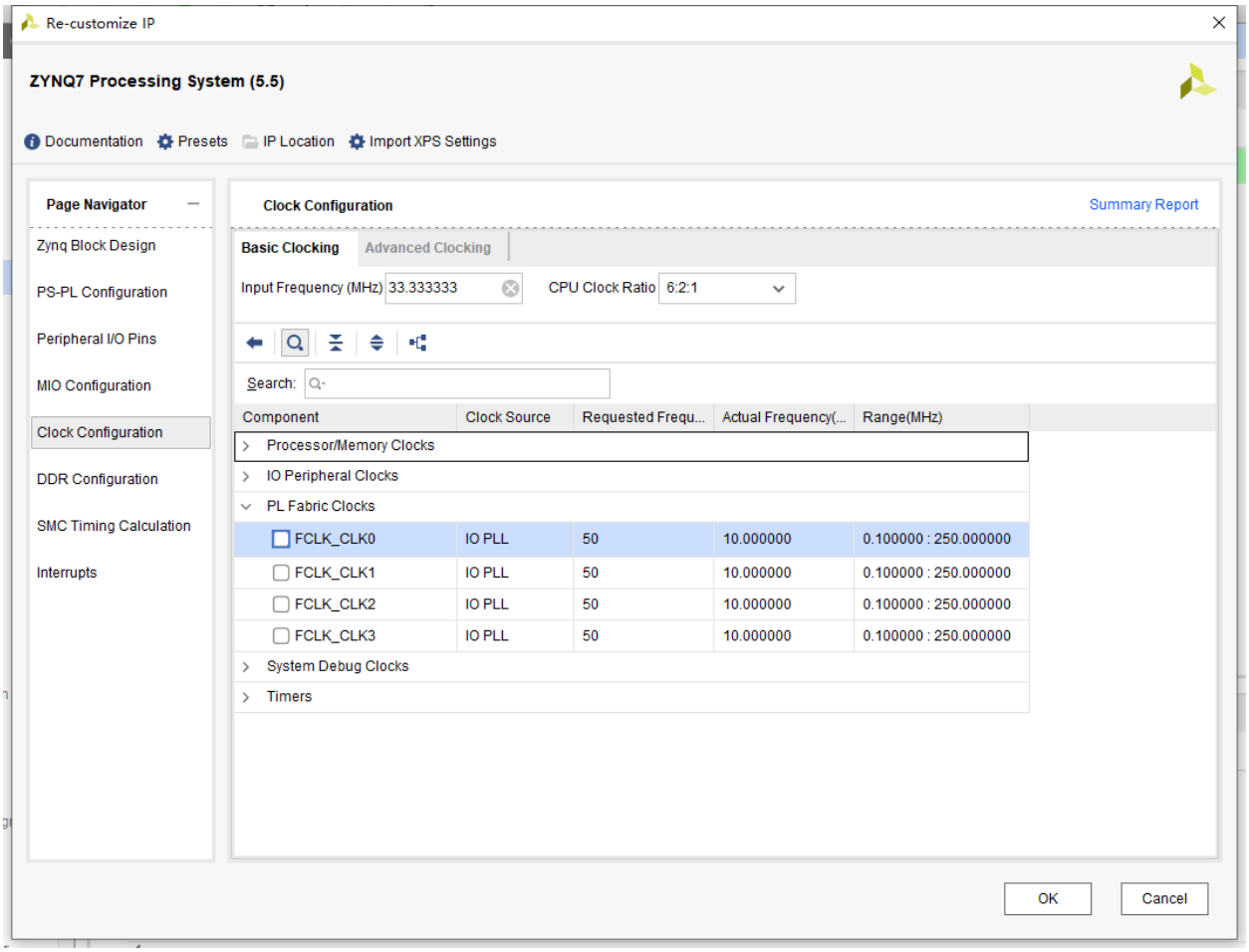
进入Peripheral I/O Pins并使能Quad SPI Flash的Single SS 4bit IO.



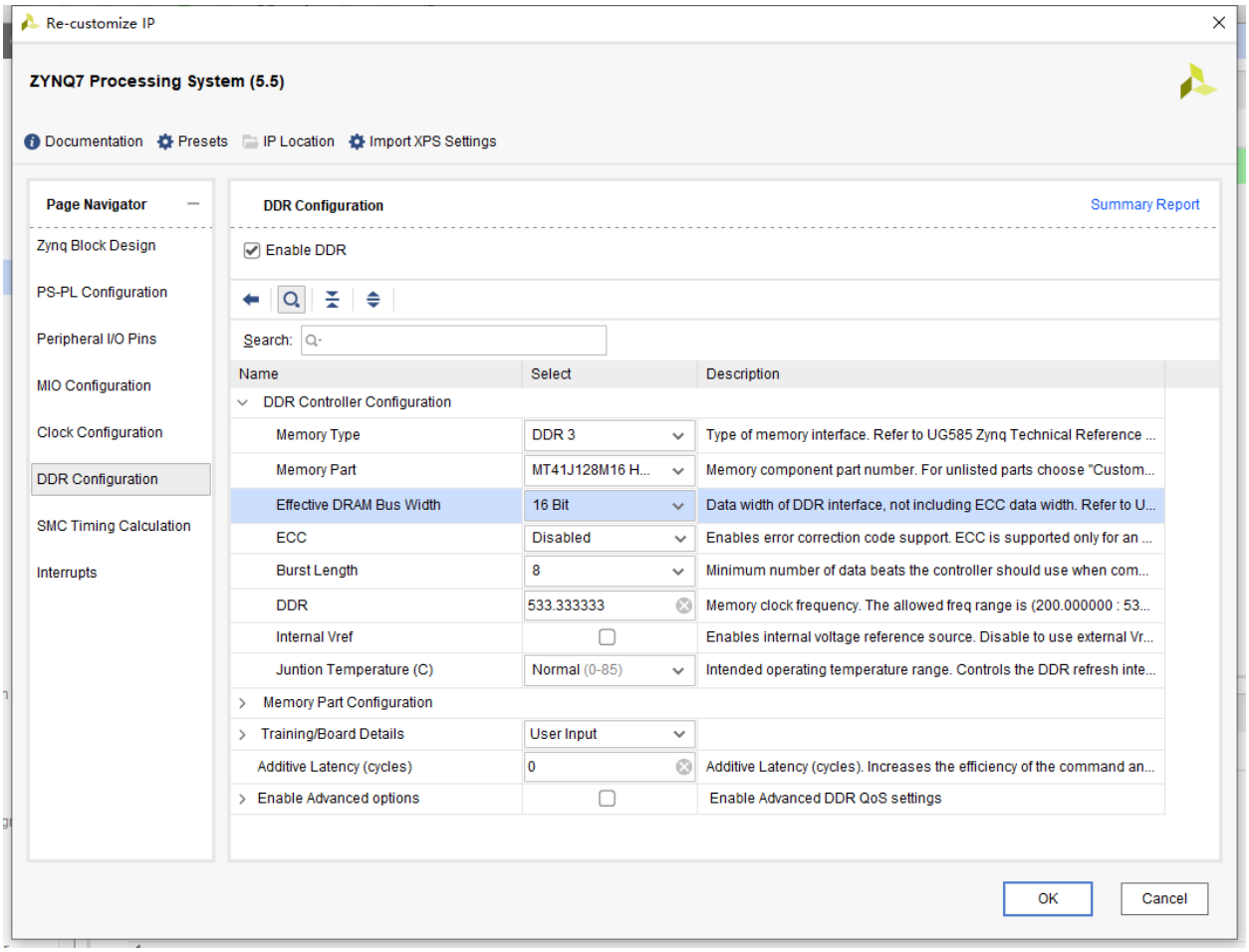
在MIO Configuration→Memory Interfaces→Quad SPI Flash→Single SS 4-bit IO改IO Speed为fast.



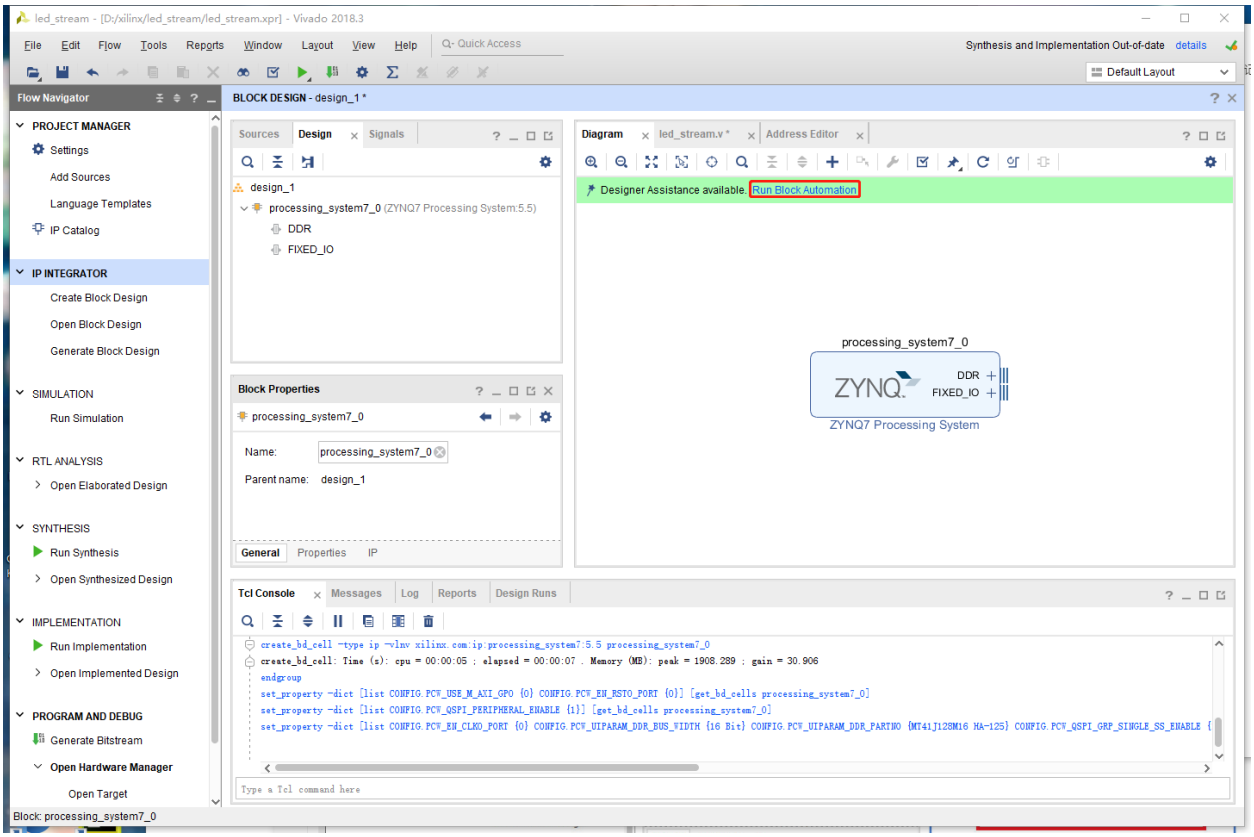
在Clock Configuration→PL Fabric Clock禁用FCLK_CLK0.



