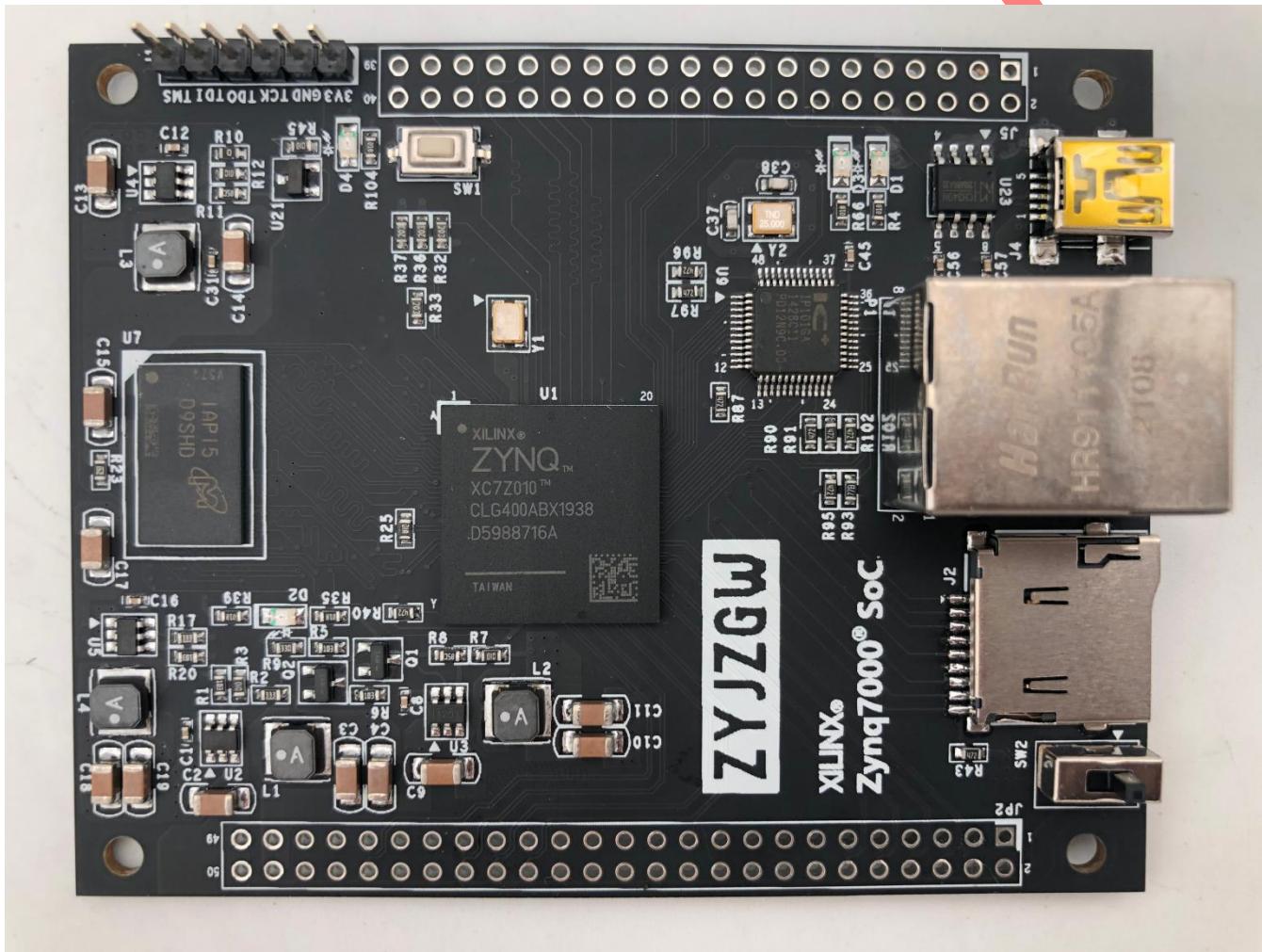


ZYJZGW ZYNQ XC7Z010 STARTER KIT

USER MANUAL



Preface

The ZYJZGW® ZYNQ XC7Z010 Starter Kit uses Xilinx Zynq®-7000 device which integrates the software programmability of an ARM®-based processor with the hardware programmability of an FPGA, enabling key analytics and hardware acceleration while integrating CPU, DSP, ASSP, and mixed signal functionality on a single device. Consisting of single-core Zynq-7000S and dual-core Zynq-7000 devices, the Zynq-7000 family is the best price to performance-per-watt, fully scalable SoC platform for your unique application requirements.

Table of Contents

1.	INTRODUCTION.....	3
1.1	DOCUMENT SCOPE.....	3
1.2	TEST EXAMPLES.....	3
1.3	STARTER KIT SETUP.....	4
2.	GETTING STARTED.....	5
2.1	STEPS TO CUSTOMIZE THE ZYNQ PROCESSING SYSTEM.....	5
2.1.1	<i>Step 1: Create New Project.....</i>	5
2.1.2	<i>Step 2: Add Zynq Processing System.....</i>	8
2.1.3	<i>Step 3: Customize Zynq Processing System.....</i>	10
2.1.4	<i>Step 4: Generate Output Products.....</i>	14
3.	EXPERIMENT 1: EMIO USER LED.....	15
4.	EXPERIMENT 2: MIO USER LED.....	19
5.	EXPERIMENT 3: DDR3 TEST.....	20
6.	EXPERIMENT 5: LWIP TEST.....	23
7.	REFERENCE.....	26
8.	REVISION.....	27

1. Introduction

1.1 Document Scope

This demo user manual introduces the non-Linux part test examples that running on the ZYJZGW ZYNQ XC7Z010 Starter Kit. Those examples are all running with Xilinx Vivado 2018.3 environment. So the prerequisites before working with the examples are shown as below:

1. Users have already installed the Vivado 2018.3 in the Windows OS.
2. Users have the basic knowledge about the usage of the Vivado environment. At least know how to synthesis, implement and generate bitstream, etc.

1.2 Test Examples

Below diagram shows the main parts that these test examples cover:

- PL side MIO(Multipurpose Input Output);
- PL side 100Mbps MII ethernet Interface;
- PS side EMIO(Extendable Multipurpose Input Output);
- PS side UART;
- PS side DDR3 Memory Controller;
- PS side ARM core, mainly running with UBoot;

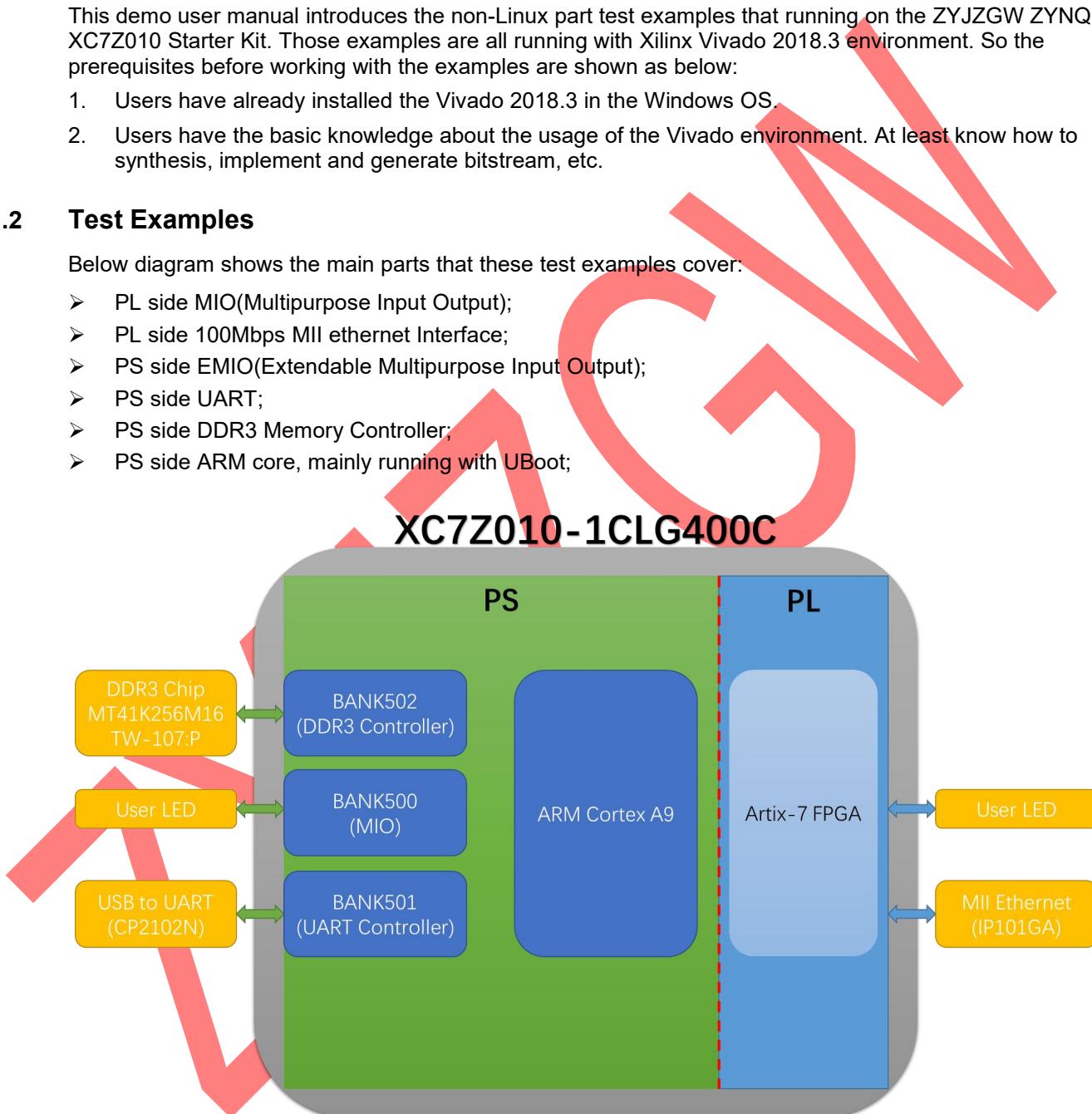


Figure 1-1. Tested Peripherals

1.3 Starter Kit Setup

Before start to test the Zynq7000 Starter Kit, users need to prepare the hardware setup shown as in below image. In default, the factory binary test images are already stored in the micro SD card and the D4 LED will be periodically blinking.

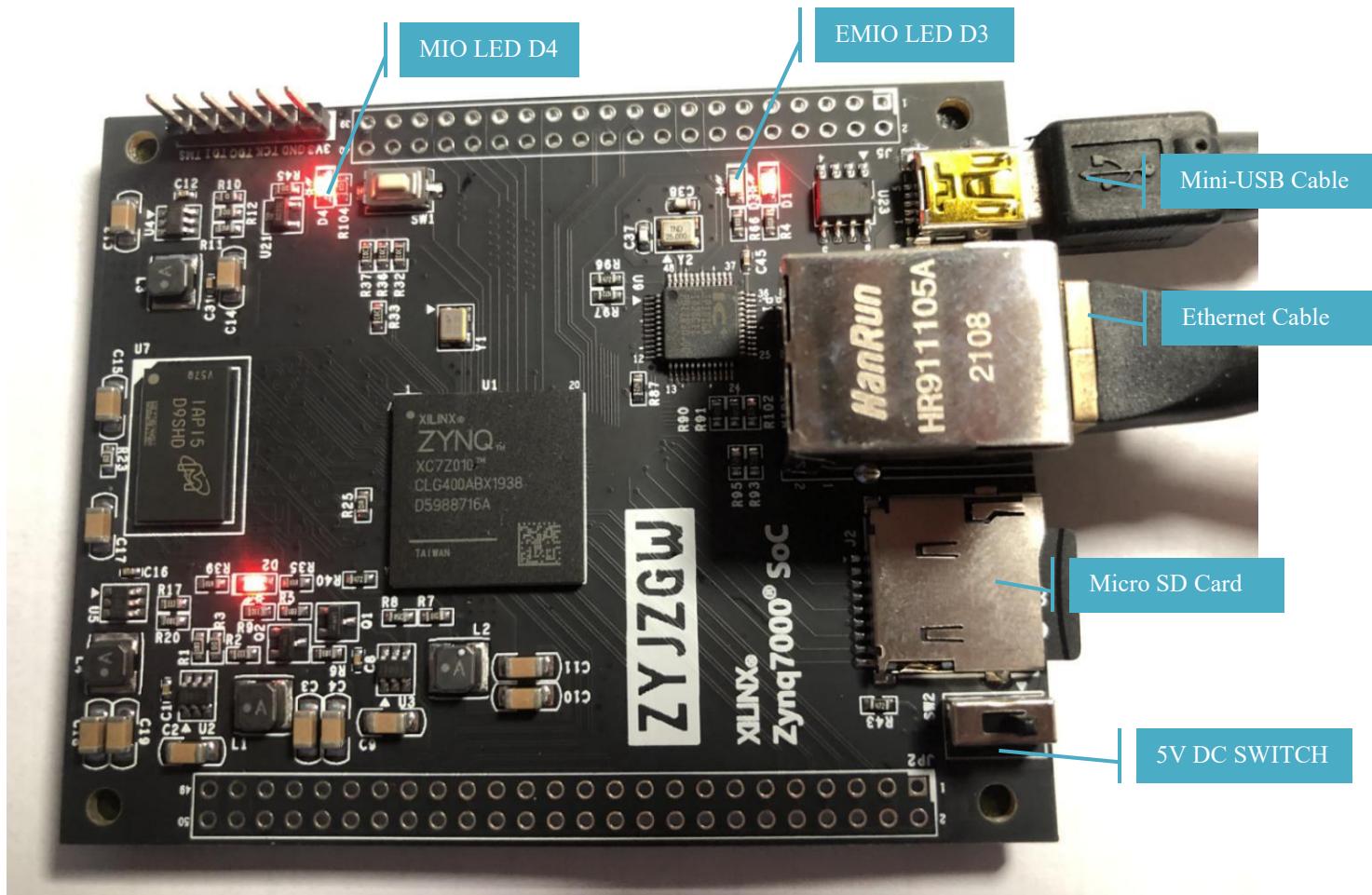


Figure 1-2. Hardware Setup

2. Getting Started

This chapter describes the detailed steps to create a customized ZYNQ Processing System. Comparing to the existing ZYNQ development board e.g. Xilinx ZC702, there are many differentiations in the ZYJZGW ZYNQ Starter Kit. For example, there's only one 16bit width DDR3 memory chip connected to PS ARM core. Another important change is the ethernet interface no longer uses the PS side MIO, instead the MII ethernet chip is connected to PL side EMIO.

2.1 Steps to Customize the ZYNQ Processing System

2.1.1 Step 1: Create New Project

Open Vivado 2018.3, then click 【Create Project】 shown in below figure.

