zynq[1] mine board helloworld

2019-04-26 8554

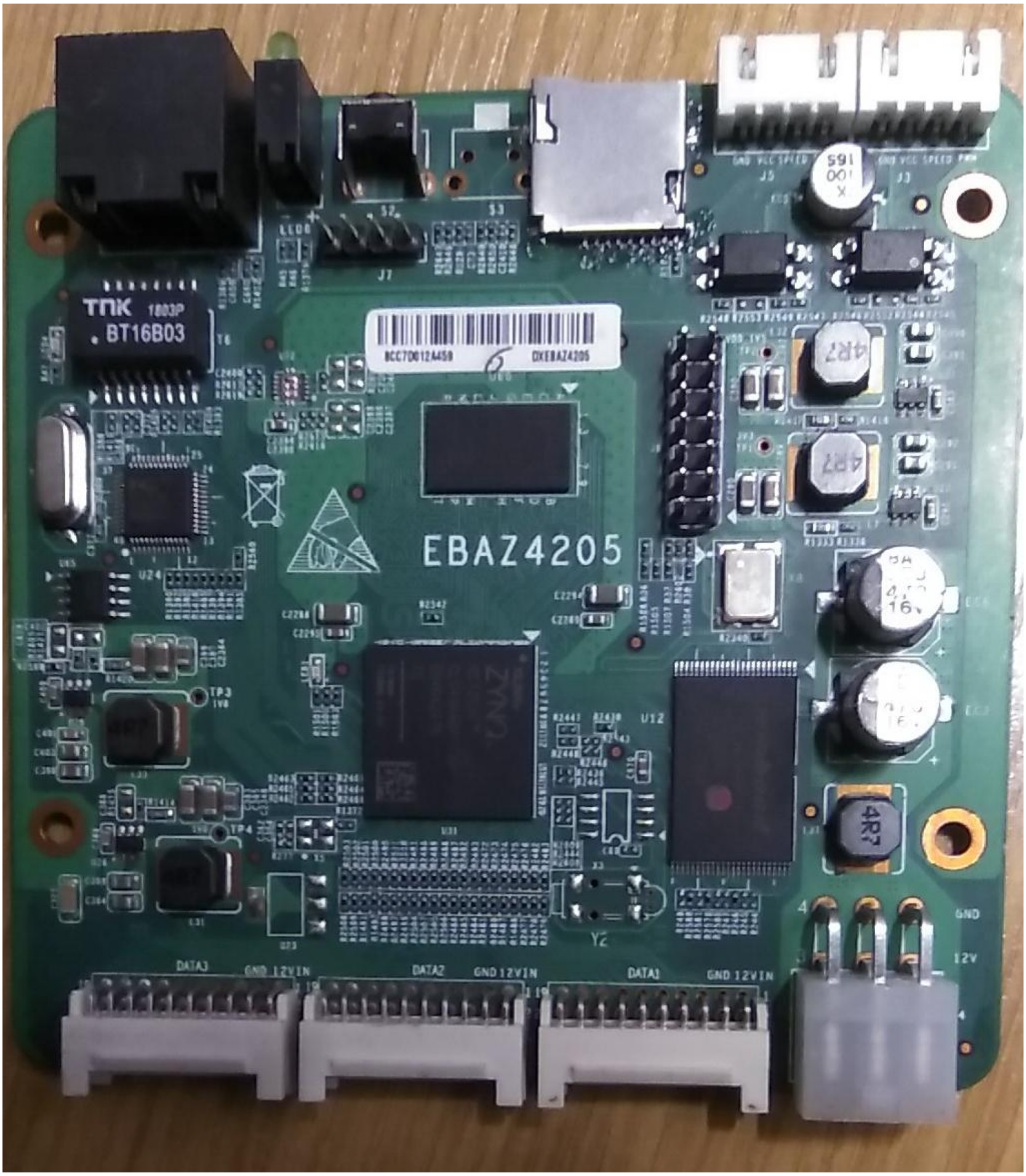
In recent mining disasters, a large number of mining machine control panels appeared on a certain treasure and a certain fish. . . ASIC is responsible for mining, and the control board is responsible for networking, monitoring, and control. There are many kinds of fruit pies on the control panel. The so-called "geek toys" such as orange pie, raspberry pie, and dog bones are among them. Presumably, domestic fruit pie manufacturers are very moisturizing. . . Geek education alone makes no money. Most geeks are poor. Seeing such a cheap "zynq development board" like me, I can't help but follow the trend and collect trash. . .