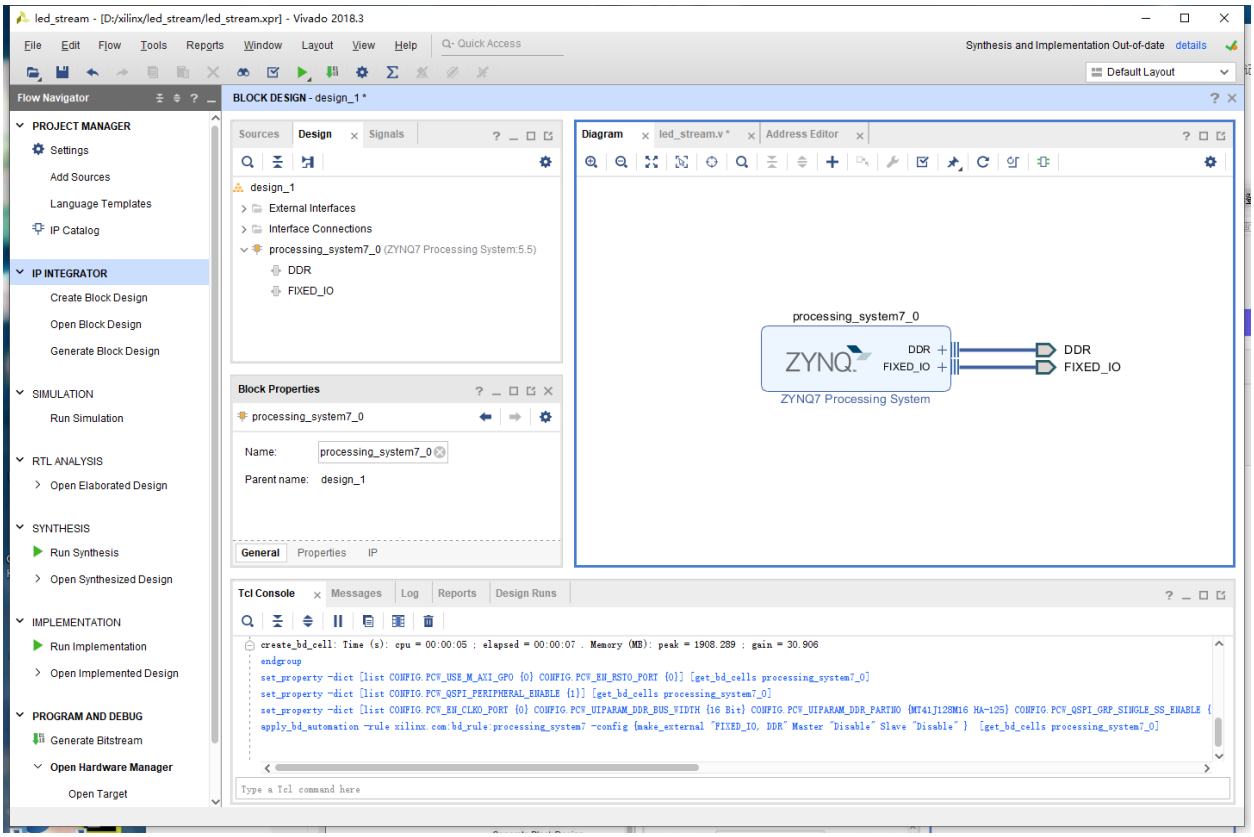
在DDR Configuration→DDR Controller Configuration→Memory Part选MT41J128M16 HA-125,Effective DRAM Bus Width选16Bit.



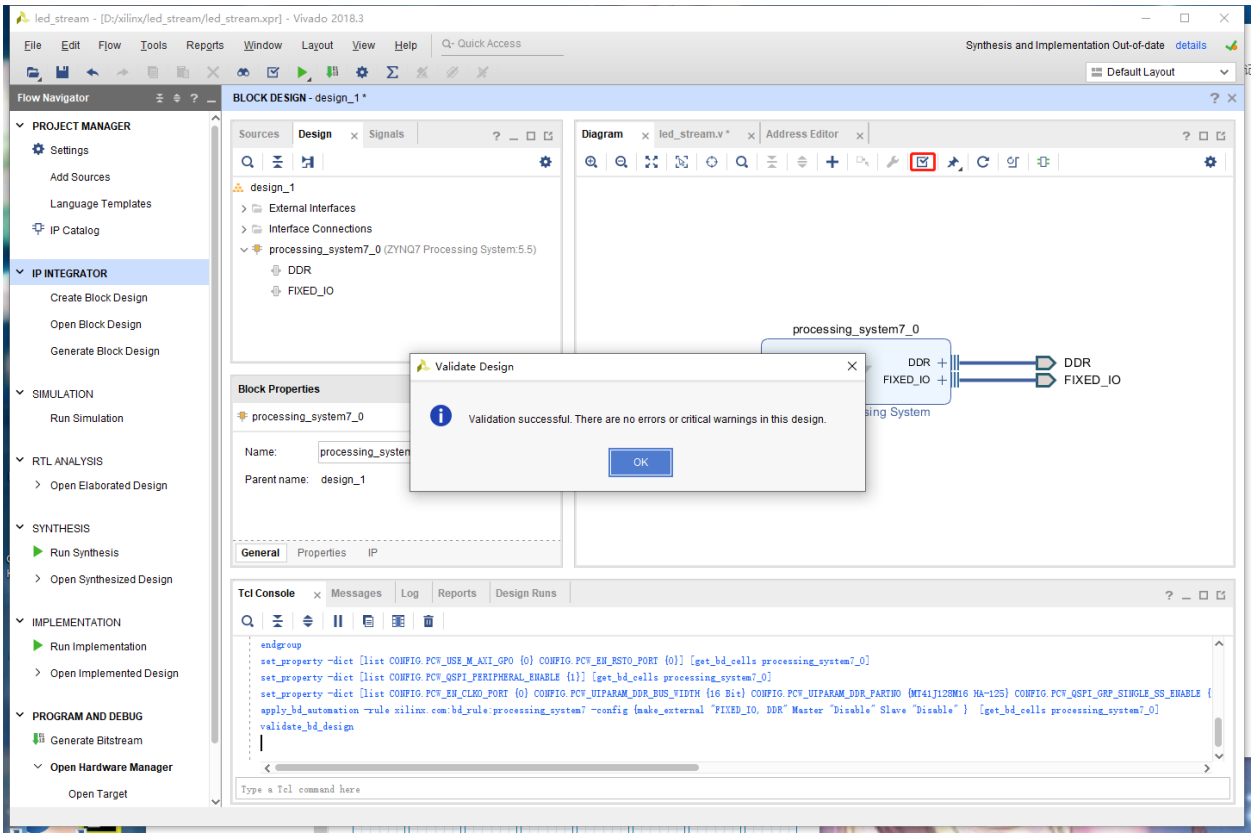
执行Run Block Automation并保持默认.



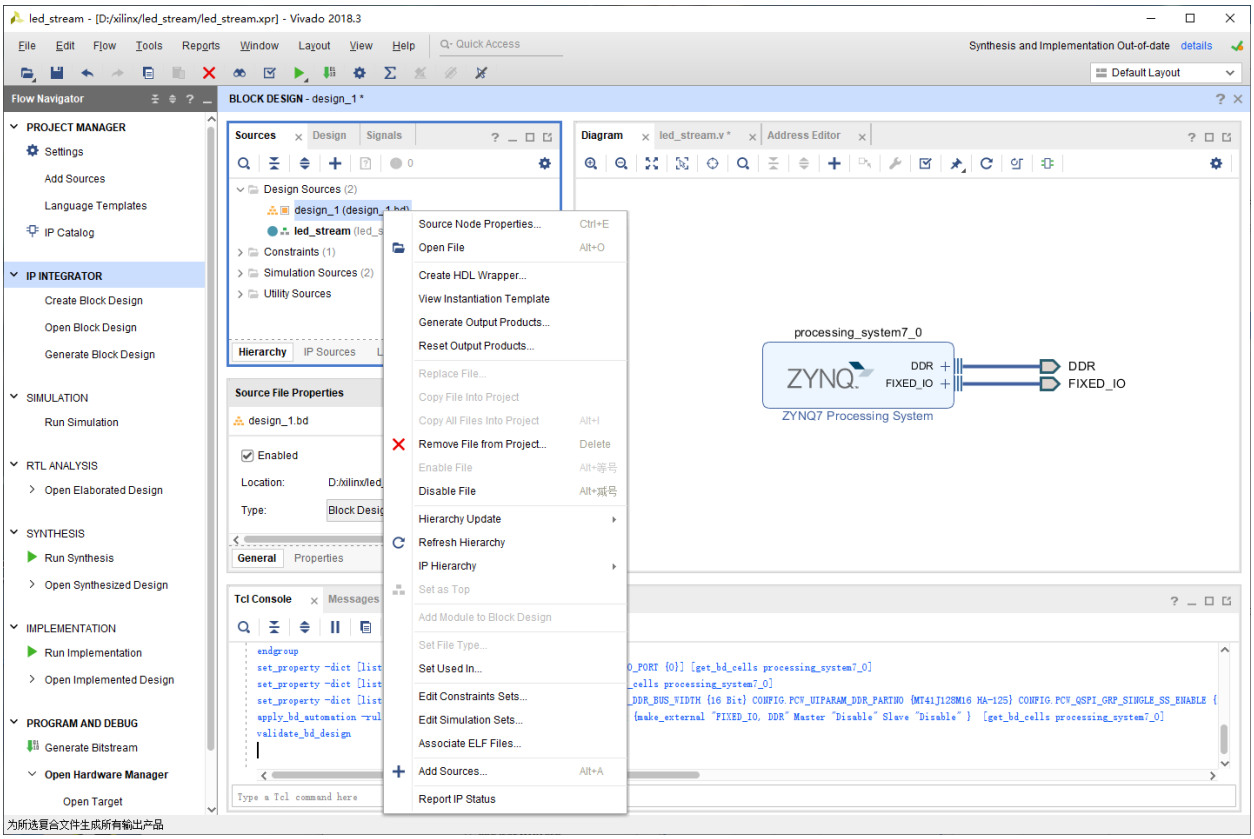
之后区块就剩余DDR和FIXED_IO了,并且是引出状态.