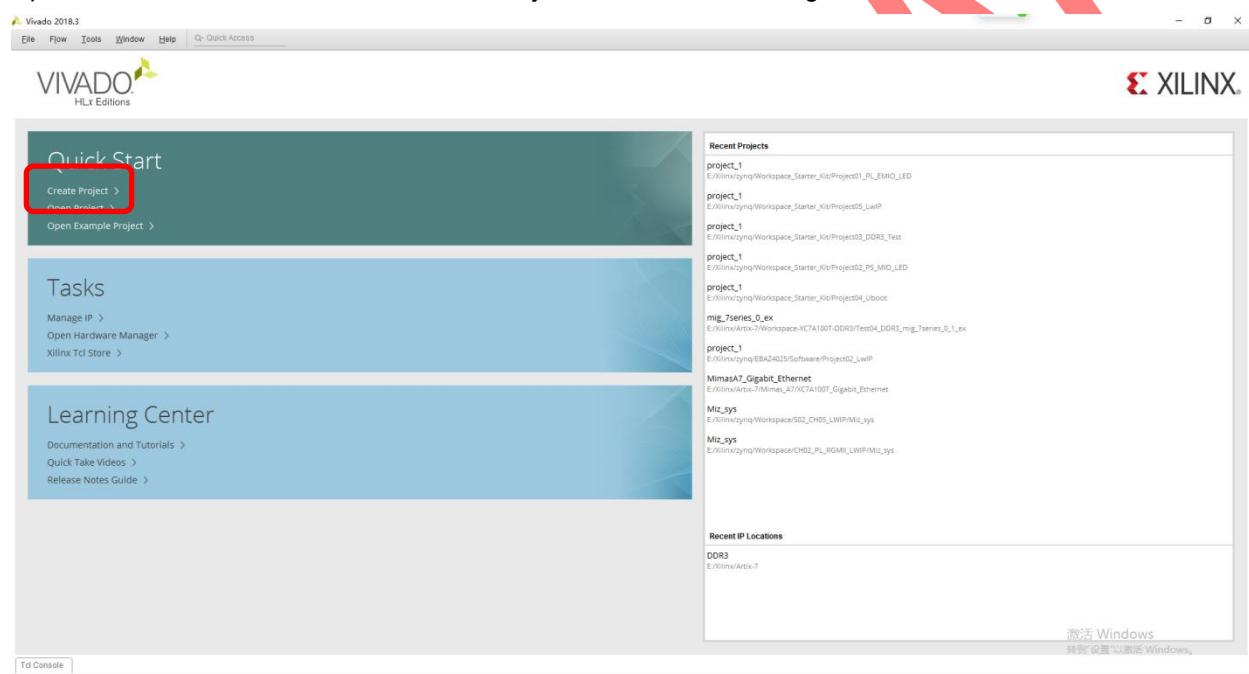


Figure 2-1. Vivado 2018.3

Below image will be shown and click 【NEXT】button:

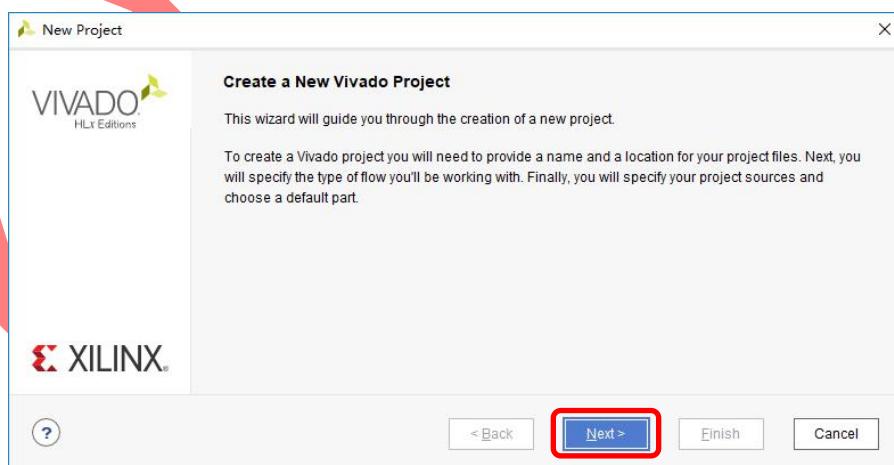


Figure 2-2. Create New Project

Set the Project name and the project location:

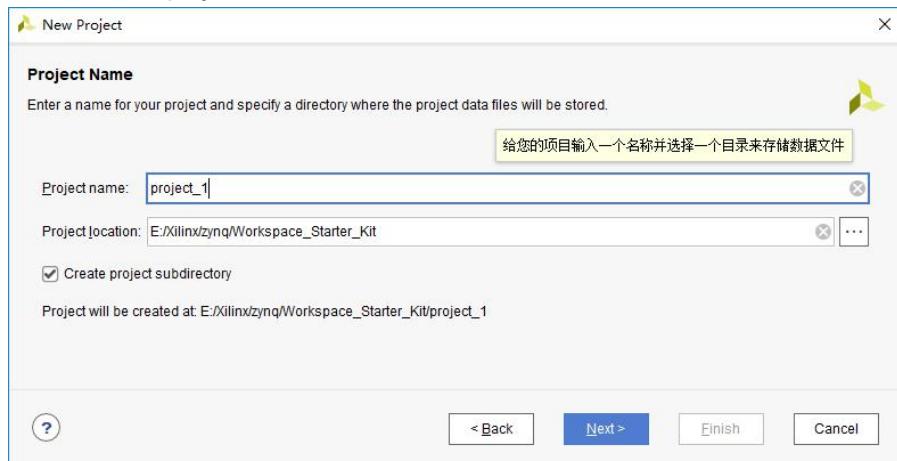


Figure 2-3. Set Project Name and Location

Click 【Next】button in below image:

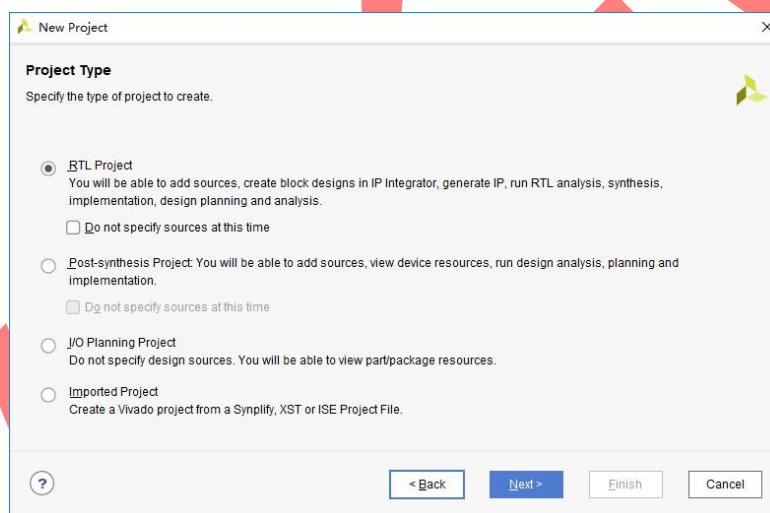
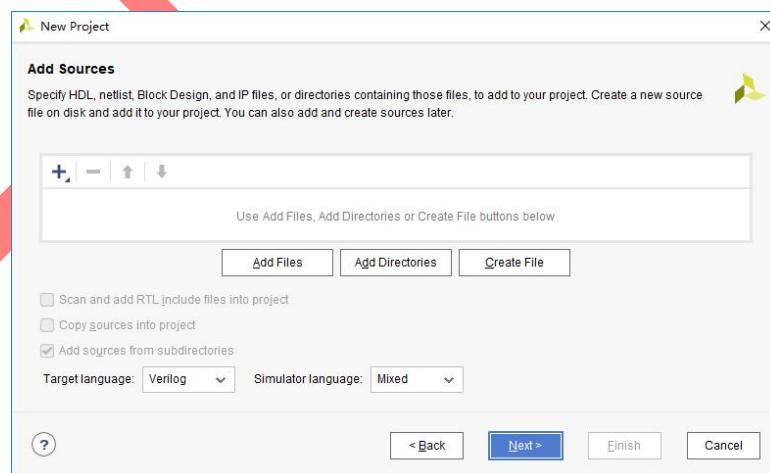


Figure 2-4. Click Next

Click 【Next】button in below image if there's no source file existing:



Click 【Next】 button in below image if there's no constraint file existing:

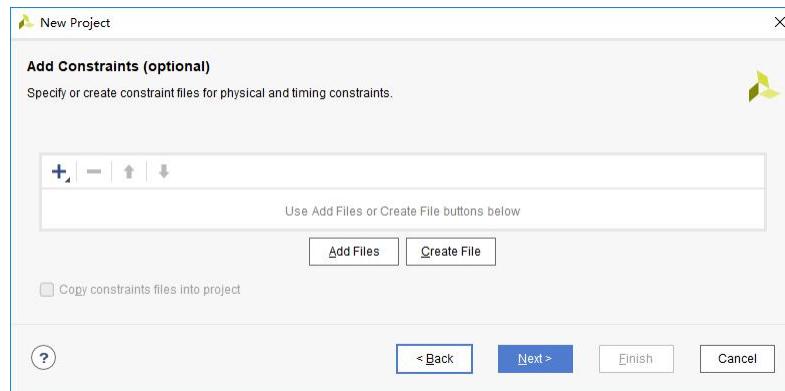


Figure 2-5. Click Next

Select the device consistent to the chip mounted on ZYJZGW ZYNQ xc7z010-1clg400c:

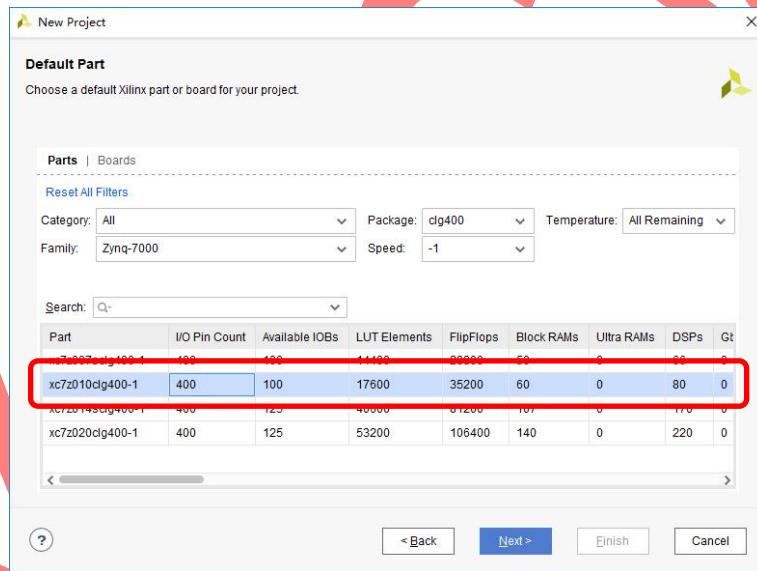
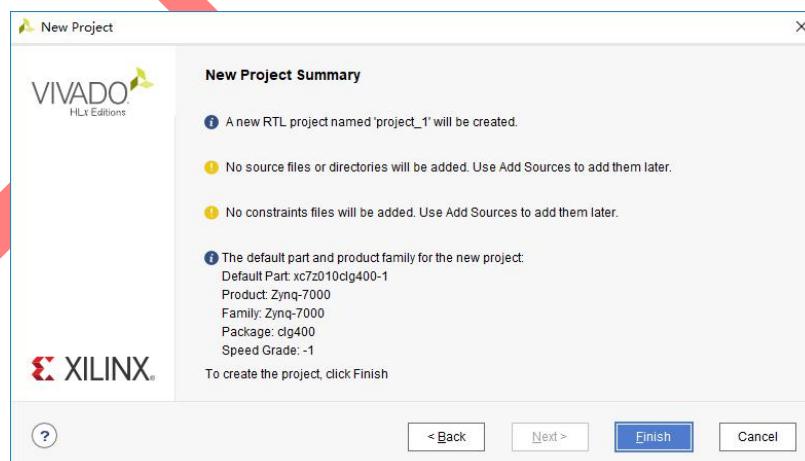


Figure 2-6. Select Device

Click 【Finish】 if there's nothing needs to be changed.



2.1.2 Step 2: Add Zynq Processing System

After the project successfully created, users can start to add the Zynq PS to the design by click 【Create Block Design】 in the IP INTEGRATOR. Type example name ‘system’ in the editbox shown in below image.

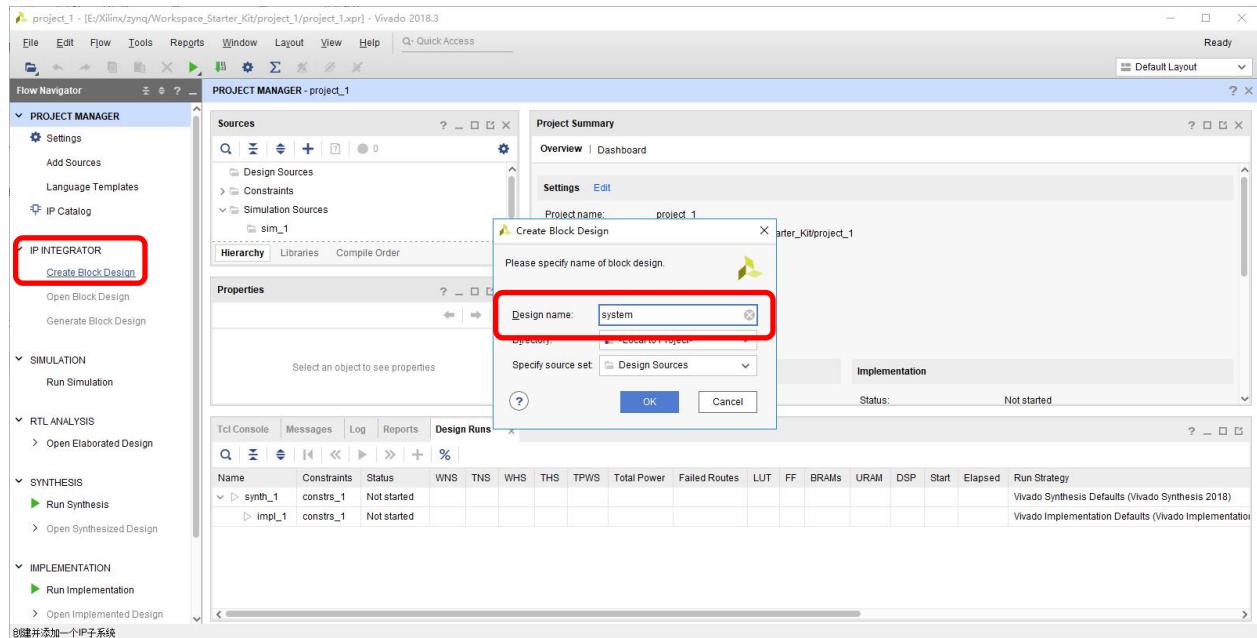


Figure 2-7. Create Block Design

Click the 【+】 button shown in below image and Search the keyword ZYNQ in the edit box. If the ZYNQ7 Processing system can be found, users could double click it and add it in the design.

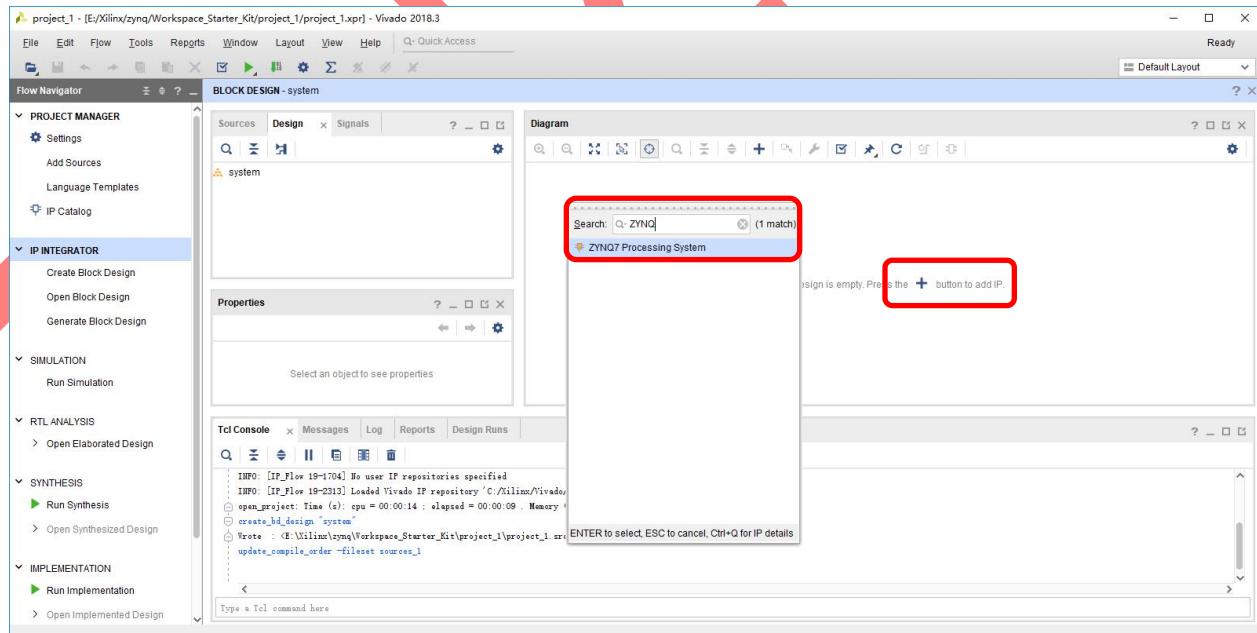


Figure 2-8. Add ZYNQ7 PS

The below image will be displayed once the ZYNQ7 is correctly added.