There was a `EBAZ4205` flood of control boards on a certain fish , which came from `翼比特E9+` mining machines. Its old version control card `EBAZ4203` configuration is basically the same.

ebaz4205

The big pile below is the computing card, the small one above is the control board

oard overview



ebaz4205

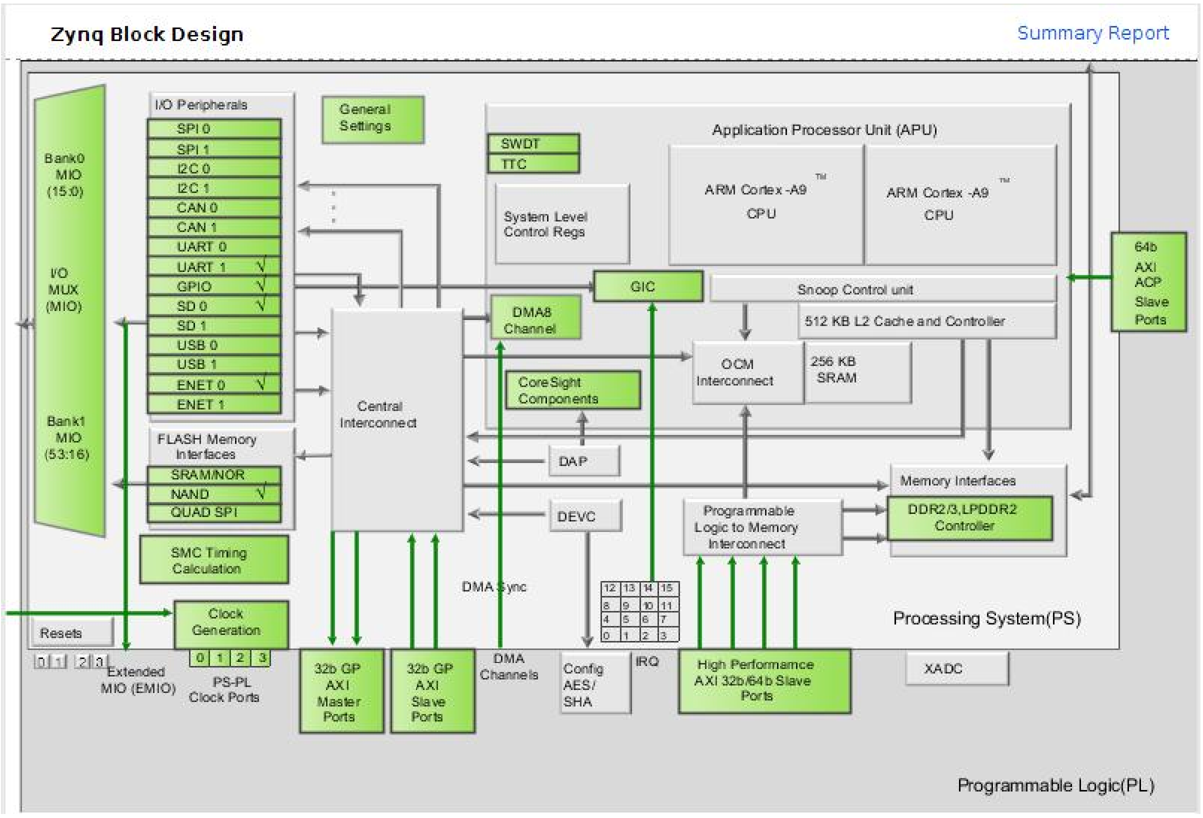
Master	XC7Z010CLG400-1
RAM	256MB DDR3, EM6GD16EWKG or MT41K128M16
nand	128MB SLC
Ethernet	100M network card, IP101GA
powered by	5V also works

other	TF card, UART1, 2 fan ports, 14-pin jtag, 3 20pin IO ports

evelopment Project

OC startup configuration

ZYNQ series SOC integrates dual-core ARM Cortex-A9 and FPGA. The entire SOC is divided into `PS` (processing system) and `PL` (programmable logic) two parts. PS includes processor, on-chip AMBA bus, memory controller, some peripherals and fixed IO ports; PL is FPGA. The ZYNQ 7010 processor's main frequency can reach about 600MHz, and the FPGA has about 28K LEs.



zynq

The PS part of ZYNQ can run independently without the PL part, because the peripherals of the PS are bound with some IO ports (`MIO`) by default , and the IO ports such as the memory controller are still unchangeable. At this time, you can develop other ARM Play ZYNQ like SOC. MIO is limited. If some peripheral ports conflict, they can `EMIO` be led out by bypassing PL. At this time, the PL part needs to be taken care of.