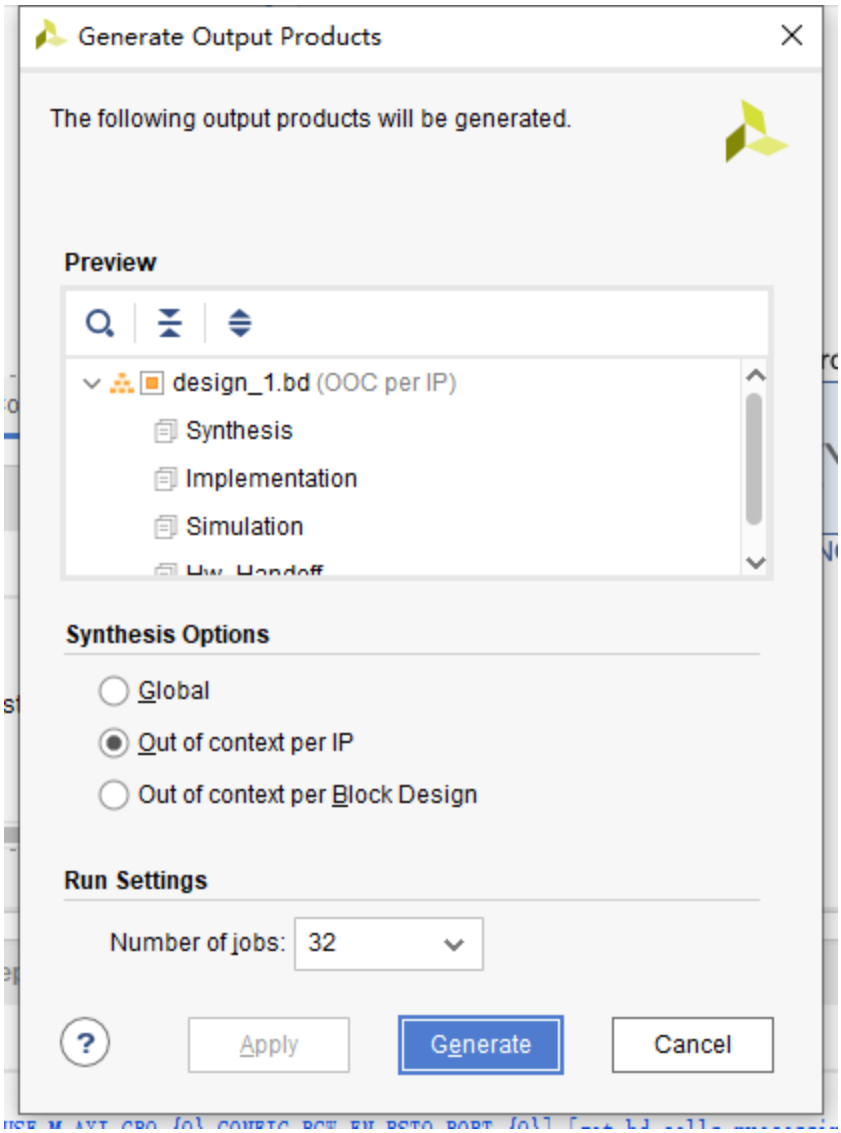
点击验证确保验证OK,每次调整IP后最好都验证一下.



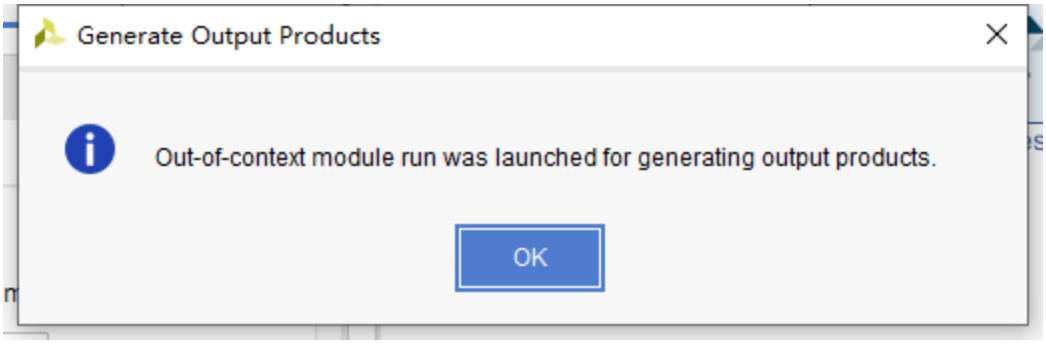
选择设计文件右键的Generate Output Products.


led_stream - [D:/xilinx/led_stream/led_stream.xpr] - Vivado 2018.3
File Edit Flow Tools Reports Window Layout View Help
Quick Access
Synthesis and Implementation Out-of-date details
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Open Target
BLOCK DESIGN - design_1 *
Sources
Design
Signals
Design Sources (2)
design_1 (design_1.bd)
led_stream (led_s
Constraints (1)
Simulation Sources (2)
Utility Sources
Hierarchy
IP Sources
Source File Properties
design_1.bd
Enabled
Location:
Type:
General
Properties
Source Node Properties... Ctrl+E
Open File Alt+O
Create HDL Wrapper...
View Instantiation Template
Generate Output Products...
Reset Output Products...
Replace File...
Copy File Into Project
Copy All Files Into Project Alt+I
Remove File from Project... Delete
Enable File Alt+等号
Disable File Alt+减号
Hierarchy Update
Refresh Hierarchy
IP Hierarchy
Set as Top
Add Module to Block Design
Set File Type...
Set Used In...
Edit Constraints Sets...
Edit Simulation Sets...
Associate ELF Files...
Add Sources... Alt+A
Report IP Status
Diagram
led_stream.v *
Address Editor
processing_system7_0
ZYNQ
DDR
FIXED_IO
ZYNQ7 Processing System
Tcl Console
Messages
endgroup
set_property -dict [list
set_property -dict [list
set_property -dict [list
apply_bd_automation -rul
validate_bd_design
Type a Tcl command here
为所选复合文件生成所有输出产品

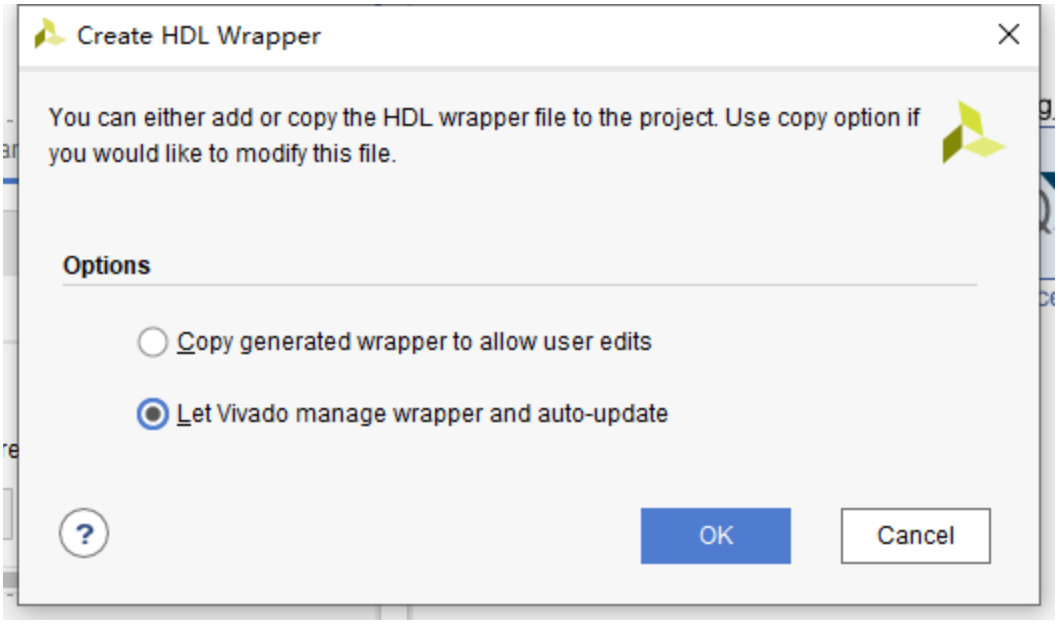
保持默认就可以.



很快就生成出来了.



继续在设计文件右键选Create HDL Wrapper,然后保持默认就可以.



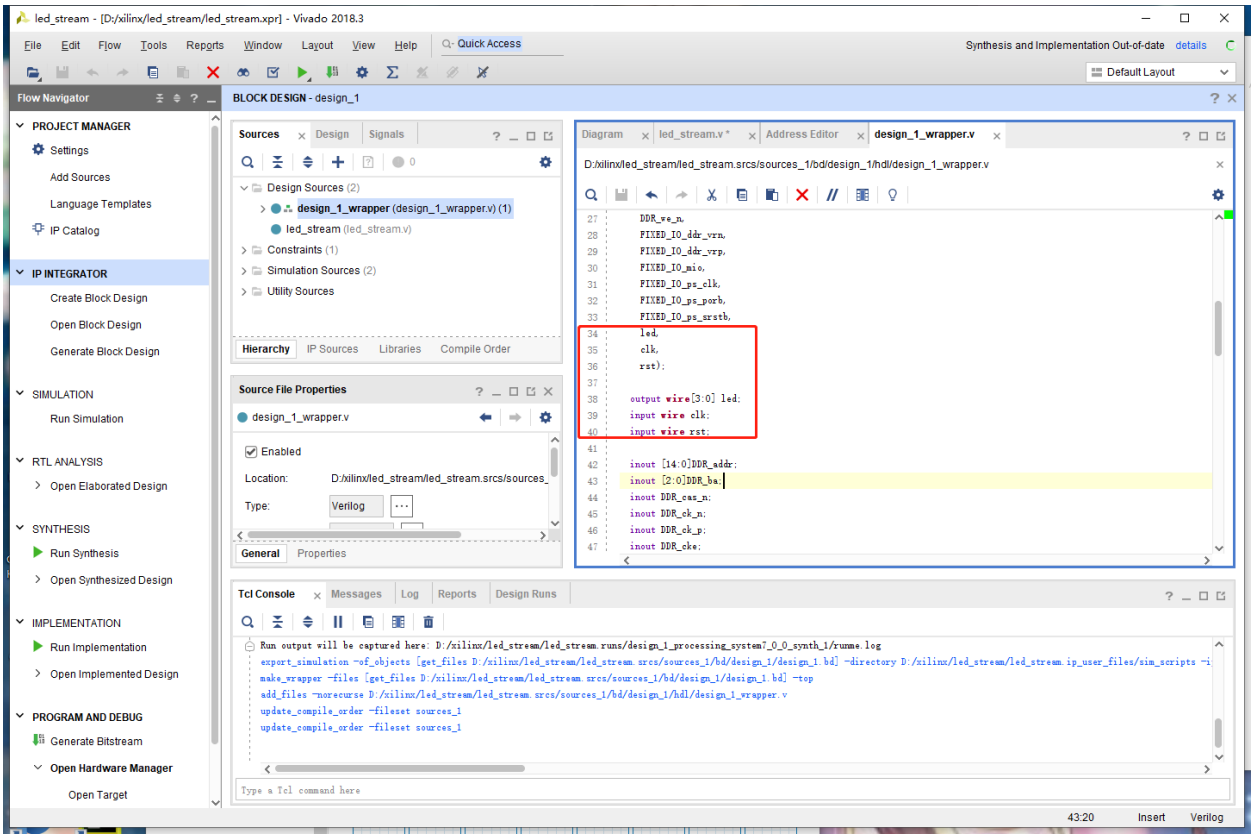
现在的设计文件是这样的.