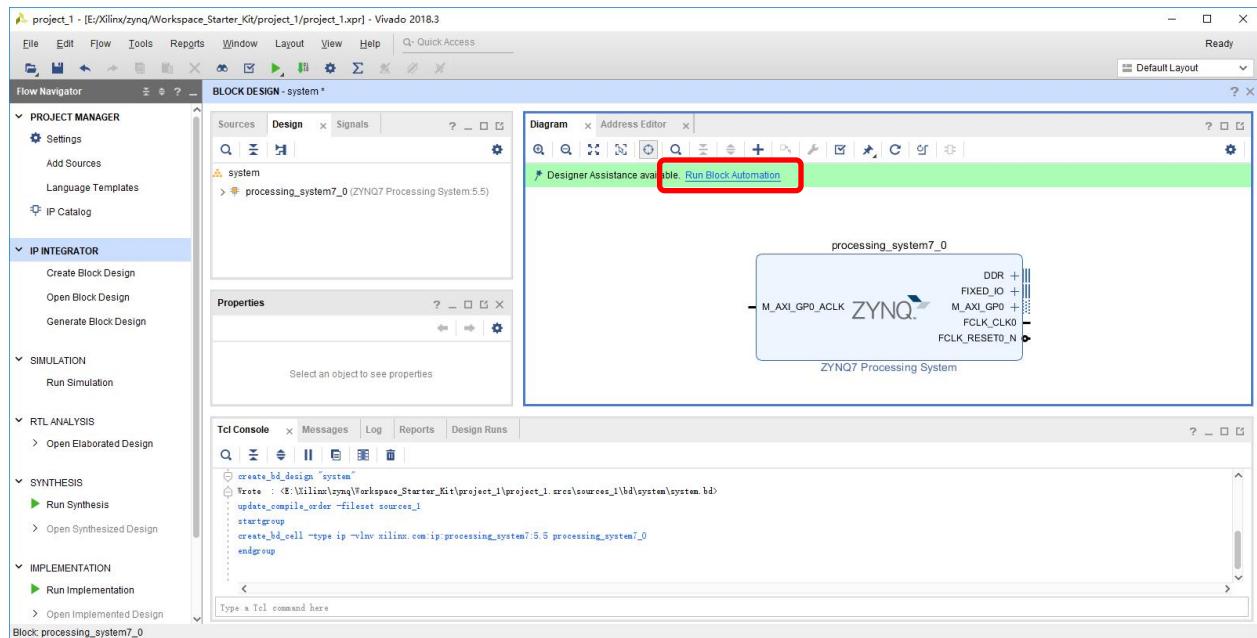


Figure 2-9. Added ZYNQ7 PS

Users could click the 【Run Block Automation】. And then connect PS clock 【FCLK_CLK0】 to 【M_AXI_GP0_ACLK】.

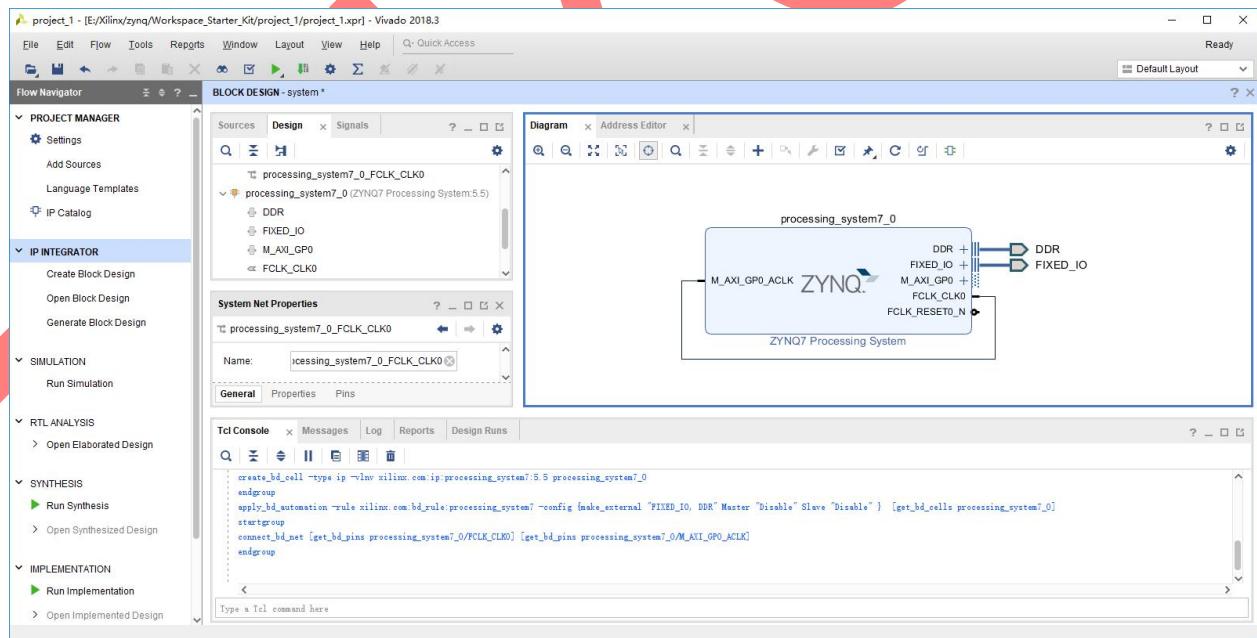


Figure 2-10. Block Automation

2.1.3 Step 3: Customize Zynq Processing System

Double click the ZYNQ IP shown in the above image to implement the detailed customizations. The architecture block diagram will be shown as below. All the configurable parts are all listed in the Page Navigator.

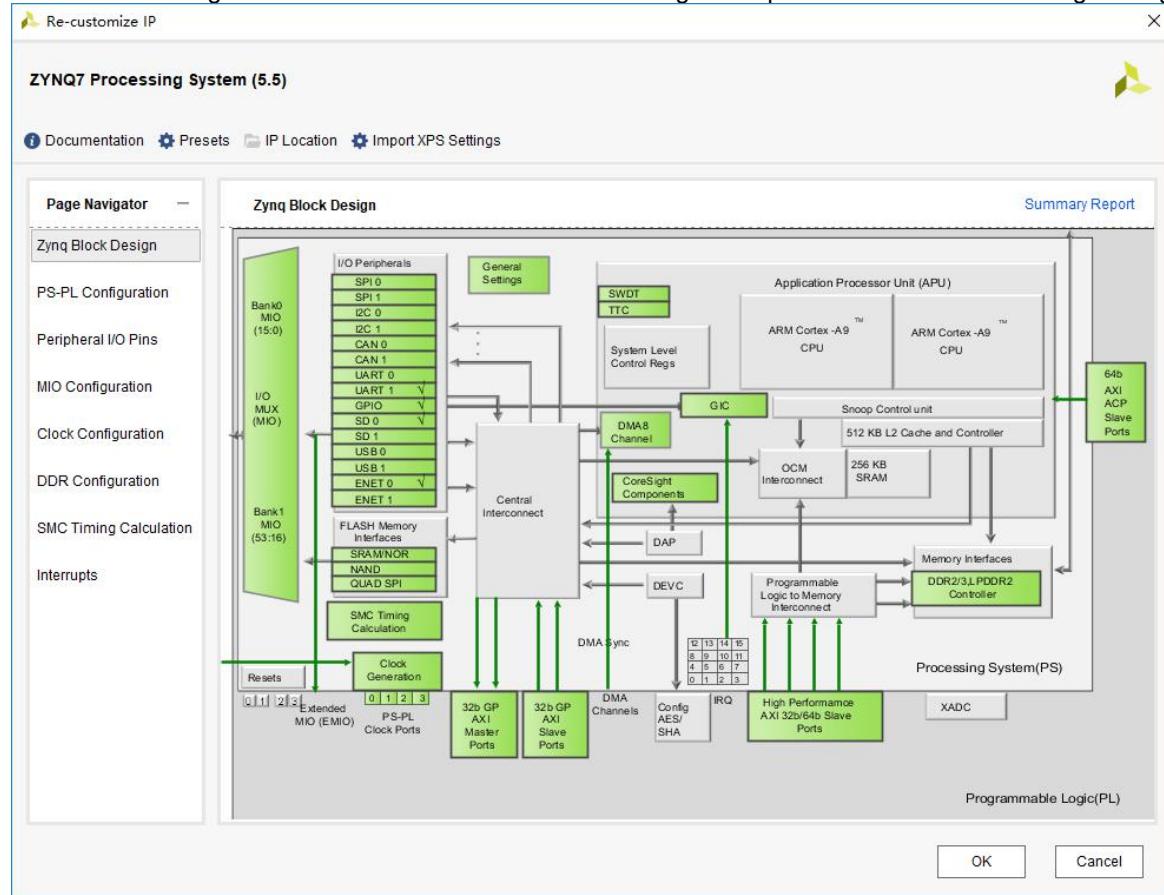


Figure 2-11. ZYNQ Architecture

Click the **PS-PL Configuration** page, nothing needs to be changed here.

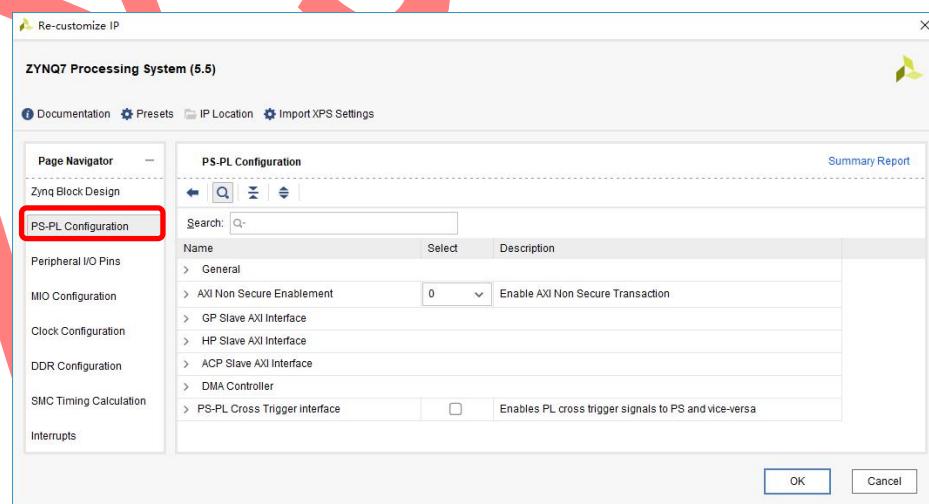


Figure 2-12. ZYNQ Architecture

Click the **Peripheral I/O Pins** page. Configure the Ethernet 0 shown as below. Assign the Ethernet 0 and MDIO I/Os on the EMIO because all the MII interface are connected to the PL side I/Os.

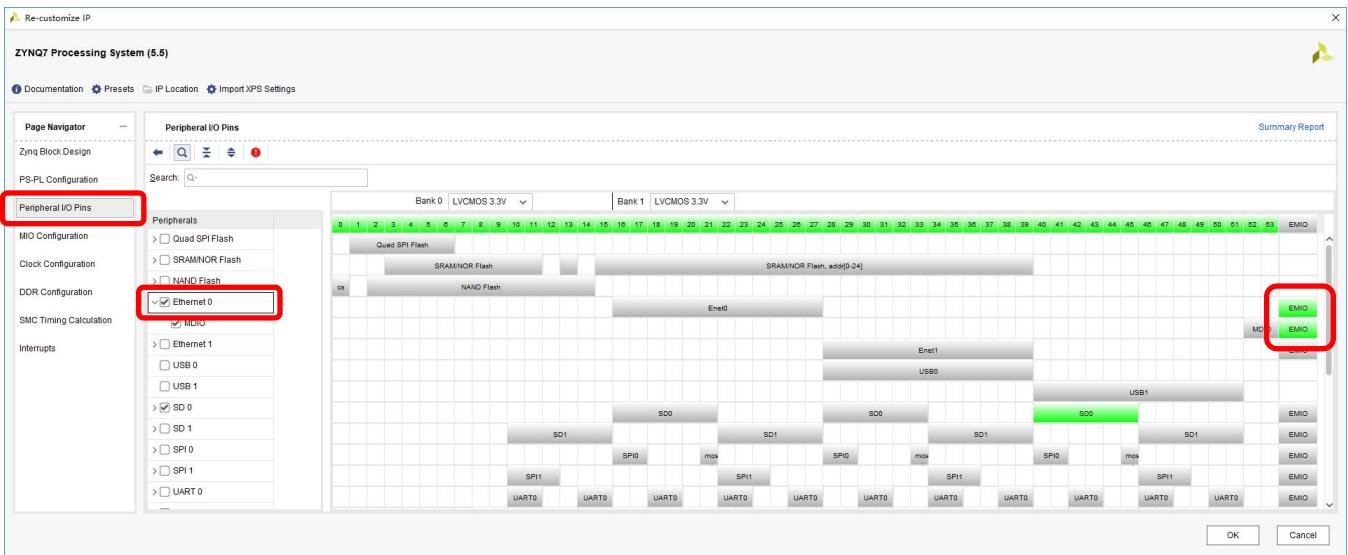


Figure 2-13. Configure Ethernet 0

Configure the SD 0 shown as in below image. The SD card slot is connected to MIO[40:45] and the SD detect signal is connected to MIO47.