The start of ZYNQ is divided into three steps:

- `BOOT ROM` , Choose where to start according to pin configuration, such as QSPI, nand/nor flash, SD card, etc. The `FSBL` (first stage bootloader) loaded onto the memory chip. 7010 has 256k on-chip memory.
- `FSBL` , Initialize more MIO ports, initialize DDR, you can also initialize the PL part, and then move the application to the DDR. The initialization part is directly generated by vivado, that is, tens of thousands of lines `ps7_init.c` . FSBL can directly use the Xilinx SDK example project, which is equivalent to just a few clicks of the mouse.
- `应用程序` . It can be directly the user's application, or it can be another loader, such as uboot, and the rest is up to the programmer to decide.

It should be noted that FSBL is equivalent to the status of uboot SPL. In the Xilinx uboot project, the tens of thousands of lines of initialization c program is compiled into SPL, so if you use SDK FSBL, uboot SPL is not needed.

The `R2577` sum `R2584` resistor of this board is used to configure the starting device. The `R2584` welded to `R2577` , the original nand will be changed to start the SD card starts.

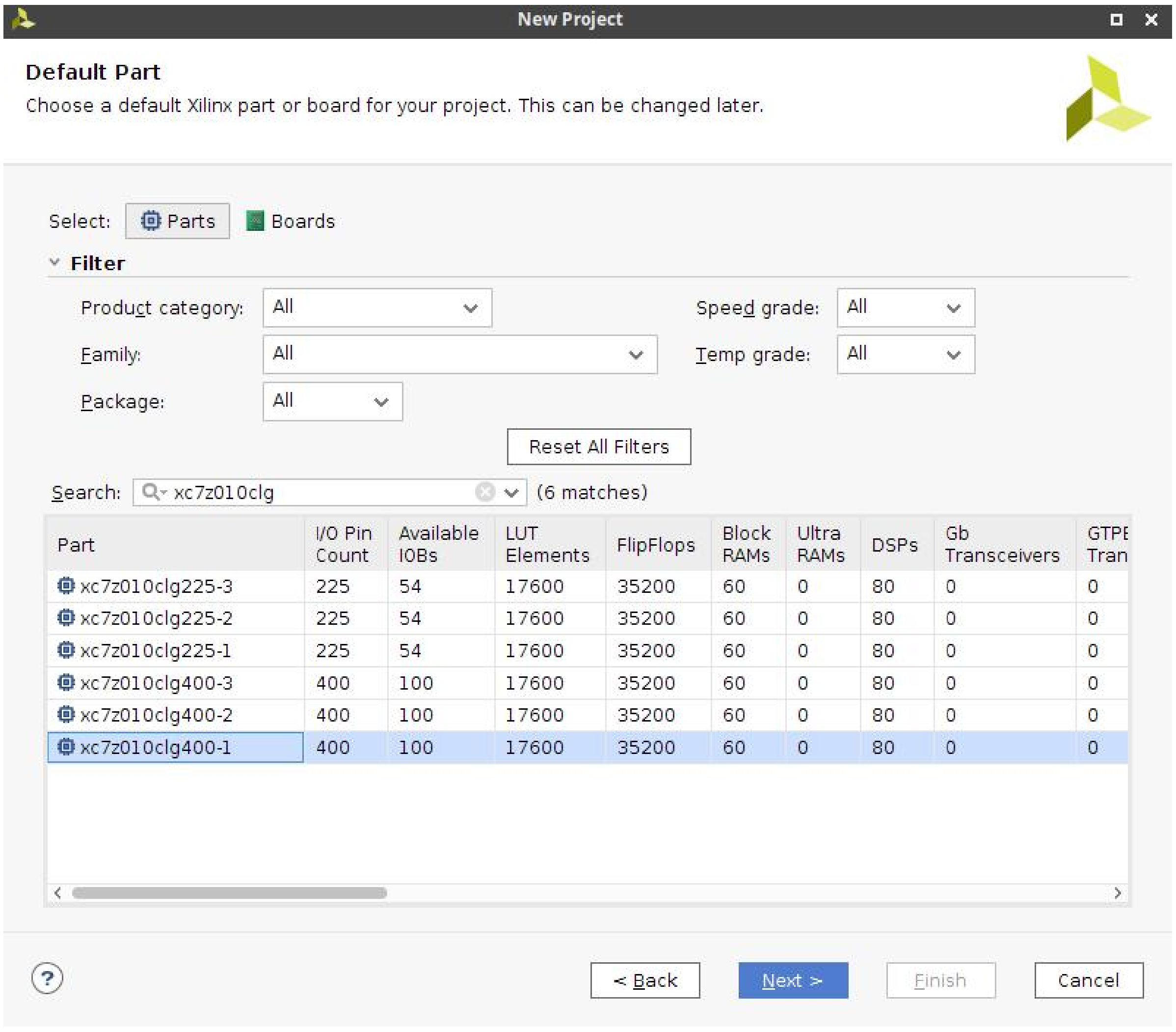
vivado

/ivado operation process

Using the cumbersome development environment of Xilinx, the helloworld project can be operated with the mouse, without writing a single line of code. .

-irst create a new project

Continue to next until the chip is selected. selected `xc7z010clg400-1`



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter

Product category: All

Speed grade: All

Family: All

Temp grade: All

Package: All

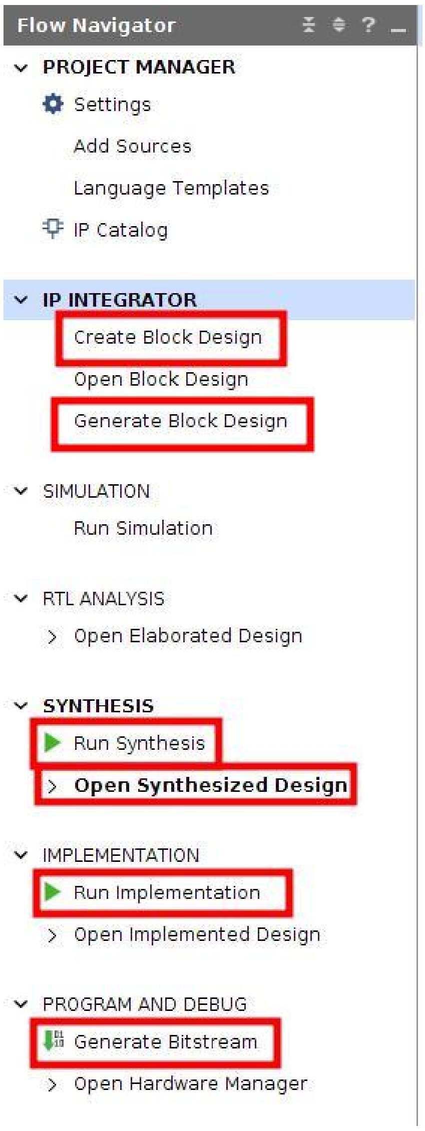
Reset All Filters

Search: xc7z010clg (6 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE Tran
xc7z010clg225-3	225	54	17600	35200	60	0	80	0	0
xc7z010clg225-2	225	54	17600	35200	60	0	80	0	0
xc7z010clg225-1	225	54	17600	35200	60	0	80	0	0
xc7z010clg400-3	400	100	17600	35200	60	0	80	0	0
xc7z010clg400-2	400	100	17600	35200	60	0	80	0	0
xc7z010clg400-1	400	100	17600	35200	60	0	80	0	0