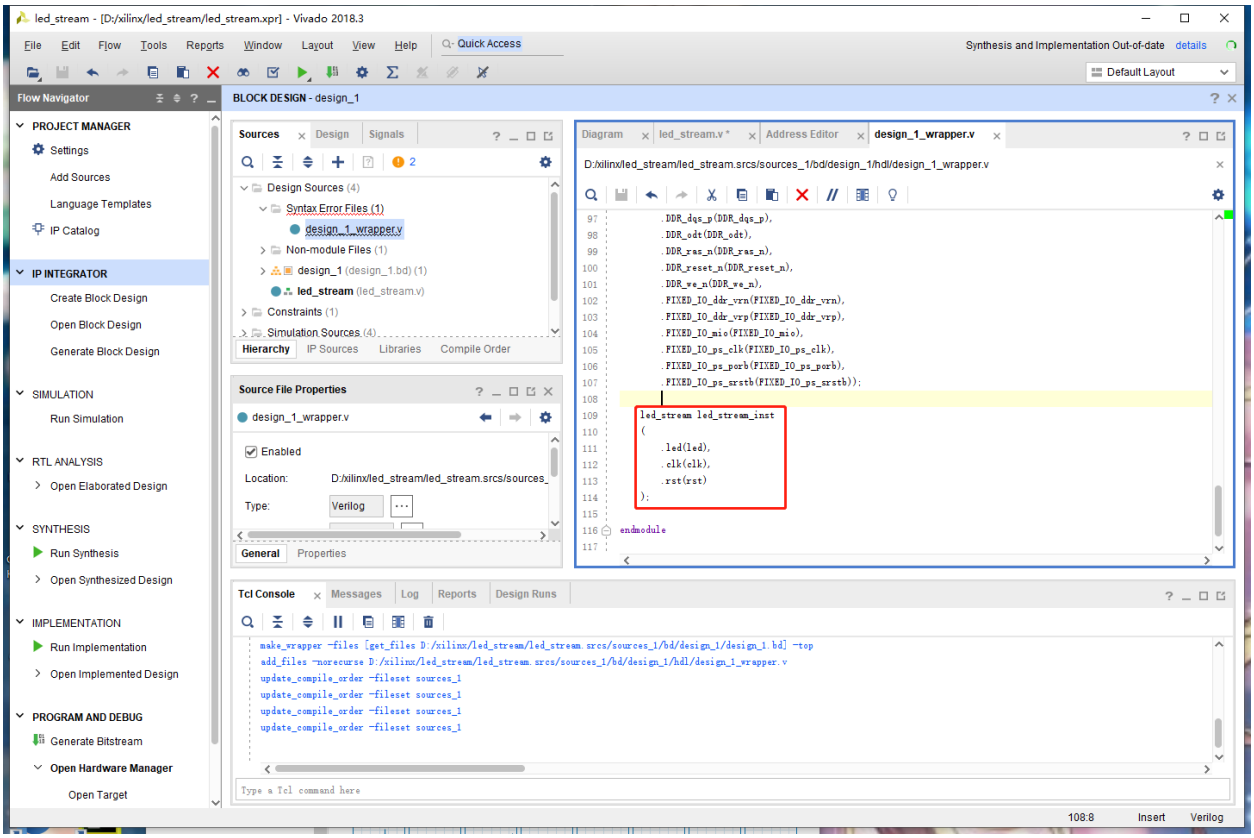
```
//Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
//----------------------------------------------------------------------------------
//Tool Version: Vivado v.2018.3 (win64) Build 2405991 Thu Dec  6 23:38:27 MST 2018
//Date        : Tue Feb 22 15:05:39 2022
//Host        : DESKTOP-JJ3MLC5 running 64-bit major release  (build 9200)
//Command     : generate_target design_1_wrapper.bd
//Design      : design_1_wrapper
//Purpose     : IP block netlist
//----------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module design_1_wrapper
   (DDR_addr,
    DDR_ba,
    DDR_cas_n,
    DDR_ck_n,
    DDR_ck_p,
    DDR_cke,
    DDR_cs_n,
    DDR_dm,
    DDR_dq,
    DDR_dqs_n,
    DDR_dqs_p,
    DDR_odt,
    DDR_ras_n,
    DDR_reset_n,
    DDR_we_n,
    FIXED_IO_ddr_vrn,
    FIXED_IO_ddr_vrp,
```

```
    FIXED_IO_mio,
    FIXED_IO_ps_clk,
    FIXED_IO_ps_porb,
    FIXED_IO_ps_srstb);
  inout [14:0]DDR_addr;
  inout [2:0]DDR_ba;
  inout DDR_cas_n;
  inout DDR_ck_n;
  inout DDR_ck_p;
  inout DDR_cke;
  inout DDR_cs_n;
  inout [3:0]DDR_dm;
  inout [31:0]DDR_dq;
  inout [3:0]DDR_dqs_n;
  inout [3:0]DDR_dqs_p;
  inout DDR_odt;
  inout DDR_ras_n;
  inout DDR_reset_n;
  inout DDR_we_n;
  inout FIXED_IO_ddr_vrn;
  inout FIXED_IO_ddr_vrp;
  inout [53:0]FIXED_IO_mio;
  inout FIXED_IO_ps_clk;
  inout FIXED_IO_ps_porb;
  inout FIXED_IO_ps_srstb;

  wire [14:0]DDR_addr;
  wire [2:0]DDR_ba;
  wire DDR_cas_n;
  wire DDR_ck_n;
  wire DDR_ck_p;
  wire DDR_cke;
  wire DDR_cs_n;
  wire [3:0]DDR_dm;
  wire [31:0]DDR_dq;
  wire [3:0]DDR_dqs_n;
  wire [3:0]DDR_dqs_p;
  wire DDR_odt;
  wire DDR_ras_n;
  wire DDR_reset_n;
  wire DDR_we_n;
  wire FIXED_IO_ddr_vrn;
  wire FIXED_IO_ddr_vrp;
  wire [53:0]FIXED_IO_mio;
  wire FIXED_IO_ps_clk;
  wire FIXED_IO_ps_porb;
  wire FIXED_IO_ps_srstb;

  design_1 design_1_i
       (.DDR_addr(DDR_addr),
        .DDR_ba(DDR_ba),
        .DDR_cas_n(DDR_cas_n),
        .DDR_ck_n(DDR_ck_n),
        .DDR_ck_p(DDR_ck_p),
```

```
        .DDR_cke(DDR_cke),
        .DDR_cs_n(DDR_cs_n),
        .DDR_dm(DDR_dm),
        .DDR_dq(DDR_dq),
        .DDR_dqs_n(DDR_dqs_n),
        .DDR_dqs_p(DDR_dqs_p),
        .DDR_odt(DDR_odt),
        .DDR_ras_n(DDR_ras_n),
        .DDR_reset_n(DDR_reset_n),
        .DDR_we_n(DDR_we_n),
        .FIXED_IO_ddr_vrn(FIXED_IO_ddr_vrn),
        .FIXED_IO_ddr_vrp(FIXED_IO_ddr_vrp),
        .FIXED_IO_mio(FIXED_IO_mio),
        .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
        .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
        .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb));
endmodule
```

实例化一下模块,需要添加IO声明,然后在文件末尾实例化一下.





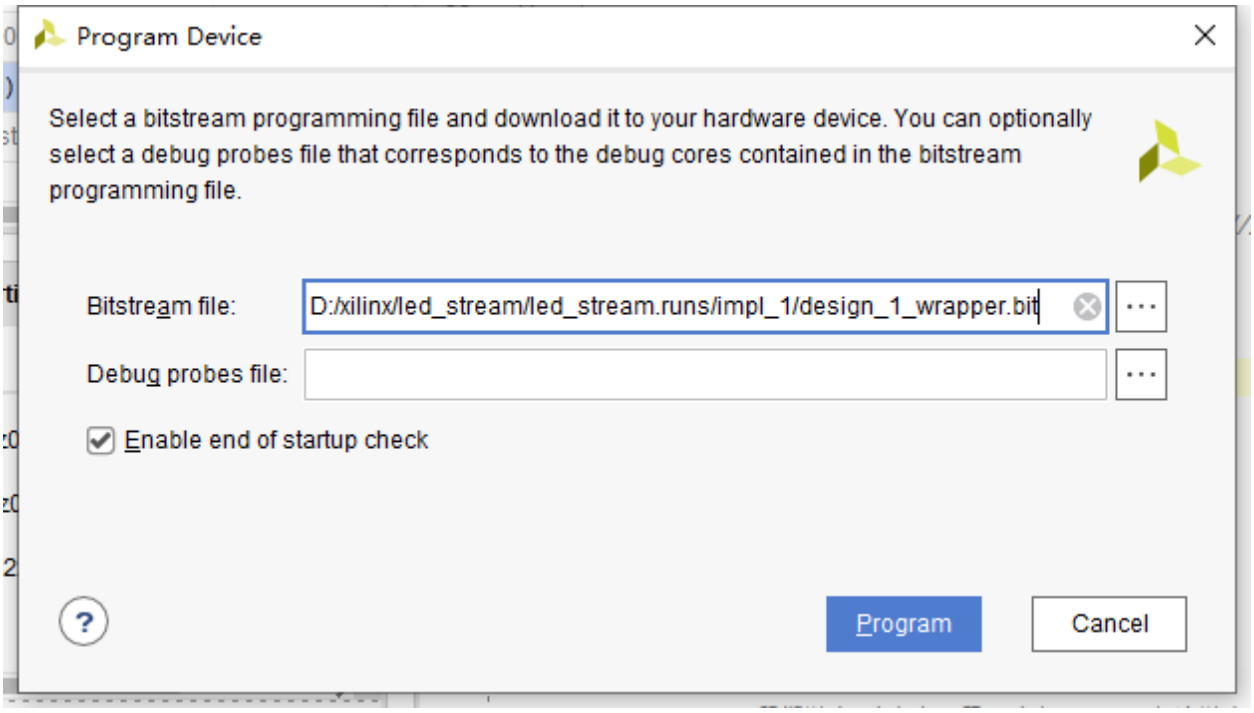
然后文件就变成这样,然后就要综合,综合,综合,这个非常重要,否则会让后续步骤失败.

```
//Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
//----------------------------------------------------------------------------------
//Tool Version: Vivado v.2018.3 (win64) Build 2405991 Thu Dec  6 23:38:27 MST 2018
//Date        : Tue Feb 22 15:05:39 2022
//Host        : DESKTOP-JJ3MLC5 running 64-bit major release  (build 9200)
//Command     : generate_target design_1_wrapper.bd
//Design      : design_1_wrapper
//Purpose     : IP block netlist
//----------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module design_1_wrapper
   (DDR_addr,
    DDR_ba,
    DDR_cas_n,
    DDR_ck_n,
    DDR_ck_p,
    DDR_cke,
    DDR_cs_n,
    DDR_dm,
    DDR_dq,
    DDR_dqs_n,
    DDR_dqs_p,
```