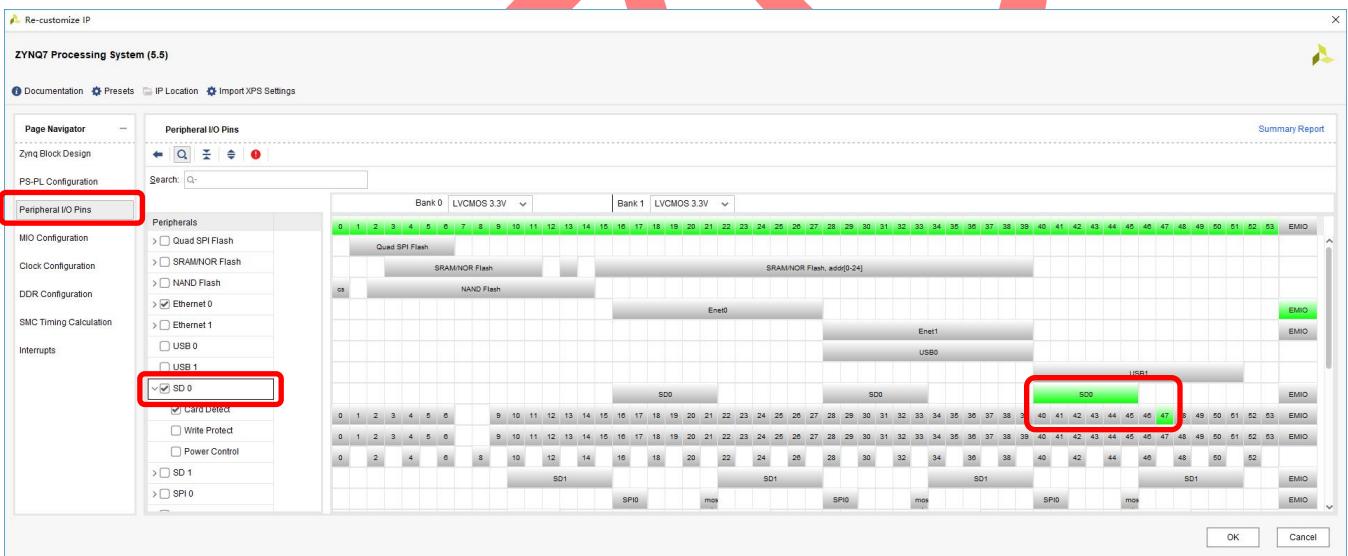
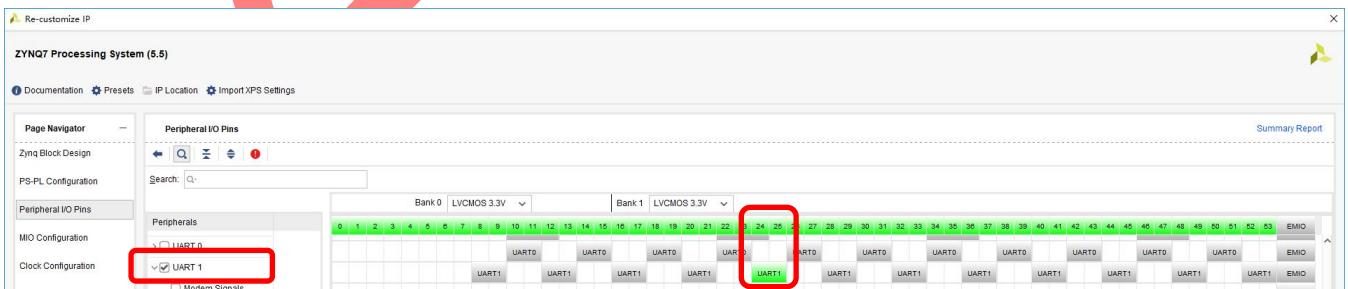


Figure 2-14. Configure SD 0

Below image shows the UART 1 port is connected to MIO24 and MIO25.



Click **MIO Configuration** page. Make sure all the MIO/EMIO assignment are same to the configurations shown in below image.

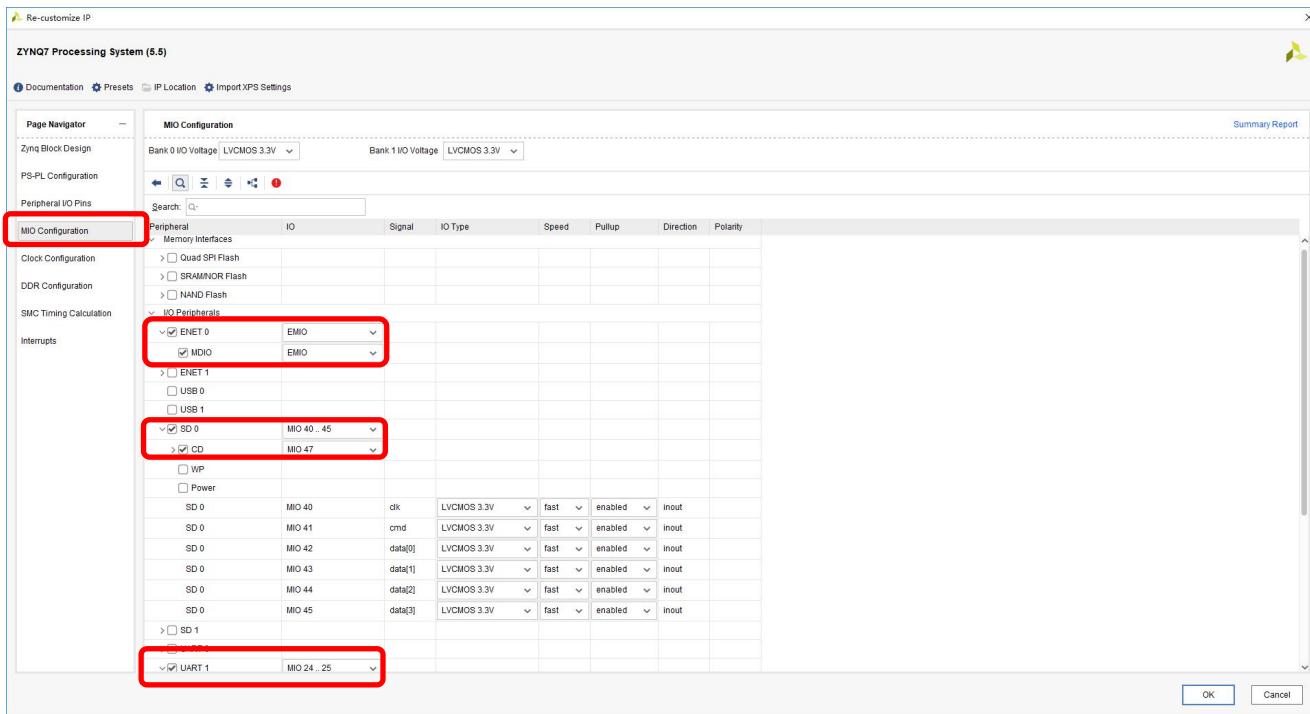
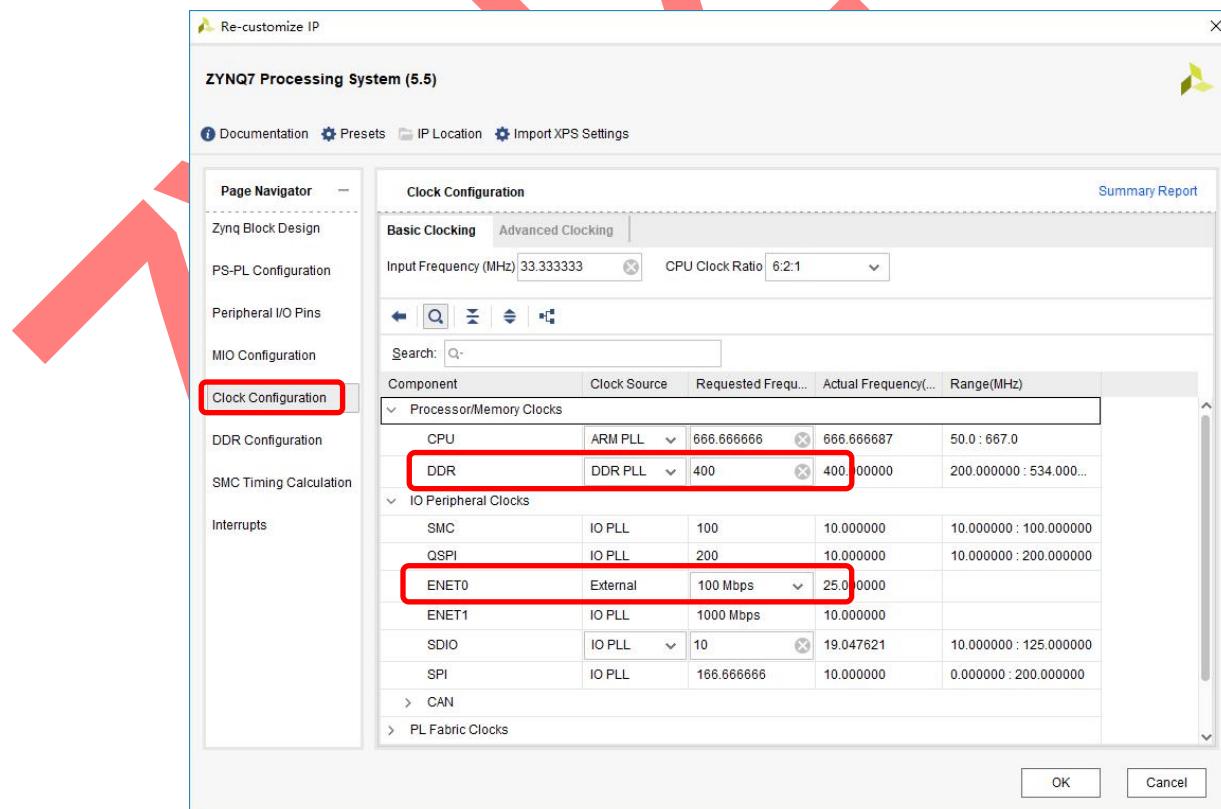


Figure 2-15. MIO/EMIO Assignment

Click **Clock Configuration** page. Configure the ENET0 clock with 100Mbps because the IP101GA only supports MII 100Mbps interface.



Click **DDR Configuration** page. Change the Effective DRAM Bus Width into 16 Bit and select MT41K256M16 RE-125 as the memory part, though the actual part mounted on the ZYJZGW ZYNQ Starter Kit is MT41K256M16TW-107:P. This is still workable because the clock frequency is only 400MHz and all the DDR3 memory chips are down speed compatible. If users still worry about the stability thing, then DDR3 memory customization also could be done in this page. And detailed parameters like the CAS latency, RAS to CAS Delay, etc. could be retrieved from Micron DDR3 datasheet and filled in this DDR Configuration page.

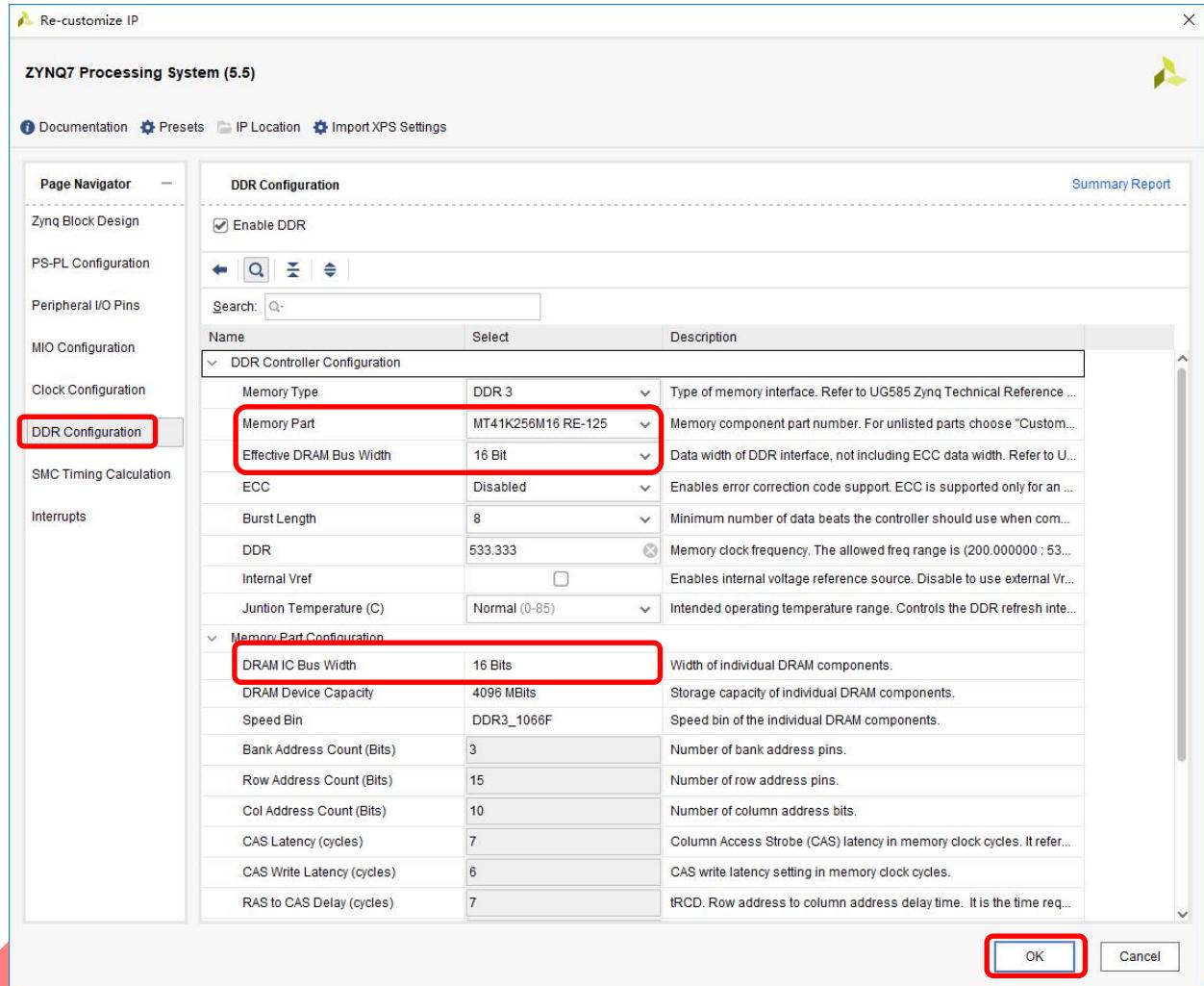


Figure 2-16. DDR3 Memory Configuration

For the **SMC Timing Calculation** and **Interrupts** pages, nothing needs to be changed here. And then click the OK button to finish the whole ZYNQ Processing System customization.

2.1.4 Step 4: Generate Output Products

After the ZYNQ PS customization finished, users still need to make the PINs external. The special thing here needs to do is to add **concat** IP to match the ethernet bus width differentiation. The original RXD/TXD bus width for GMII interface is 8 bit while the RXD/TXD bus width for MII interface is only 4 bit.

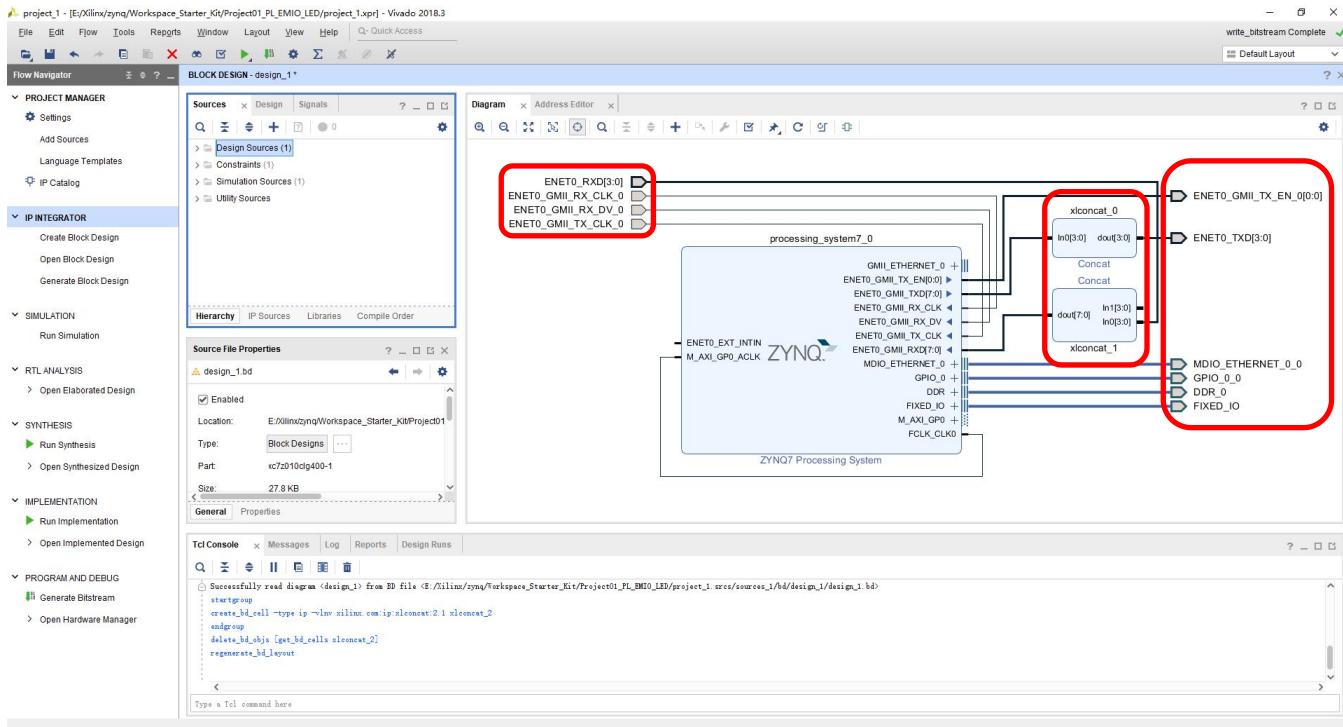
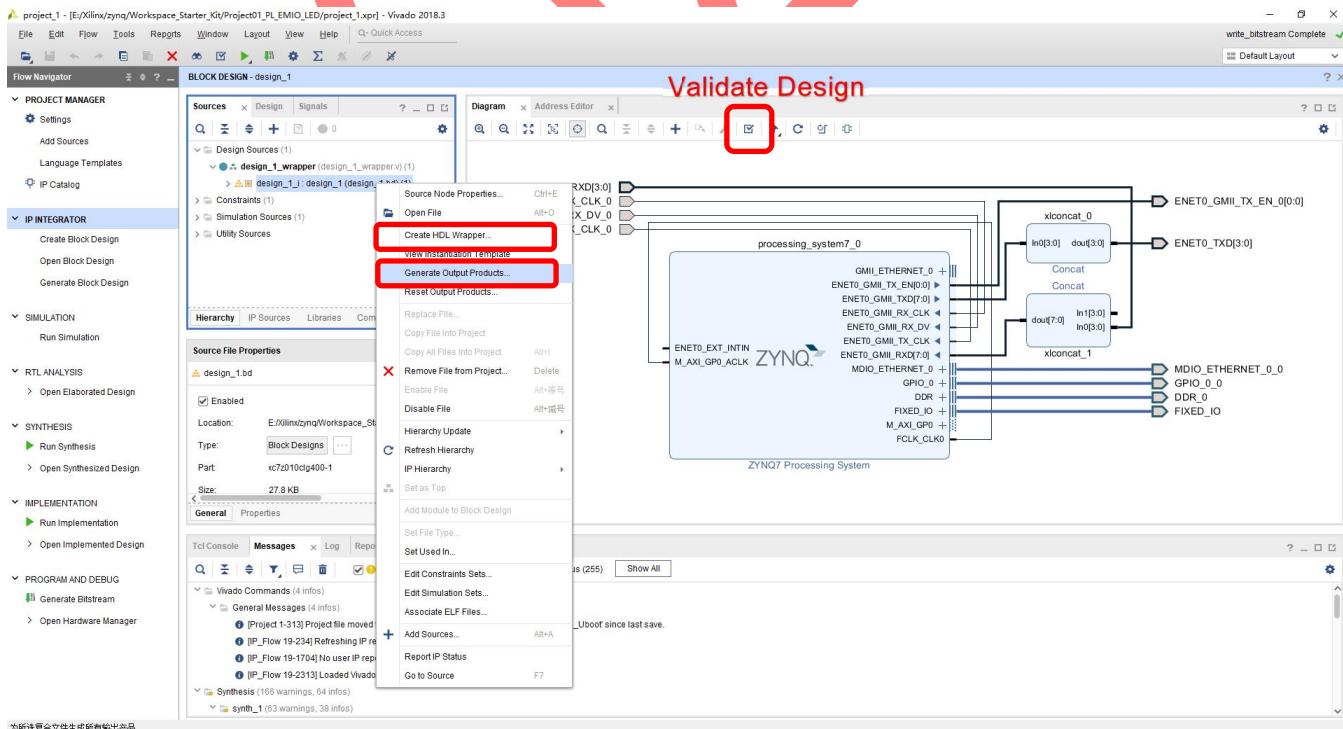


Figure 2-17. Make External

Then **Validate Design**, **Generate Output Products** and **Create HDL Wrapper** to finish the whole procedure.



为所选复合文件生成所有输出产品

3. Experiment 1: EMIO User LED

Before start to test the user LED connected to EMIO, users need to make sure the designer_1_wrapper is correctly generated in the previous step. And then execute the whole compilation procedure: **Run Synthesis**, **Run Implementation** and **Generate Bitstream**. Make sure there's no error generated in any of these three steps.

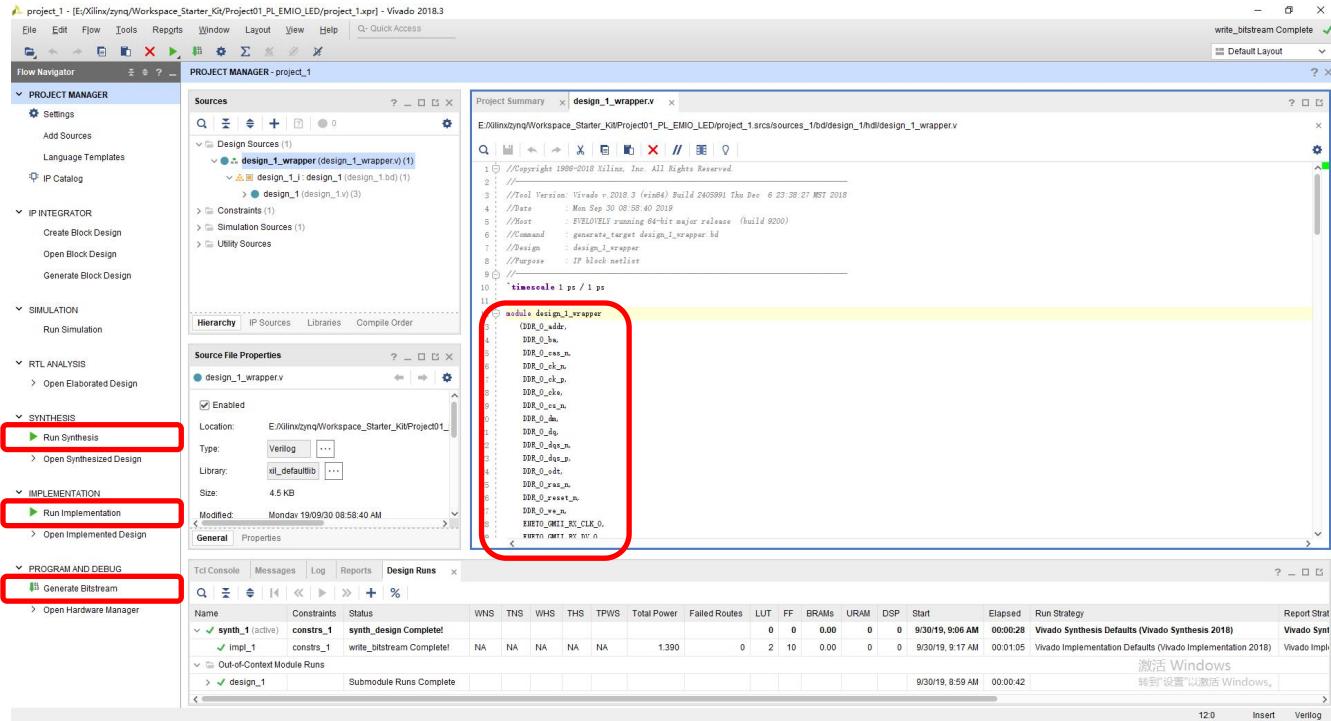
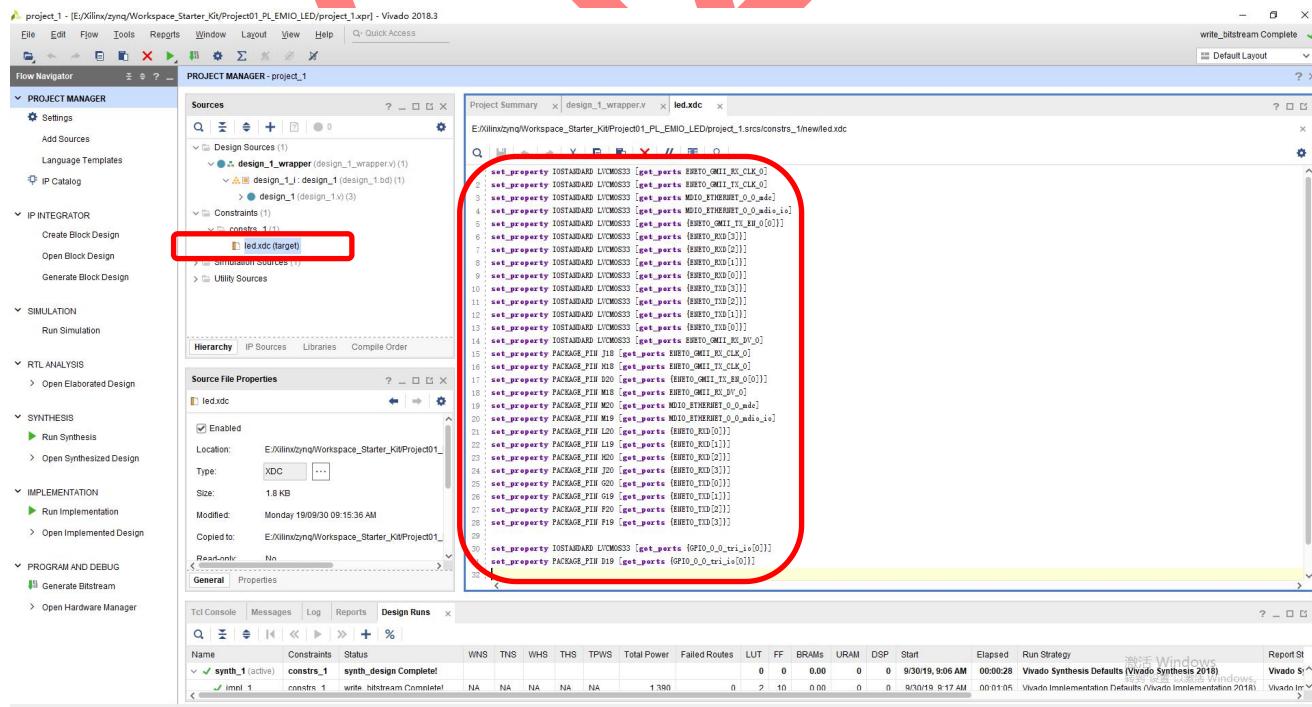


Figure 3-1. Generate Bitstream

Below image shows the constraint file of this project. Users may modify this constraint file if needed.



Then export hardware to the SDK by clicking 【File】→【Export Hardware】 , select the checkbox **Include bitstream** and click the OK button. The detailed procedure is shown as below:

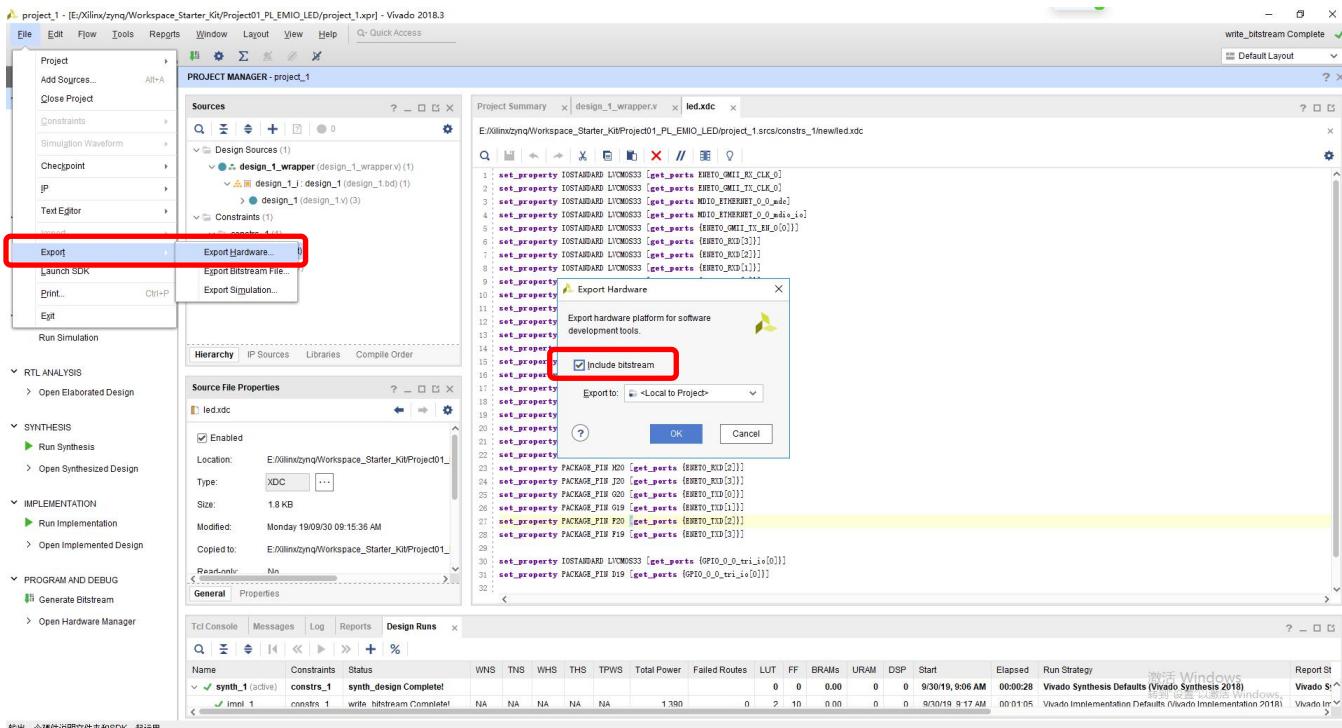


Figure 3-2. Export Hardware to SDK

Start the SDK by clicking 【File】→【Launch SDK】 and then clicking the OK button:

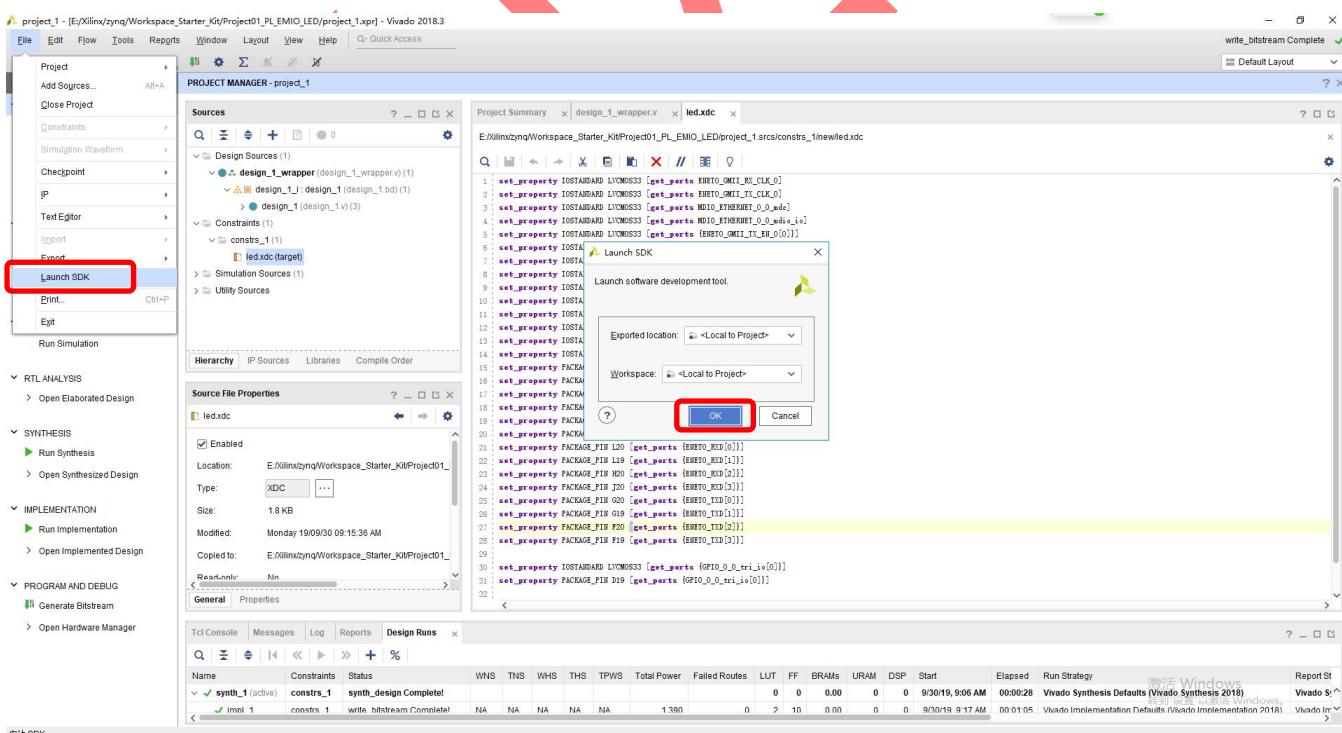


Figure 3-3. Start SDK

Setup a new project for testing the EMIO by clicking 【New】 → 【Application Project】 , and type EMIO_Test in the project name.