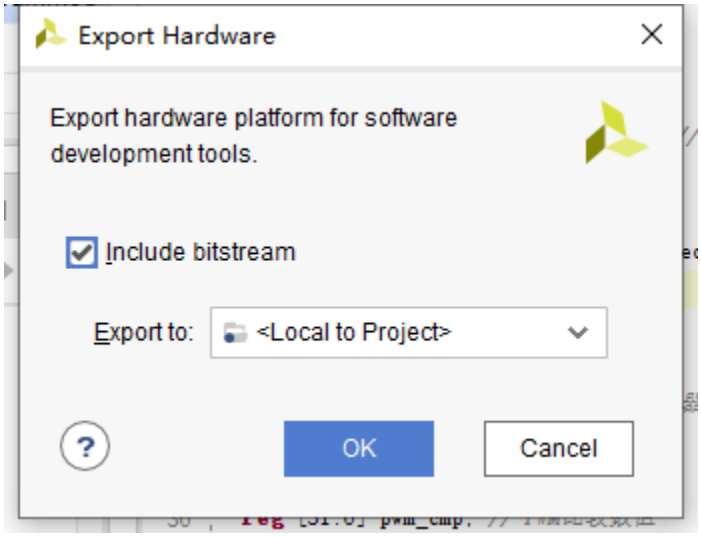
```
    DDR_odt,
    DDR_ras_n,
    DDR_reset_n,
    DDR_we_n,
    FIXED_IO_ddr_vrn,
    FIXED_IO_ddr_vrp,
    FIXED_IO_mio,
    FIXED_IO_ps_clk,
    FIXED_IO_ps_porb,
    FIXED_IO_ps_srstb,
    led,
    clk,
    rst);

  output wire[3:0] led;
  input wire clk;
  input wire rst;

  inout [14:0]DDR_addr;
  inout [2:0]DDR_ba;
  inout DDR_cas_n;
  inout DDR_ck_n;
  inout DDR_ck_p;
  inout DDR_cke;
  inout DDR_cs_n;
  inout [3:0]DDR_dm;
  inout [31:0]DDR_dq;
  inout [3:0]DDR_dqs_n;
  inout [3:0]DDR_dqs_p;
  inout DDR_odt;
  inout DDR_ras_n;
  inout DDR_reset_n;
  inout DDR_we_n;
  inout FIXED_IO_ddr_vrn;
  inout FIXED_IO_ddr_vrp;
  inout [53:0]FIXED_IO_mio;
  inout FIXED_IO_ps_clk;
  inout FIXED_IO_ps_porb;
  inout FIXED_IO_ps_srstb;

  wire [14:0]DDR_addr;
  wire [2:0]DDR_ba;
  wire DDR_cas_n;
  wire DDR_ck_n;
  wire DDR_ck_p;
  wire DDR_cke;
  wire DDR_cs_n;
  wire [3:0]DDR_dm;
  wire [31:0]DDR_dq;
  wire [3:0]DDR_dqs_n;
  wire [3:0]DDR_dqs_p;
  wire DDR_odt;
  wire DDR_ras_n;
  wire DDR_reset_n;
```

```
  wire DDR_we_n;
  wire FIXED_IO_ddr_vrn;
  wire FIXED_IO_ddr_vrp;
  wire [53:0]FIXED_IO_mio;
  wire FIXED_IO_ps_clk;
  wire FIXED_IO_ps_porb;
  wire FIXED_IO_ps_srstb;

  design_1 design_1_i
       (.DDR_addr(DDR_addr),
        .DDR_ba(DDR_ba),
        .DDR_cas_n(DDR_cas_n),
        .DDR_ck_n(DDR_ck_n),
        .DDR_ck_p(DDR_ck_p),
        .DDR_cke(DDR_cke),
        .DDR_cs_n(DDR_cs_n),
        .DDR_dm(DDR_dm),
        .DDR_dq(DDR_dq),
        .DDR_dqs_n(DDR_dqs_n),
        .DDR_dqs_p(DDR_dqs_p),
        .DDR_odt(DDR_odt),
        .DDR_ras_n(DDR_ras_n),
        .DDR_reset_n(DDR_reset_n),
        .DDR_we_n(DDR_we_n),
        .FIXED_IO_ddr_vrn(FIXED_IO_ddr_vrn),
        .FIXED_IO_ddr_vrp(FIXED_IO_ddr_vrp),
        .FIXED_IO_mio(FIXED_IO_mio),
        .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
        .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
        .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb));

    led_stream led_stream_inst
    (
        .led(led),
        .clk(clk),
        .rst(rst)
    );

endmodule
```

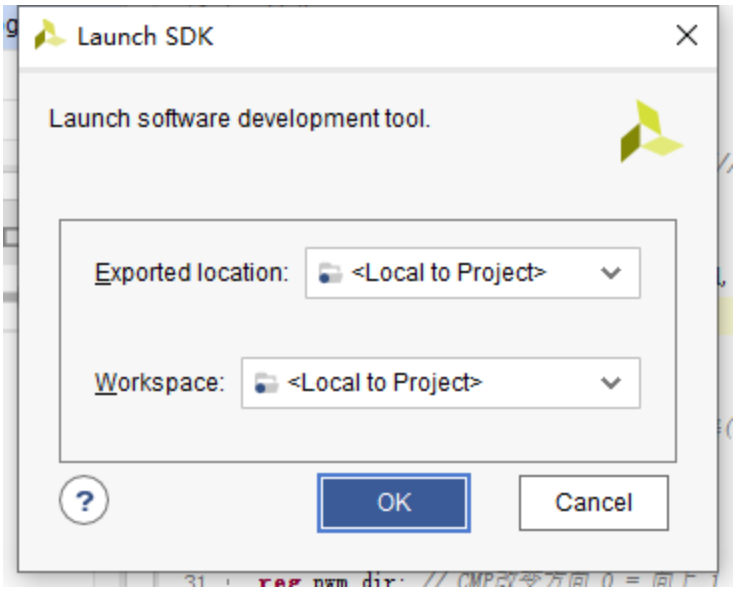
之后可以生成Bitstream文件,烧录确定没问题了.



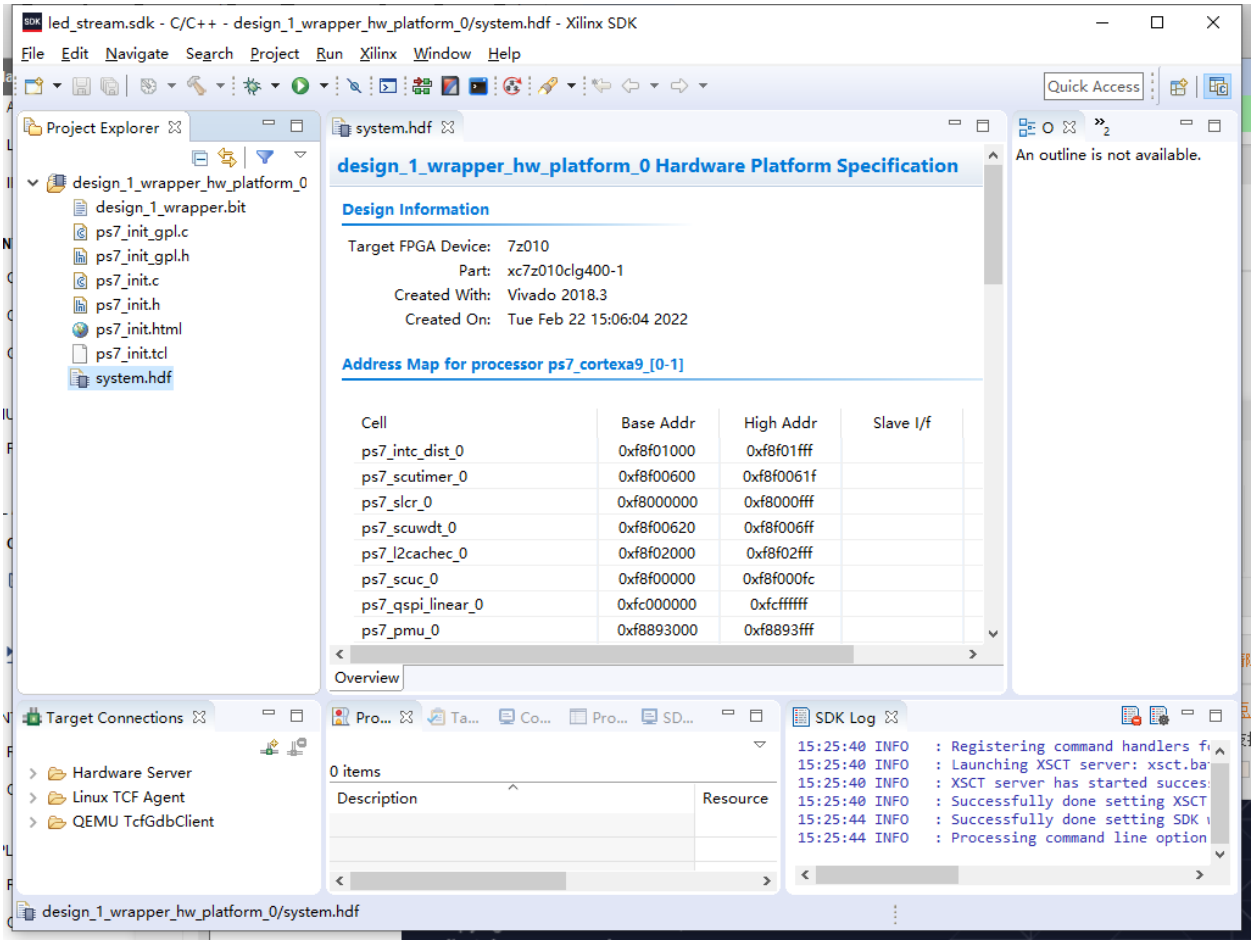
点击File→Export→Export Hardware,并勾选Include bitstream,生成一个工程文件.