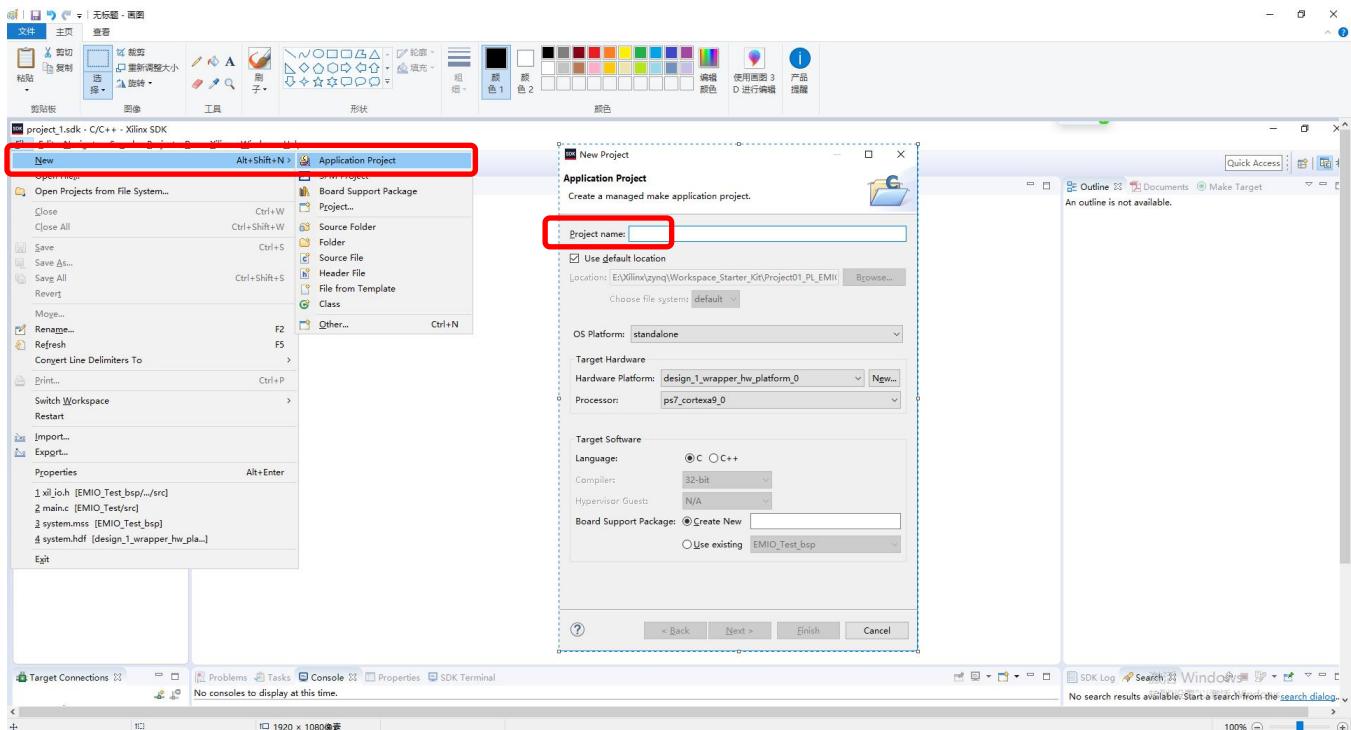
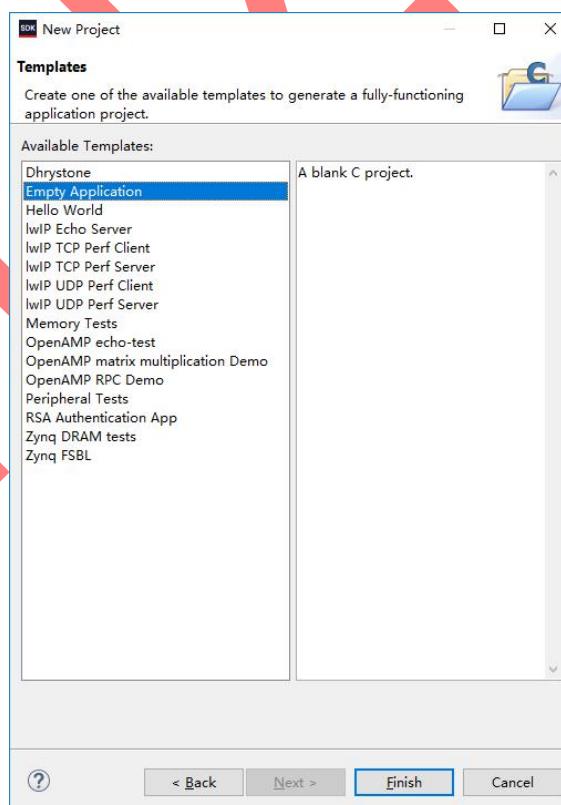
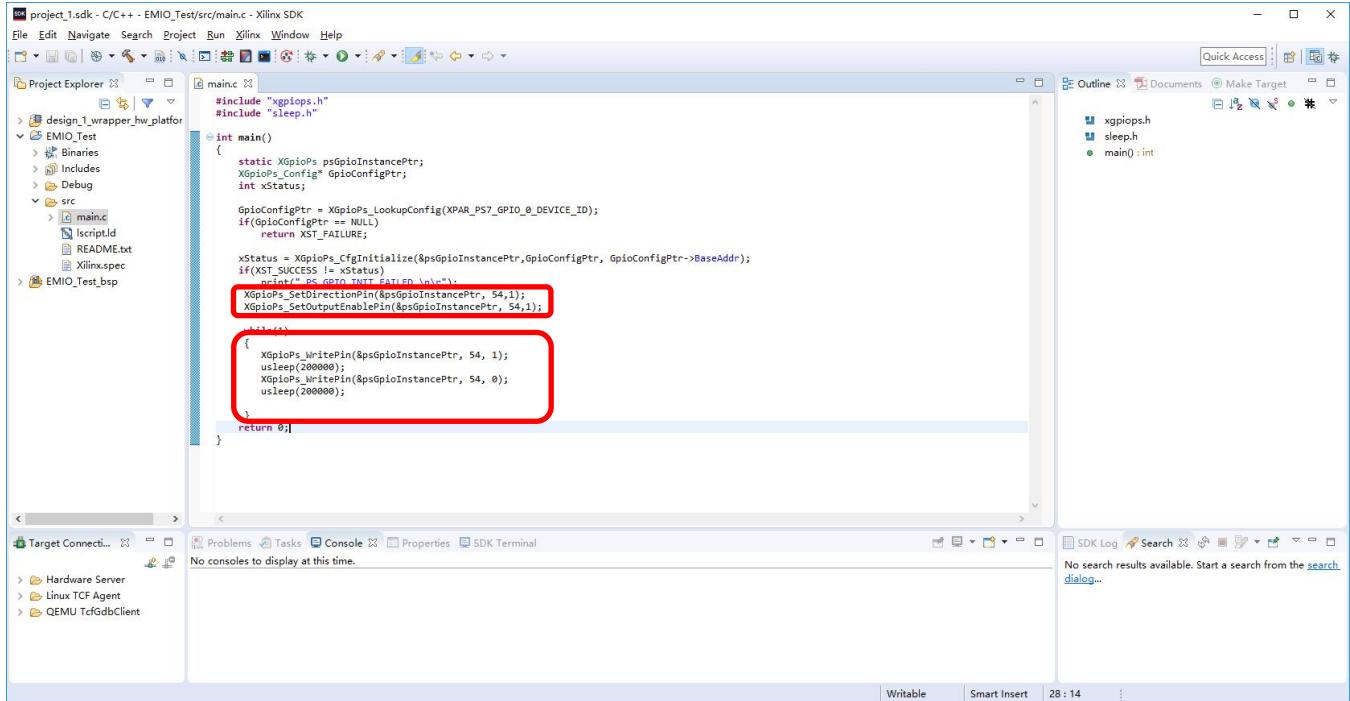


Figure 3-4. Create New Project

Choose Empty Application and click the OK button.



Add source file main.c in the EMIO_Test/src folder and the project will be automatically built by the Xilinx SDK environment. The below test routine configures the EMIO pin D19 as the output pin and enable it. After that it will periodically toggle the pin.



```

project_1.sdk - C/C++ - EMIO_Test/src/main.c - Xilinx SDI
File Edit Navigate Search Project Run Xilinx Window Help
Project Explorer
EMIO_Test
src
main.c
#include "xgpiops.h"
#include "sleep.h"

int main()
{
    static XGpioPs psGpioInstancePtr;
    XGpioPs_Config* GpioConfigPtr;
    int xStatus;

    GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
    if(GpioConfigPtr == NULL)
        return XST_FAILURE;

    xStatus = XGpioPs_CfgInitialize(&psGpioInstancePtr,GpioConfigPtr, GpioConfigPtr->BaseAddr);
    if(XST_SUCCESS != xStatus)
        nsim("PS GPIO INIT FAILED \n");

    XGpioPs_SetDirectionPin(&psGpioInstancePtr, 54,1);
    XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, 54,1);

    while(1)
    {
        XGpioPs_WritePin(&psGpioInstancePtr, 54, 1);
        usleep(200000);
        XGpioPs_WritePin(&psGpioInstancePtr, 54, 0);
        usleep(200000);
    }

    return 0;
}

```

Figure 3-5. Toggle EMIO Pin D19

Right click the project folder EMIO_Test and then select **Run As→1 Launch on Hardware (System Debugger)**. And then users could see the D3 LED will be periodically blinking.

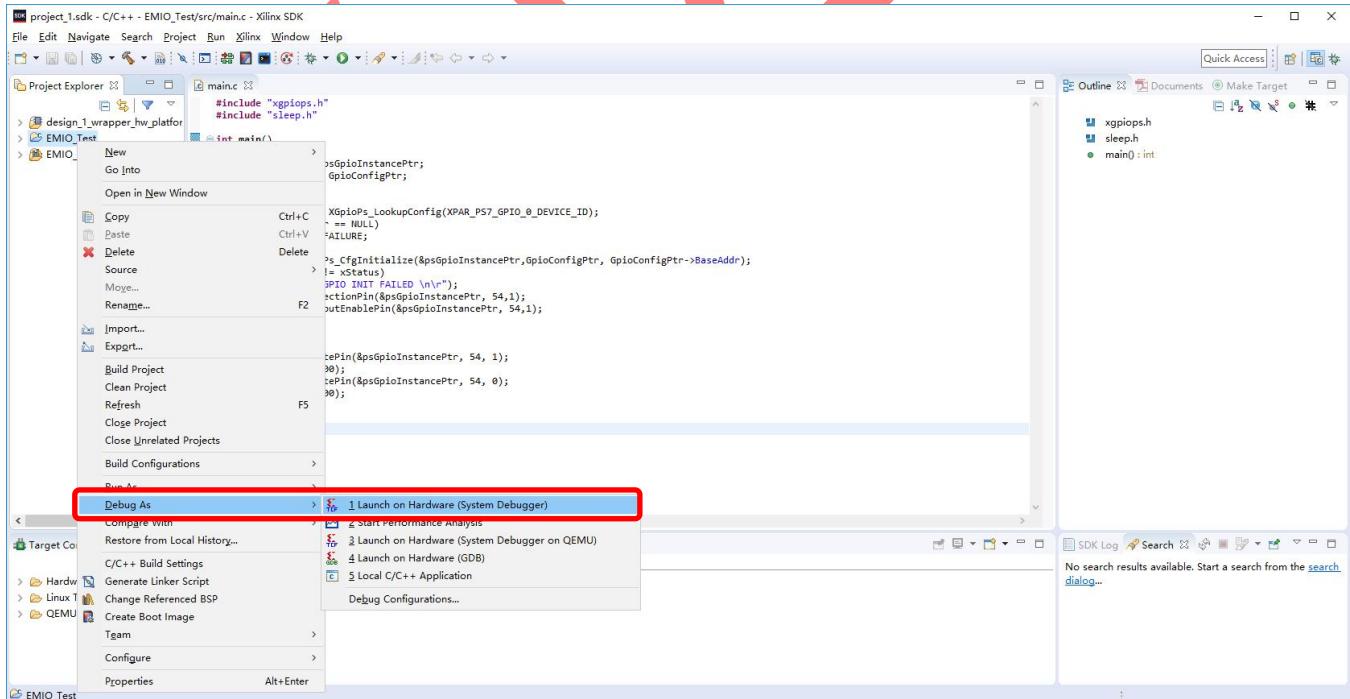


Figure 3-6. Start to Run

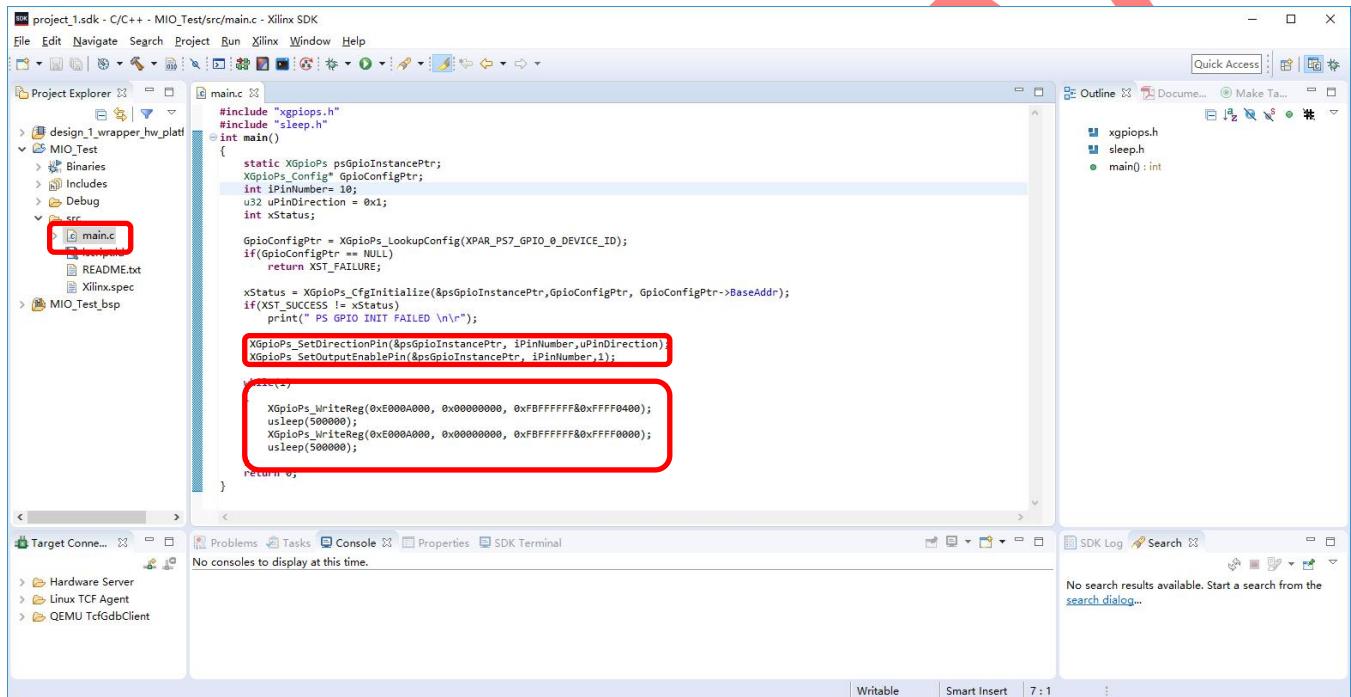
4. Experiment 2: MIO User LED

Before start to test the user LED connected to MIO10, users need to make sure the designer_1_wrapper is correctly generated in the previous step. And then execute the whole compilation procedure: **Run Synthesis**, **Run Implementation** and **Generate Bitstream**. Make sure there's no error generated in any of these three steps.