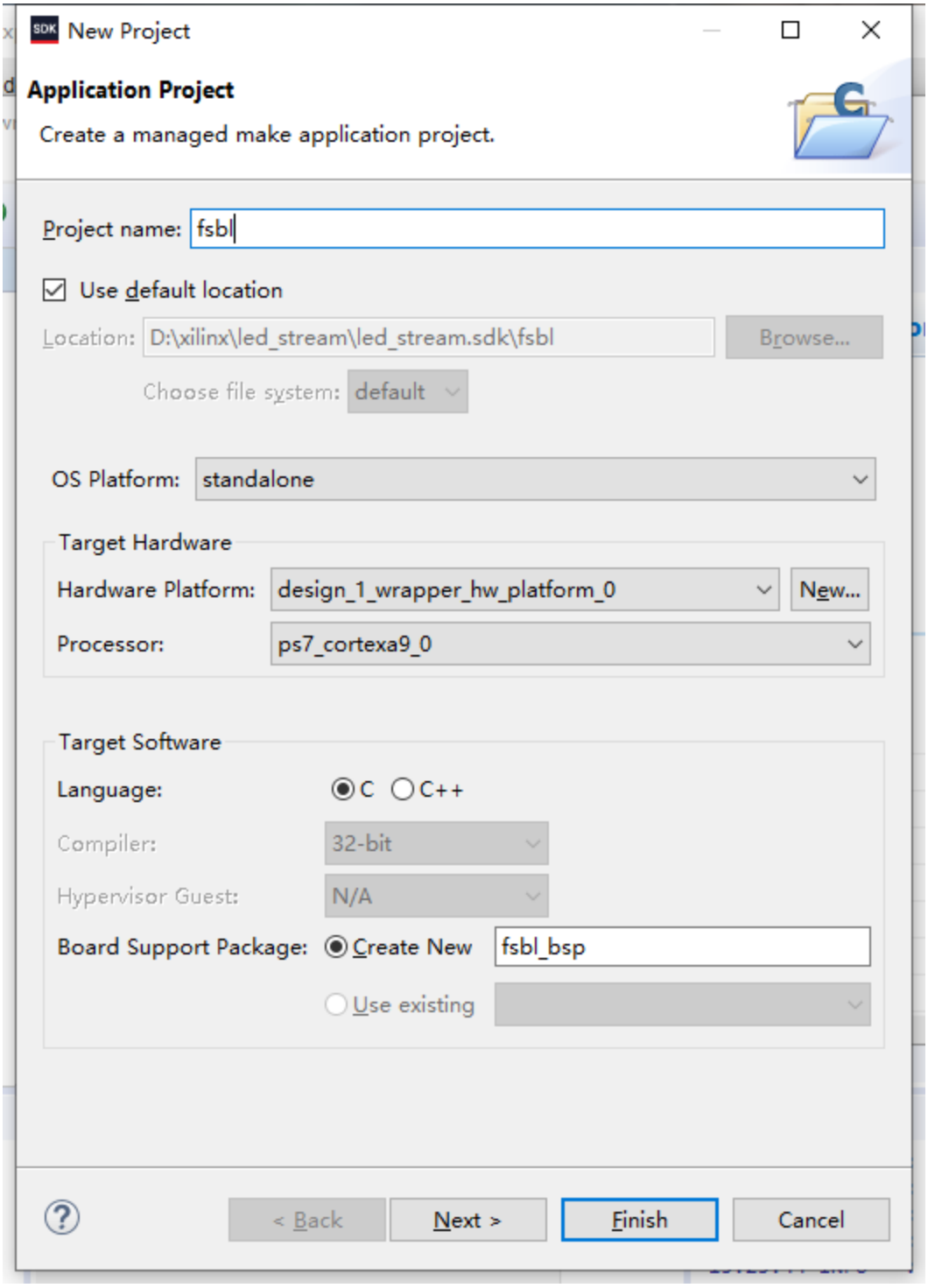
然后到File→Launch SDK.



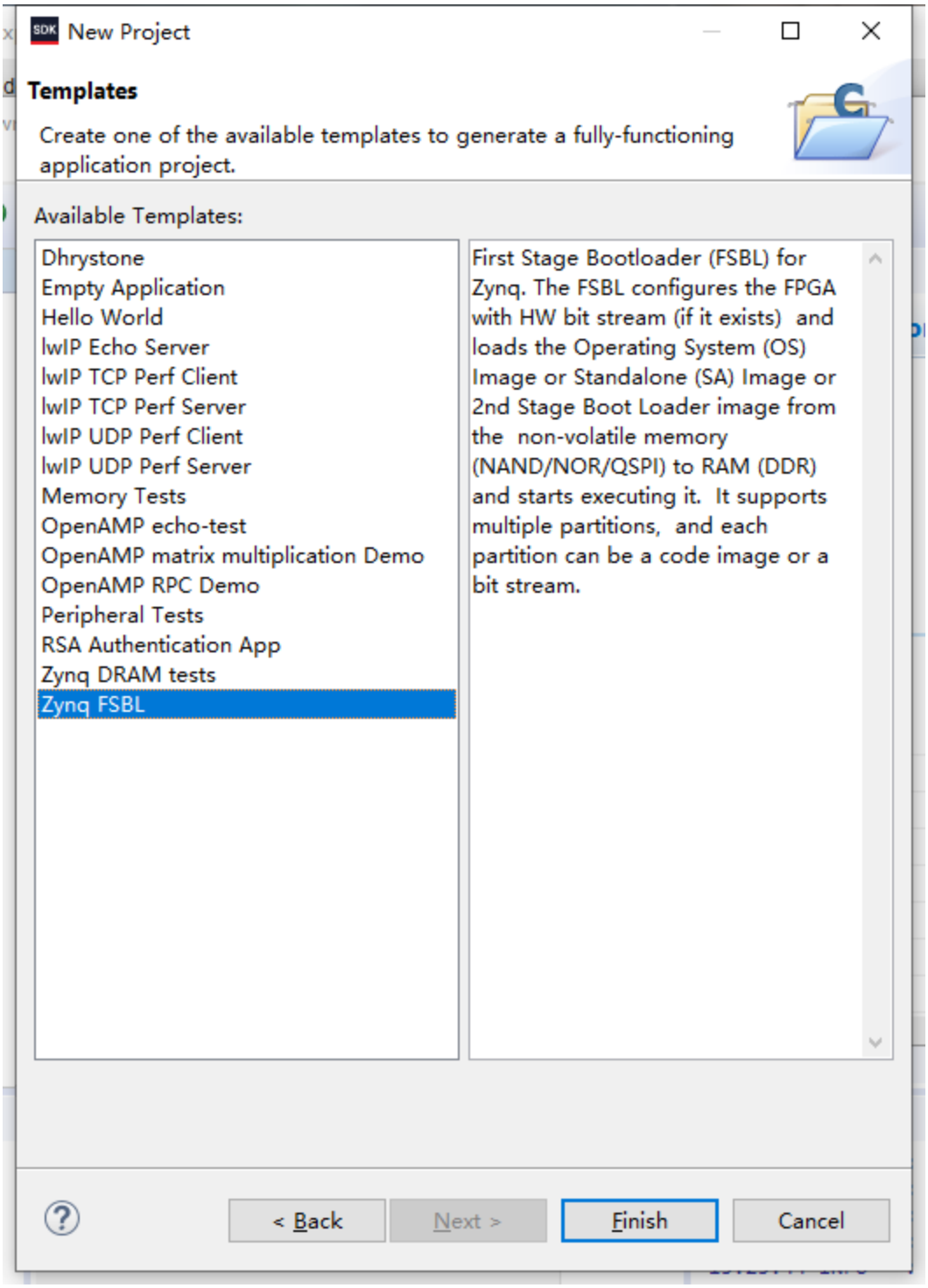
生成的工程中的hdf文件即硬件描述文件.



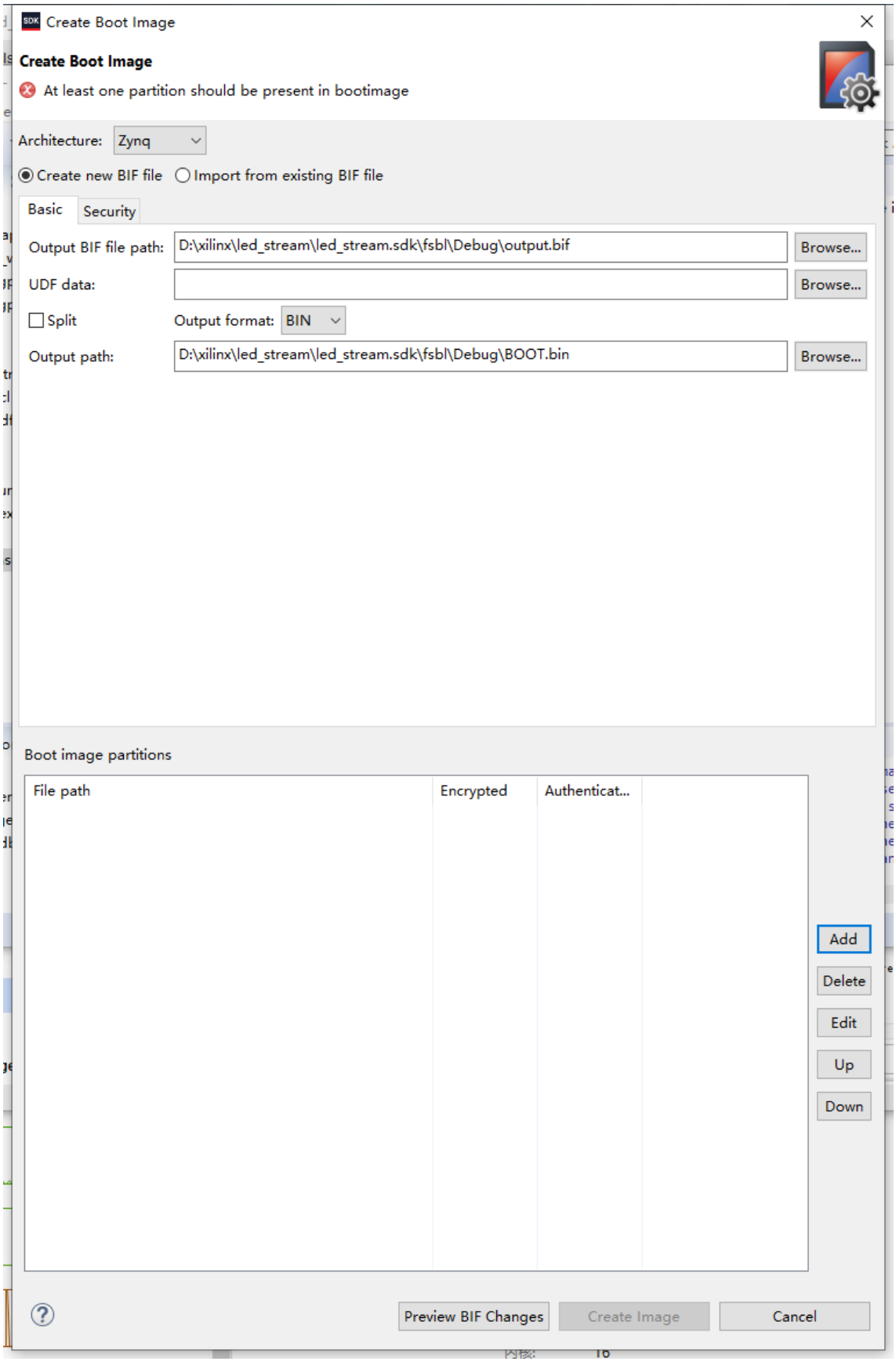
点击SDK软件的File→New→Application Project,工程名可以随便起,这里写fsbl.