And then export hardware to the SDK and Lunch SDK.

In the SDK environment, create a new empty project named as MIO_Test.

Add the main.c file in MIO_Test/src folder and wait until the SDK project build finished. In the tet routine, it firstly configures the MIO10 as the output pin and enables it. In the main loop, it periodically toggles the MIO.



```
#include "xgpiops.h"
#include "sleep.h"

int main()
{
    static XGpioPs psGpioInstancePtr;
    XGpioPs_Config* GpioConfigPtr;
    int iPinNumber= 10;
    u32 uPinDirection = 0x1;
    int xStatus;

    GpioConfigPtr = XGpioPs_LookupConfig(XPAR_P57_GPIO_0_DEVICE_ID);
    if(GpioConfigPtr == NULL)
        return XST_FAILURE;

    xStatus = XGpioPs_CfgInitialize(&psGpioInstancePtr,GpioConfigPtr, GpioConfigPtr->BaseAddr);
    if(XST_SUCCESS != xStatus)
        print("PS GPIO INIT FAILED \n\r");

    XGpioPs_SetDirectionPin(&psGpioInstancePtr, iPinNumber,uPinDirection);
    XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, iPinNumber,1);

    while(1)
    {
        XGpioPs_WriteReg(0xE000A000, 0x00000000, 0xFFFFFFF&0xFFFF0400);
        usleep(500000);
        XGpioPs_WriteReg(0xE000A000, 0x00000000, 0xFFFFFFF&0xFFFF0000);
        usleep(500000);
    }

    return 0;
}
```

Figure 4-1. Toggle MIO10

Right click the project folder MIO_Test and then select **Run As→1 Launch on Hardware (System Debugger)**. And then users could see the D4 LED will be periodically blinking.

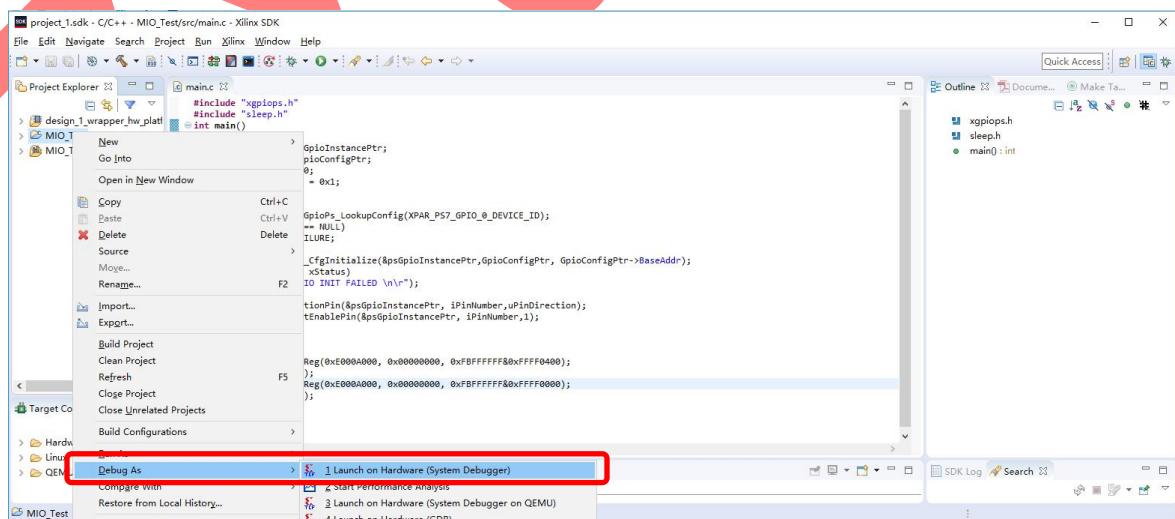


Figure 4-2. Start to Run

5. Experiment 3: DDR3 Test

Before start to test the DDR3 memory connected to PS side, users need to make sure the designer_1_wrapper is correctly generated in the previous step. And then execute the whole compilation procedure: **Run Synthesis**, **Run Implementation** and **Generate Bitstream**. Make sure there's no error generated in any of these three steps.

And then export hardware to the SDK and Lunch SDK.

In the SDK environment, create a new project named as DDR3_Test and click Next button.

Select Available Templates: Zynq DRAM tests and click Finish button.

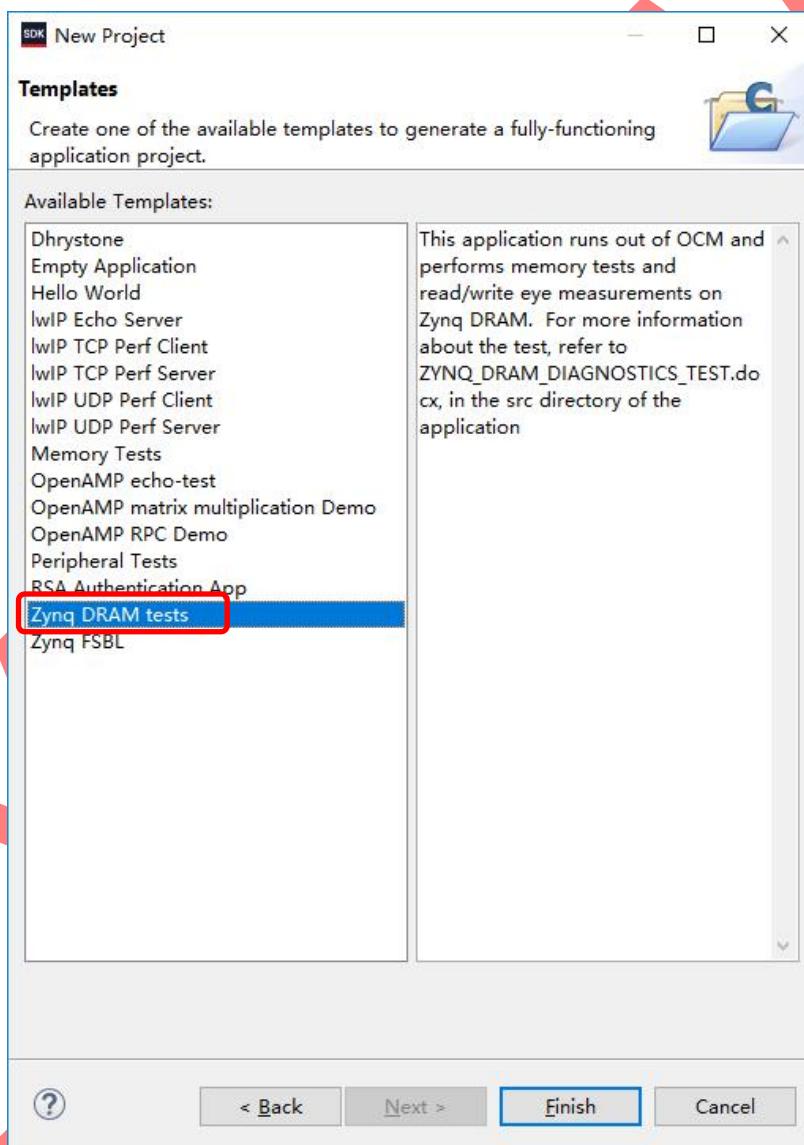


Figure 5-1. Create DDR3 Test Project

Below image shows the main function of the DDR3 test project.

The screenshot shows the Xilinx SDK IDE interface. The Project Explorer on the left lists the project structure, including the source file 'test01.c'. The code editor in the center displays the main() function. The function initializes memory, configures SLCR registers, and performs a memory test loop. The Outline view on the right shows various memory test functions. The bottom status bar indicates '激活 Windows' (Activate Windows) and '转到“设置”以激活 Windows.' (Go to 'Settings' to activate Windows.).

```

main()
{
    /* Function: main
     * Description:
     *     This function performs a memory test on the DDR3.
     *     It initializes memory, configures SLCR registers, and performs a memory test loop.
     */

    int abyte = 0x04*1024;
    int i,j,k,lp,a,go_rc,imaski,sel,last;
    int test_start,test_size,loop_cnt;
    int init,inax,itep,testsize_scl,fast_fix_center,printer;
    int tdm_idm,remote_mode,mem_test,cmd;
    char c[10];
    imemtest_enable = 1;

    rc = 0;

    // unlock slc
    REG_WRITE(SLCR_BASE+SLCR_UNLOCK, SLCR_UNLOCK_VALUE);
    REG_WRITE(SLCR_MODE, IBBU_DIS_MODE);
    clear_tdm_idm();

    // init data, go to 0
    REG_WRITE(MALBOX_GO, 0);
    REG_WRITE(MALBOX_DONE, 0);
    REG_WRITE(MALBOX_START, 1);
    REG_WRITE(MALBOX_MODE, 0x00FF);
    REG_WRITE(MALBOX_MODE, 0x00FF); // enable all 15 tests
    REG_WRITE(MALBOX_LAST, 0x0000);
    go = 0;
    // check wrap mode
    k = REG_READ(MALBOX_XREGV);
    remote_mode = 0;

    if (k == 0x55)
        remote_mode = 1;

    // Create the 128-word patterns
    for (i=0; i<128; i++)
    {
        imaski = (i >> 4) & 0x07; // invert mask index
    }
}

```

Figure 5-2. Main() Function

Right click the project folder DDR3_Test and then select **Run As→1 Launch on Hardware (System Debugger)**:

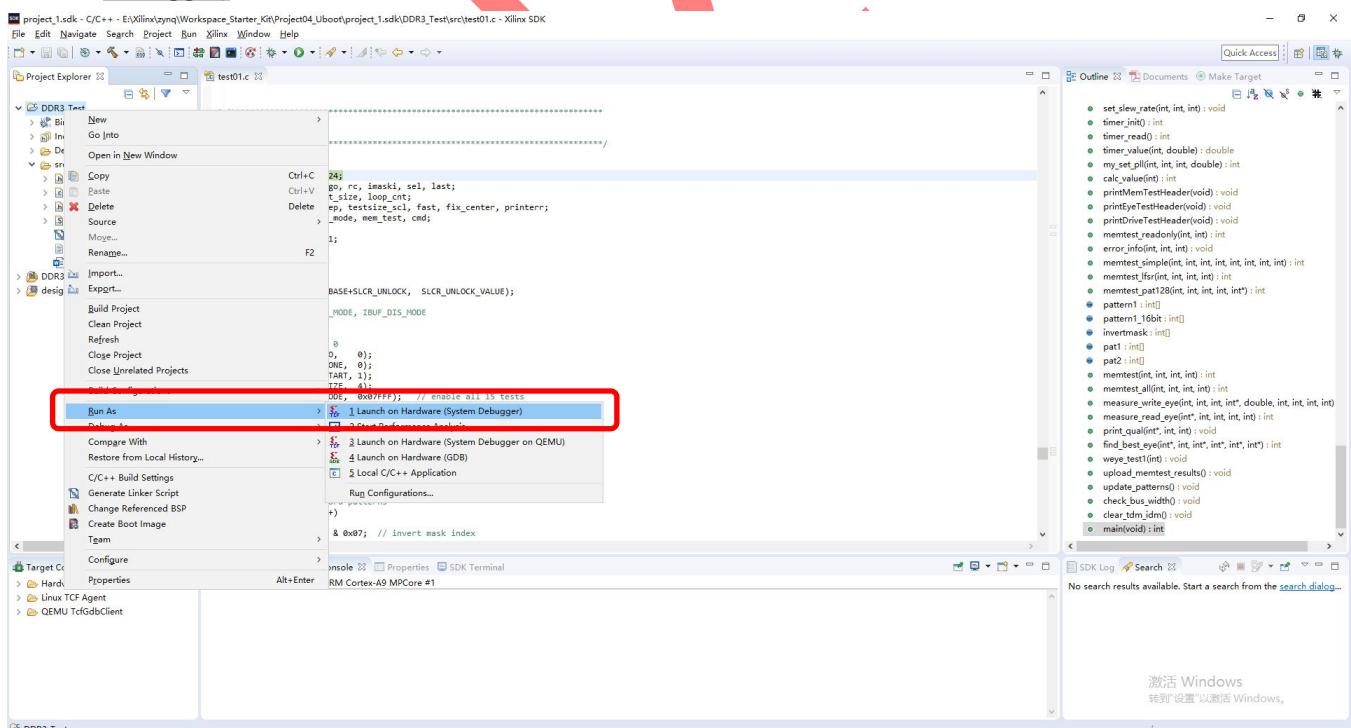
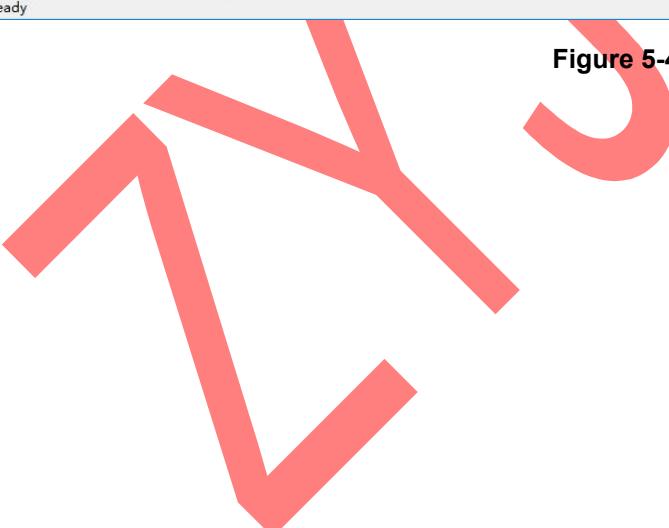


Figure 5-3. Launch Debugger

Use mini USB cable to connect the PC and the Zynq Starter Kit J4 connector. Below image shows the output log sent by the DDR3_Test example. And users could type '1', '2', etc. to select different test pattern. And the test results will also be displayed once the test finished.



serial-com5 - SecureCRT

File Edit View Options Transfer Script Tools Window Help

Enter host <Alt+R>

Session Manager

Sessions

- 192.168.1.10
- serial-com206
- serial-com207
- serial-com209
- serial-com214
- serial-com4
- serial-com5

serial-com5 x

```
----- ZYNQ DRAM DIAGNOSTICS TEST -----  
Select one of the options below:  
## Memory Test #  
Bus width = 32, XADC Temperature = 42,3573  
.1. - Test 1MB length from address 0x100000  
.2. - Test 32MB length from address 0x100000  
.3. - Test 64MB length from address 0x100000  
.4. - Test 128MB length from address 0x100000  
.5. - Test 256MB length from address 0x100000  
.6. - Test 512MB length from address 0x100000  
## Read Data Eye Measurement Test  
'r' - Measure Read Data Eye  
## Write Data Eye Measurement Test  
'i' - Measure Write Data Eye  
other options for write Eye Data Test:  
'f' - Fast Mode: Toggles Fast mode - ON/OFF  
'c' - Centre Mode: Toggles Centre mode - ON/OFF  
'e' - Vary the size of memory test for Read/Write Eye Measurement tests  
## Data Cache Enable / Disable option:  
'z' - D-Cache Enable / Disable  
## other options  
'v' - Verbose Mode ON/OFF  
option Selected : 3  
  
Starting Memory Test '3' - Testing 128MB length from address 0x100000...  
TEST WORD ERROR PER-BYTE-LANE ERROR COUNT TIME  
COUNT [ LANE-0 ] [ LANE-1 ] [ LANE-2 ] [ LANE-3 ] (sec)  
.....■
```

Ready

Serial: COM5, 115200 39, 7 52 Rows, 118 Cols VT100 CAP NUM

Figure 5-4. Test Pattern

6. Experiment 5: LwIP Test

Before start to test the LwIP running with the MII ethernet interface, users need to make sure the designer_1_wrapper is correctly generated in the previous step. And then execute the whole compilation procedure: **Run Synthesis**, **Run Implementation** and **Generate Bitstream**. Make sure there's no error generated in any of these three steps.