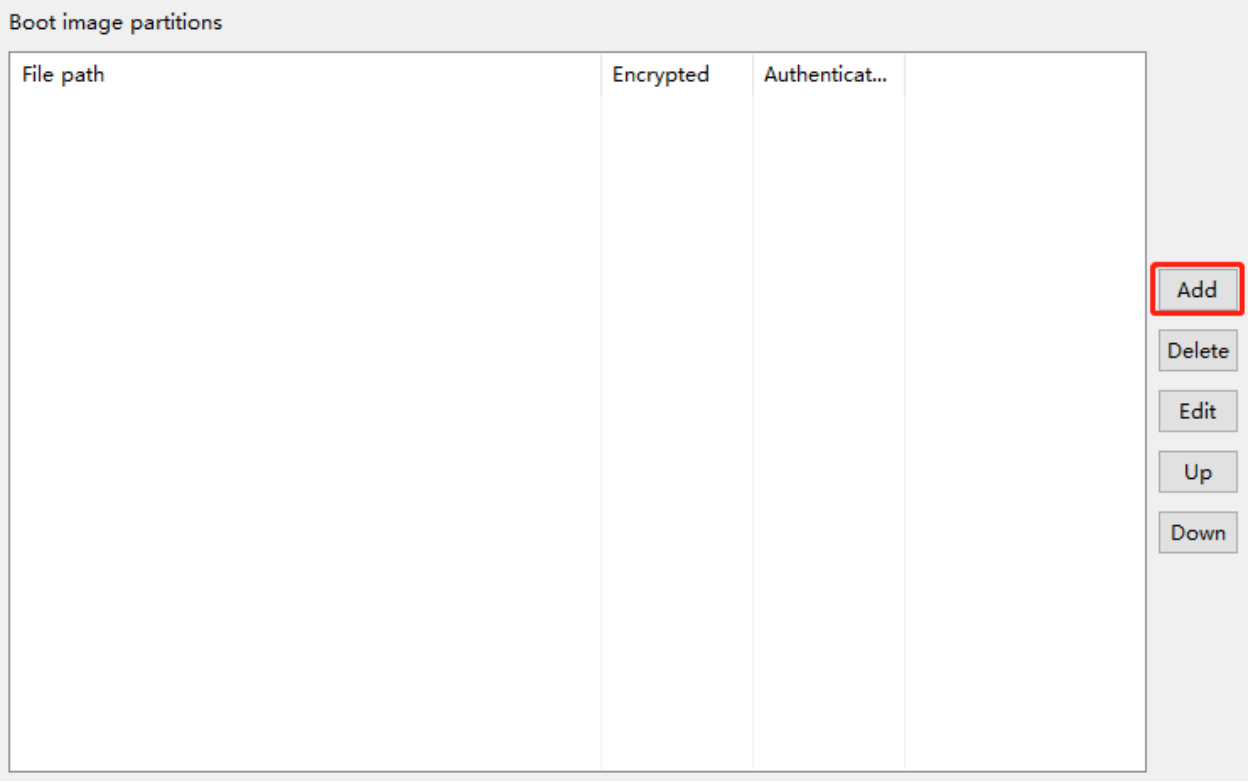
点击Next后选Zynq FSBL.



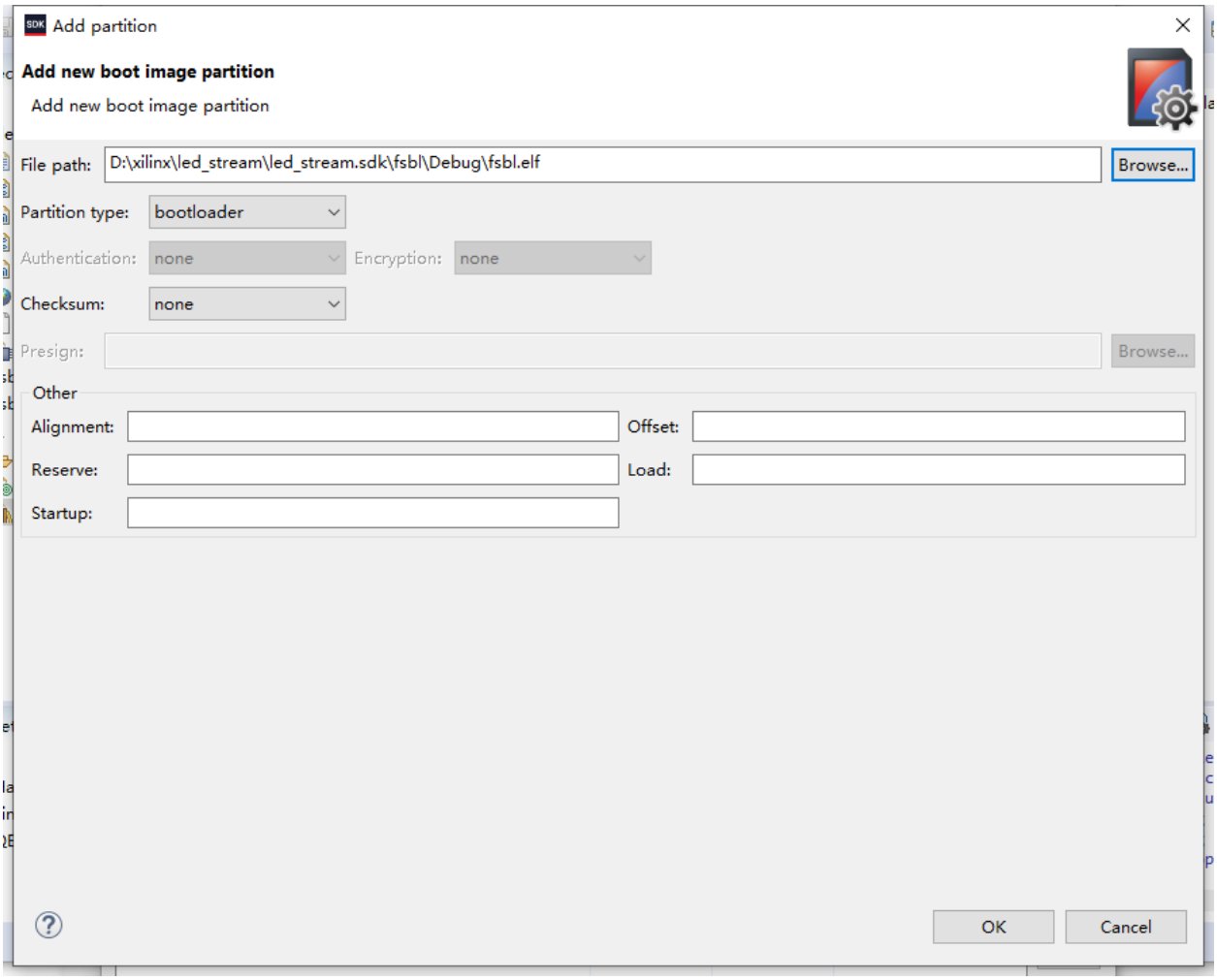
之后就有三个工程,分别是硬件描述,fsbl启动引导,fsbl软件,由于这里不做软件,只是为了固化,所以点击上方菜单的XILINUX→Create Boot Image,准备自动启动BIN,选择BIF路径,我习惯于保存在SDK的调试目录下.


Create Boot Image
Create Boot Image
At least one partition should be present in bootimage
Architecture: Zynq
Create new BIF file
Import from existing BIF file
Basic
Security
Output BIF file path: D:\xilinx\led_stream\led_stream.sdk\fsbl\Debug\output.bif
Browse...
UDF data:
Browse...
Split
Output format: BIN
Output path: D:\xilinx\led_stream\led_stream.sdk\fsbl\Debug\BOOT.bin
Browse...
Boot image partitions
File path
Encrypted
Authenticat...
Add
Delete
Edit
Up
Down
Preview BIF Changes
Create Image
Cancel

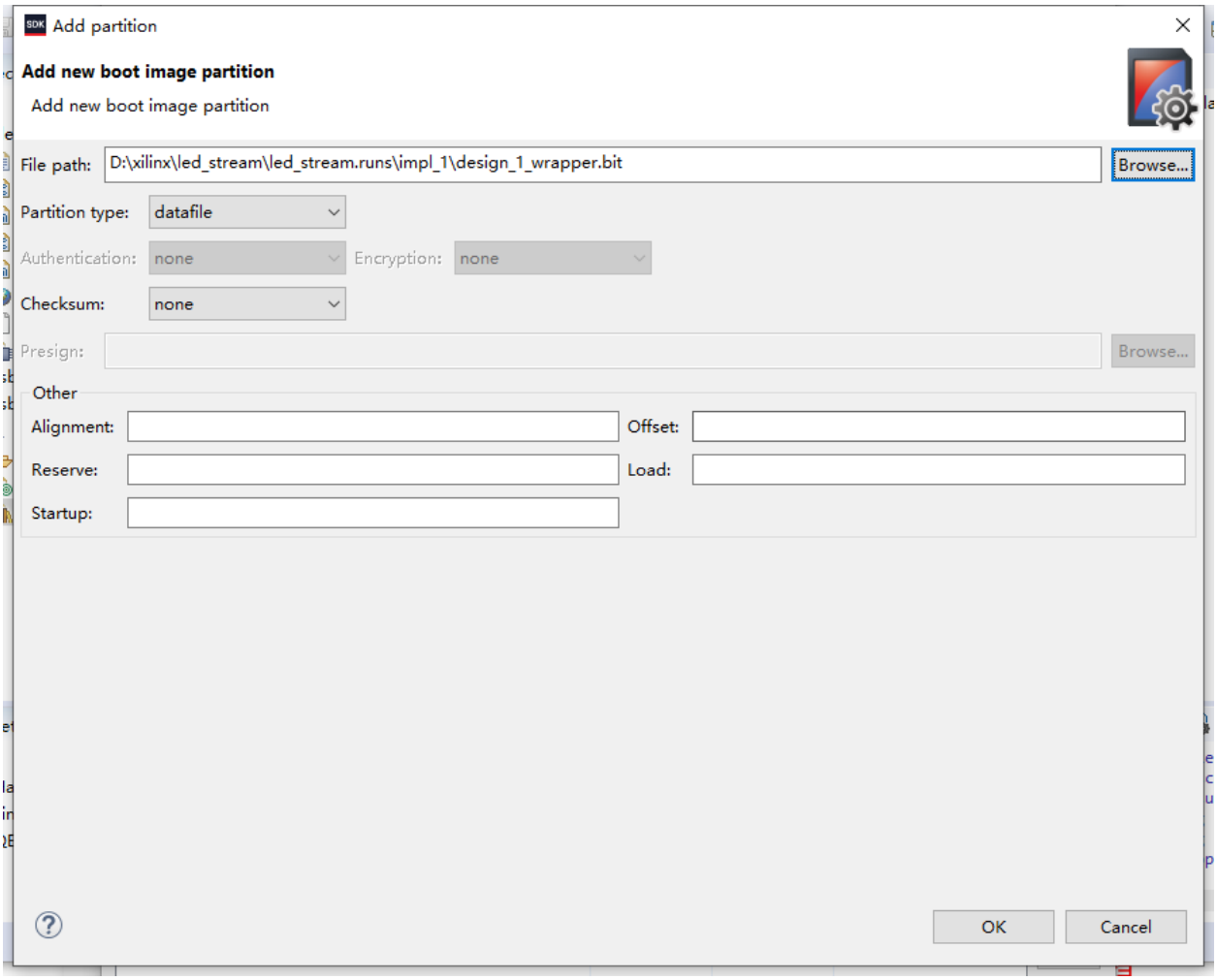
添加引导文件.