And then export hardware to the SDK and Lunch SDK.

In the SDK environment, create a new project named as LwIP_Test and click Next button.

Select Available Templates: LwIP Echo Server and click Finish button.

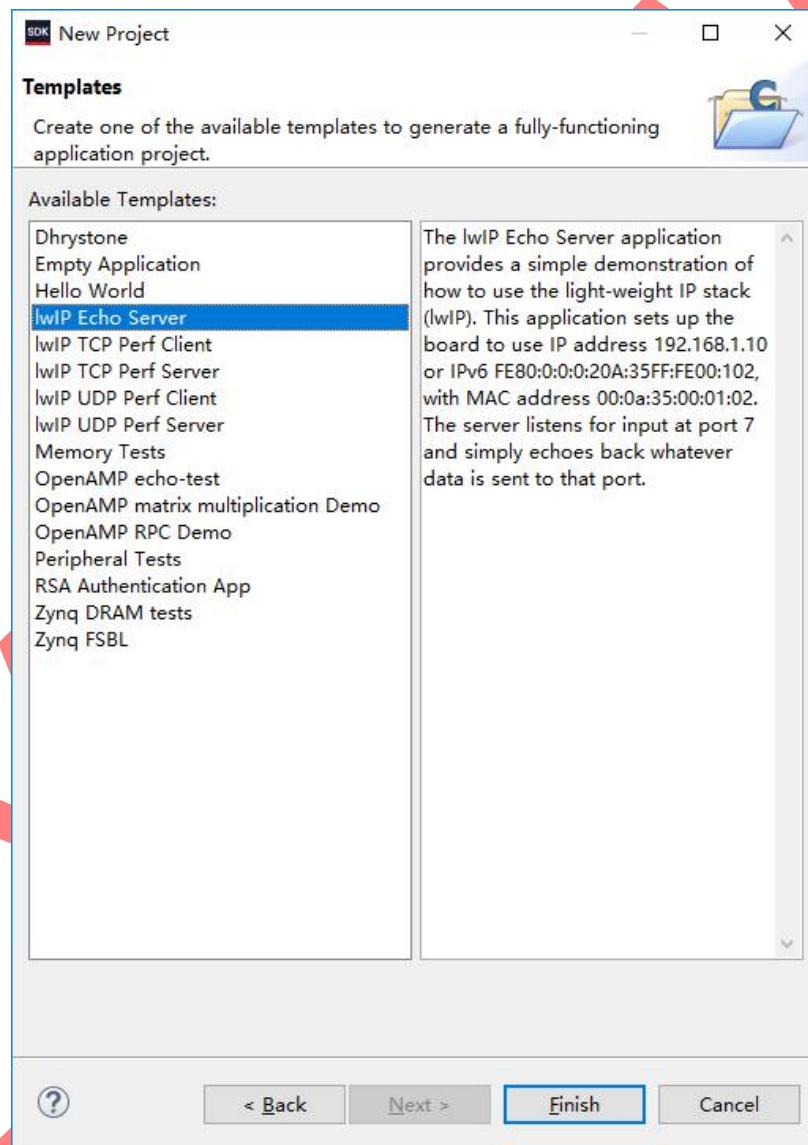


Figure 6-1. Create LwIP Test Project

Below image shows the main function of the LwIP test project. If DHCP server is not applicable, then set this static IP in the test routine directly.

```

project_1.sdk - C/C++ - LwIP_Test/src/main.c - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
Project Explorer
LwIP_Test
Binaries
Includes
Debug
echo.c
I2C_access.c
IIC_preset.c
main.c
platform_config.h
platform_mb.c
platform_pcc.c
platform_zynq.c
platform_zynqmpc.c
platformc
platformh
stpc
si5324.c
README.txt
LwIP.spec
LwIP_Test.bsp

main.c
/* now enable interrupts */
platform_enable_interrupts();

/* specify that the network if is up */
netif_set_up(echo_netif);

#if ((LWIP_IPV4 & LWIP_DHCPC) > 0)
    /* Create a new DHCP client for this interface.
     * Note: you must call dhcp_fine_tmr() and dhcp_coarse_tmr() at
     * the predefined regular intervals after starting the client.
     */
    dhcp_start(echo_netif);
    dhcp_timeoutcnr = 24;

    while(((echo_netif->ip_addr.addr == 0) && (dhcp_timeoutcnr > 0))
        xmemcif_input(echo_netif);

    if (dhcp_timeoutcnr <= 0)
        if ((echo_netif->ip_addr.addr == 0) {
            xil_printf("DHCP Timeout\r\n");
            xil_printf("Configuring default IP of 192.168.1.10\r\n");
            IP4_ADDR(&(echo_netif->ip_addr), 192, 168, 1, 10);
            IP4_ADDR(&(echo_netif->netmask), 255, 255, 255, 0);
            IP4_ADDR(&(echo_netif->gw), 192, 168, 2, 1);
        }
    }

    ipaddr.addr = echo_netif->ip_addr.addr;
    gw.addr = echo_netif->gw.addr;
    netmask.addr = echo_netif->netmask.addr;
#endif

/* start the application (web server, ctest, tctest, etc..) */
start_application();

/* receive and process packets */
while (1) {
    if (TcpFastTimeFlag) {
        ...
    }
}

```

Figure 6-2. Main() Function

Right click the project folder DDR3_Test and then select **Run As → 1 Launch on Hardware (System Debugger)**:

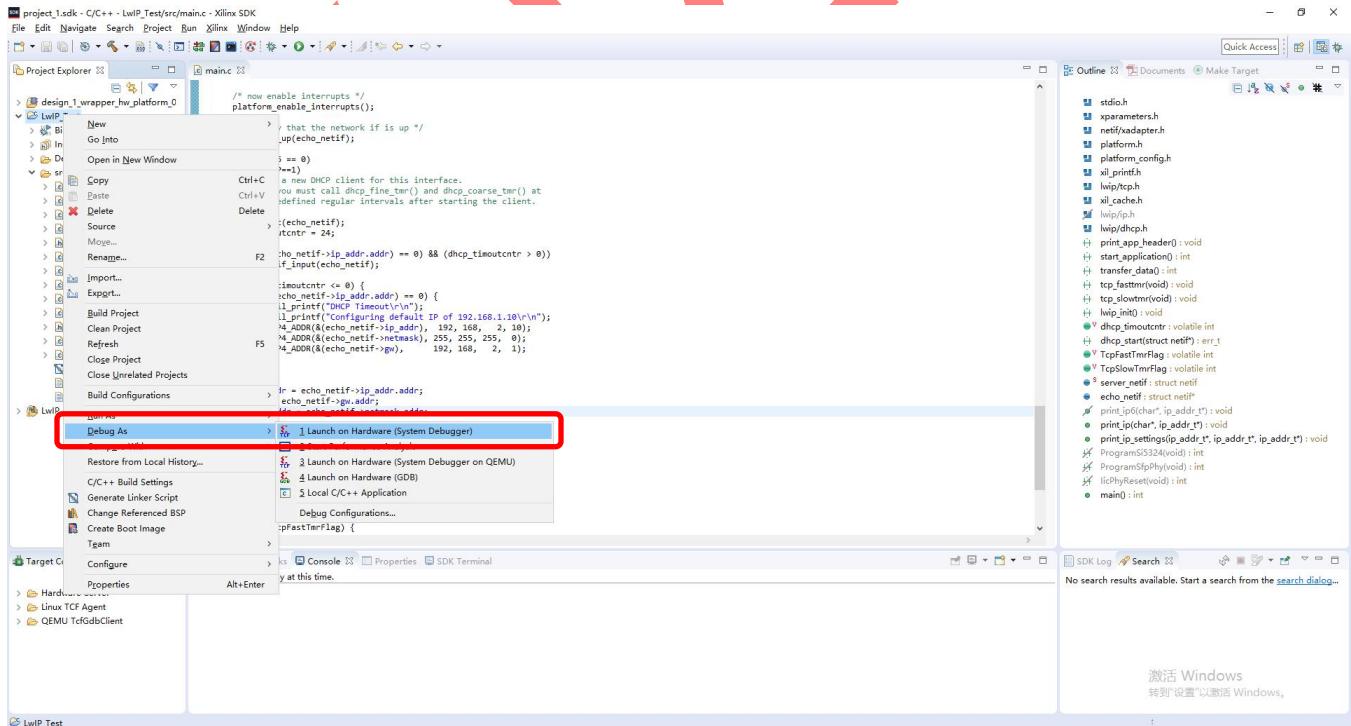


Figure 6-3. Launch Debugger

Use mini USB cable to connect the PC and the Zynq Starter Kit J4 connector. Below image shows the output log sent by the LwIP_Test example.

The screenshot shows the SecureCRT application window titled "serial-com5 - SecureCRT". The menu bar includes File, Edit, View, Options, Transfer, Script, Tools, Window, and Help. The toolbar has icons for session management, file operations, and communication. The main pane displays a session named "serial-com5". A red box highlights the terminal output which reads:

```
----lWIP TCP echo server ----
TCP packets sent to port 6001 will be echoed back
Start PHY autonegotiation
waiting for PHY to complete autonegotiation.
autonegotiation complete
Board Speed : 100M
Board IP: 192.168.2.3
Netmask : 255.255.255.0
Gateway : 192.168.2.1
TCP echo server started @ port 7
```

The bottom status bar shows "Ready", "Serial: COM5, 115200", "13. 1 52 Rows, 118 Cols", "VT100", and "CAP NUM".

Figure 6-4. Log Info

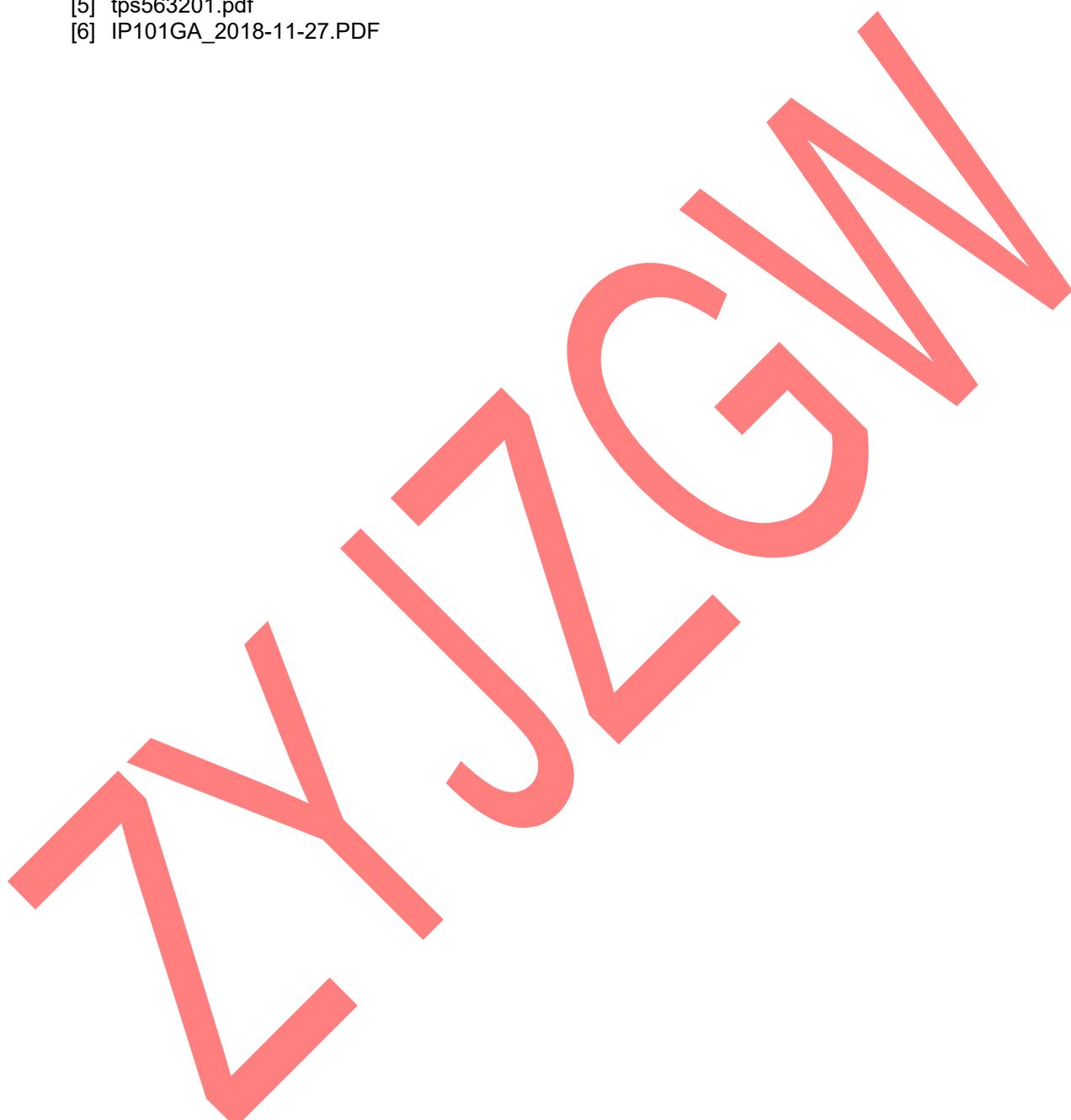
Open ethernet test assistant and set the configurations as shown in below image. TCP Client@192.168.2.3:7 and connect to server. "<http://www.cmssoft.cn>" is the message will be echo back from the server side.



Figure 6-5. Echo Message

7. Reference

- [1] ug585-Zynq-7000-TRM.pdf
- [2] ds187-XC7Z010-XC7Z020-Data-Sheet.pdf
- [3] ug865-Zynq-7000-Pkg-Pinout.pdf
- [4] MT41K256M16TW-107:P.pdf
- [5] tps563201.pdf
- [6] IP101GA_2018-11-27.PDF



8. Revision

Doc. Rev.	Date	Comments
0.1	13/06/2021	Initial Version.
1.0	23/06/2021	V1.0 Formal Release.

A large, stylized watermark in red ink, reading "ZYJZGW" in a bold, blocky font. The letters are slightly curved and overlap each other, creating a sense of depth. The watermark is positioned in the lower half of the page, spanning from the left edge to the right edge.