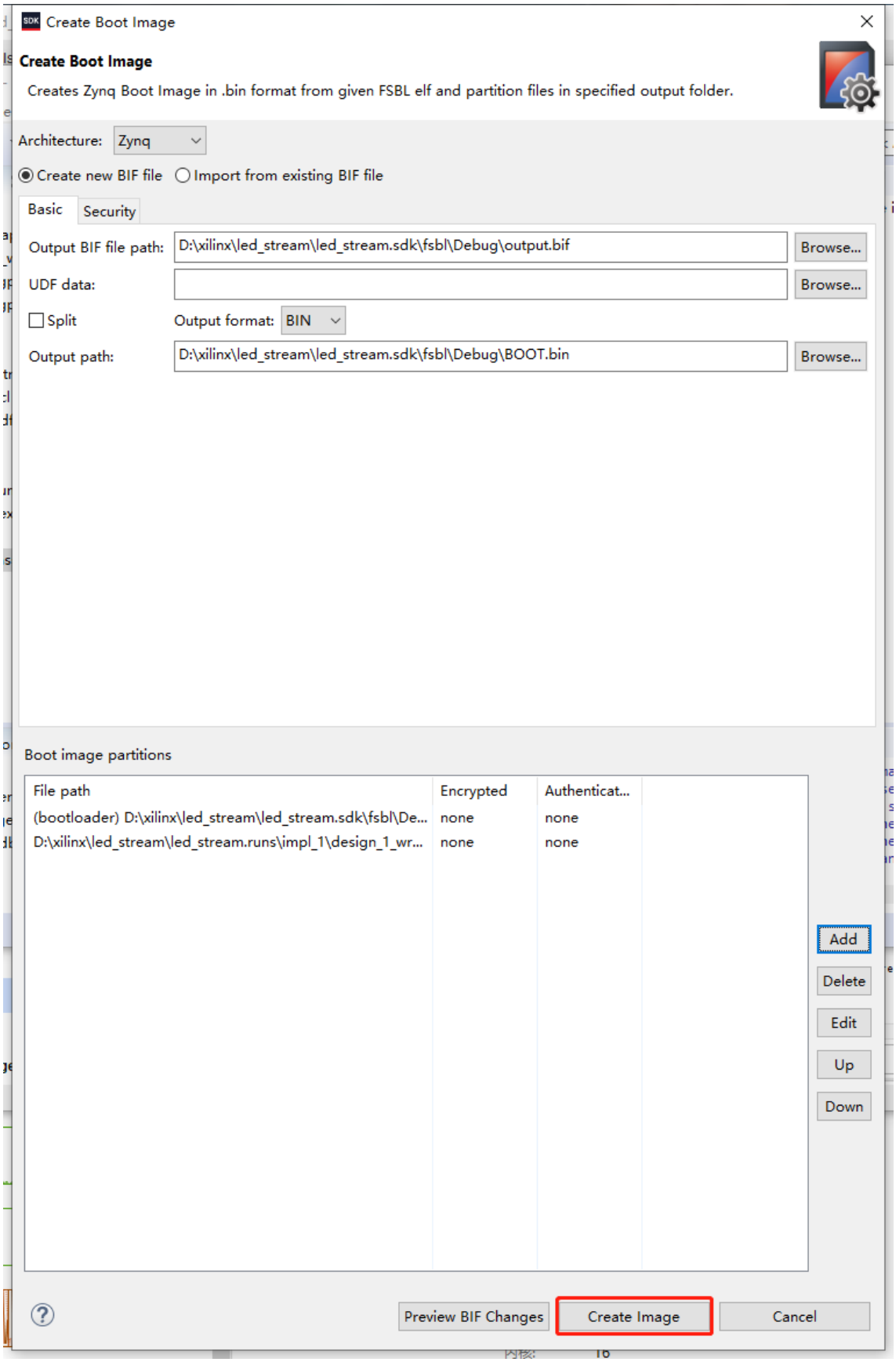
添加fsbl.elf,类型bootloader.



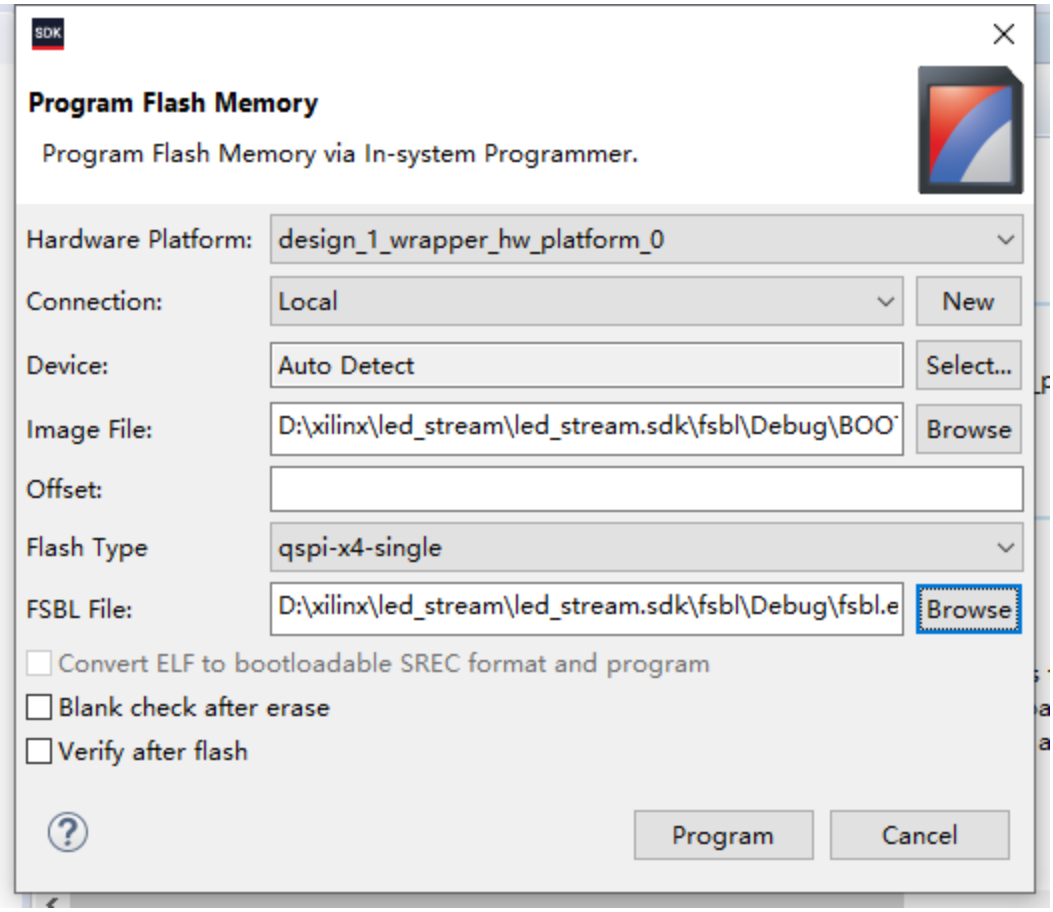
并把bit作为datafile添加.



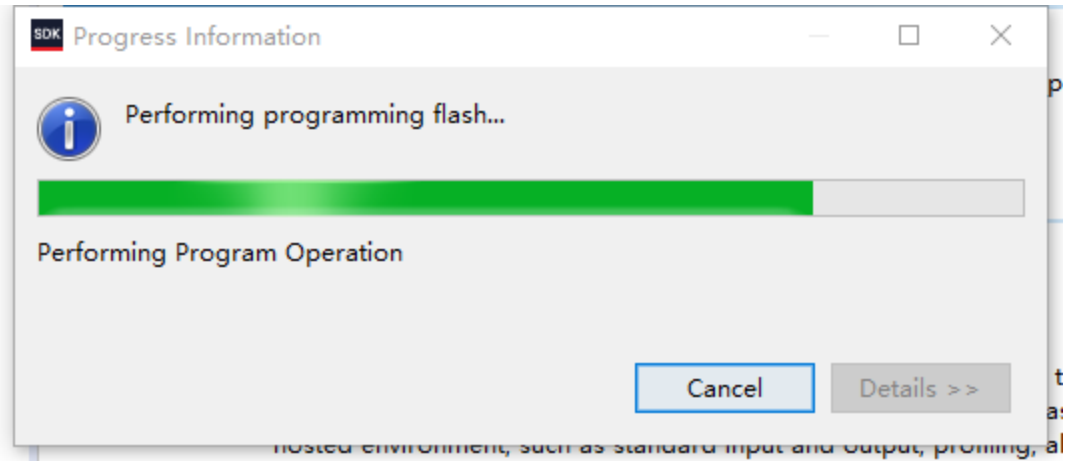
现在可以开始创建镜像了.


Create Boot Image
Create Boot Image
Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.
Architecture: Zynq
Create new BIF file
Import from existing BIF file
Basic
Security
Output BIF file path: D:\xilinx\led_stream\led_stream.sdk\fsbl\Debug\output.bif
Browse...
UDF data:
Browse...
Split
Output format: BIN
Output path: D:\xilinx\led_stream\led_stream.sdk\fsbl\Debug\BOOT.bin
Browse...
Boot image partitions
File path
Encrypted
Authenticat...
(bootloader) D:\xilinx\led_stream\led_stream.sdk\fsbl\De...
none
none
D:\xilinx\led_stream\led_stream.runs\impl_1\design_1_wr...
none
none
Add
Delete
Edit
Up
Down
Preview BIF Changes
Create Image
Cancel

断电调整板子上BOOT到JTAG模式,然后重新上电,选择SDK的XILINUX→Program Flash来烧录.



烧录完了要断电重新调整BOOT模式到正常启动,比如我是QSPI模式,然后重新上电就OK了.



固件固化真的很繁琐,因此平时直接调试就好了,因为ZYNQ其实是ARM主导,由ARM启动FPGA,因此就比较麻烦了,当然直接用VIVADO也可以固化,不过确实就体现不出ZYNQ的异构了.