vivado

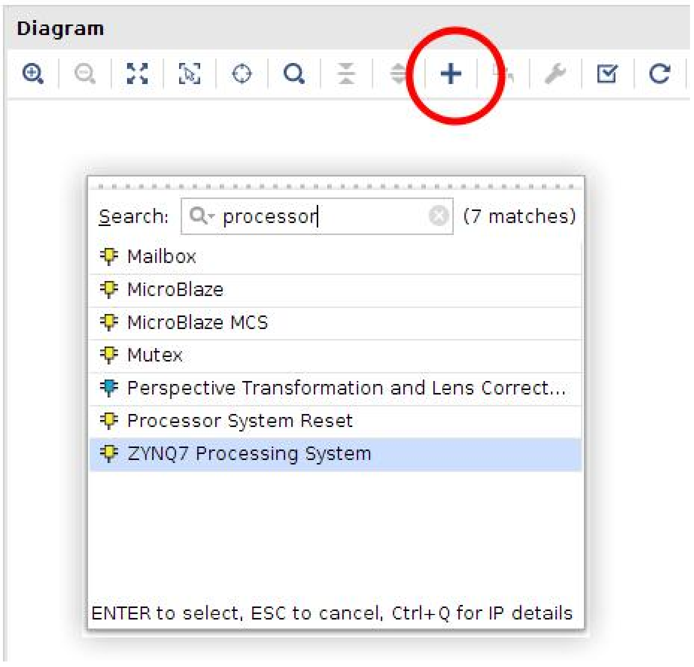
Then follow the left column `Flow Navigator` to operate.



vivado

reate Board Design

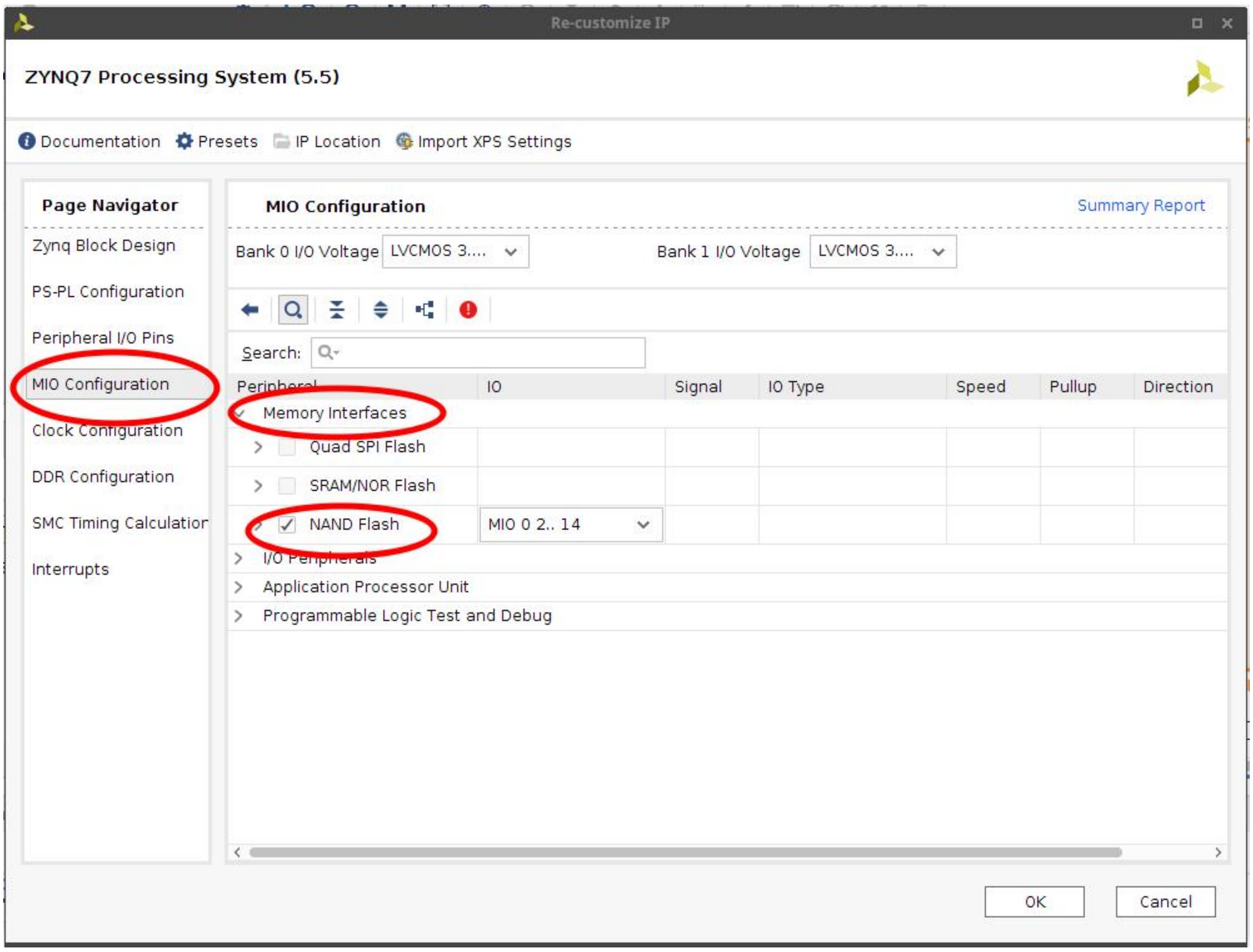
Click the plus sign to add `ZYNQ7 Processing System` :



vivado

Double-click the blue box of zynq7 processing system to configure the PS system:

- Add the nand controller, the default will do:



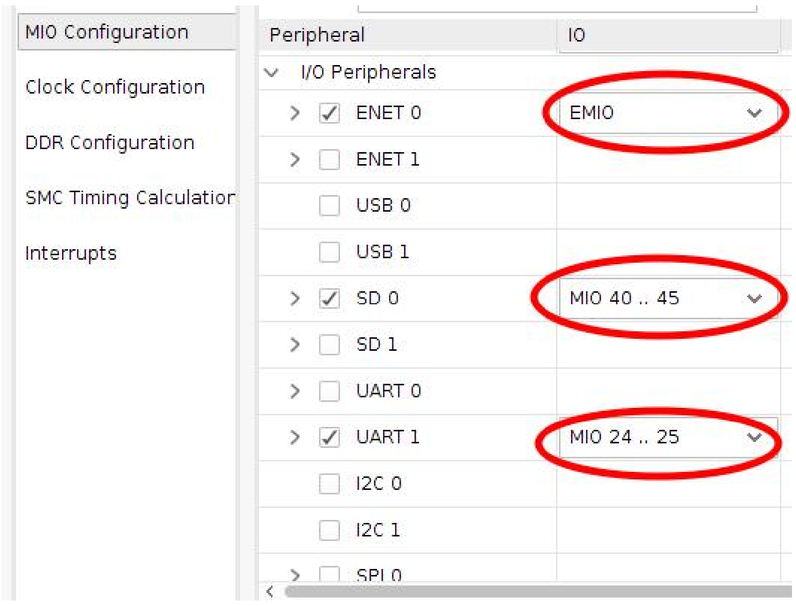
MIO Configuration

Bank 0 I/O Voltage LVCMOS 3.... Bank 1 I/O Voltage LVCMOS 3....

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction
Memory Interfaces						
Quad SPI Flash						
SRAM/NOR Flash						
✓ NAND Flash	MIO 0 2.. 14					
I/O Peripherals						
Application Processor Unit						
Programmable Logic Test and Debug						

vivado

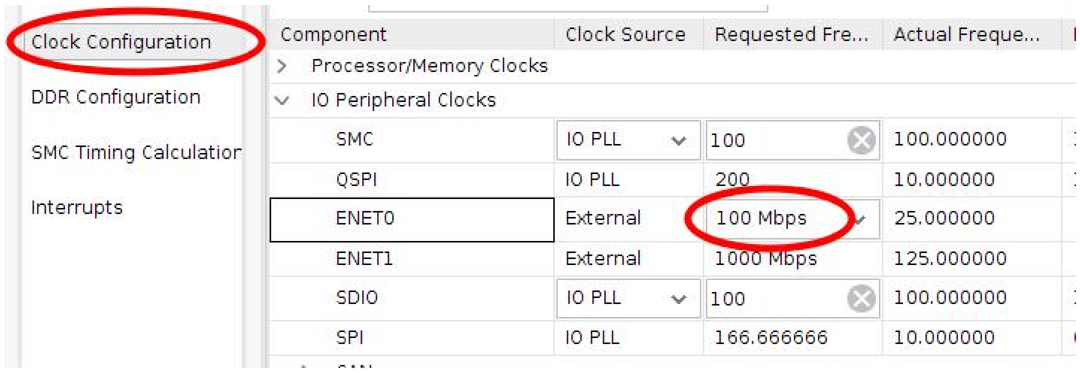
- Add MIO settings. Check ENET0, SD0, UART1. Pay attention to the pin configuration:



Peripheral	IO
I/O Peripherals	
✓ ENET 0	EMIO
ENET 1	
USB 0	
USB 1	
✓ SD 0	MIO 40 .. 45
SD 1	
UART 0	
✓ UART 1	MIO 24 .. 25
I2C 0	
I2C 1	
SPI 0	

vivado

- Set the peripheral clock. Change the network port to 100M:

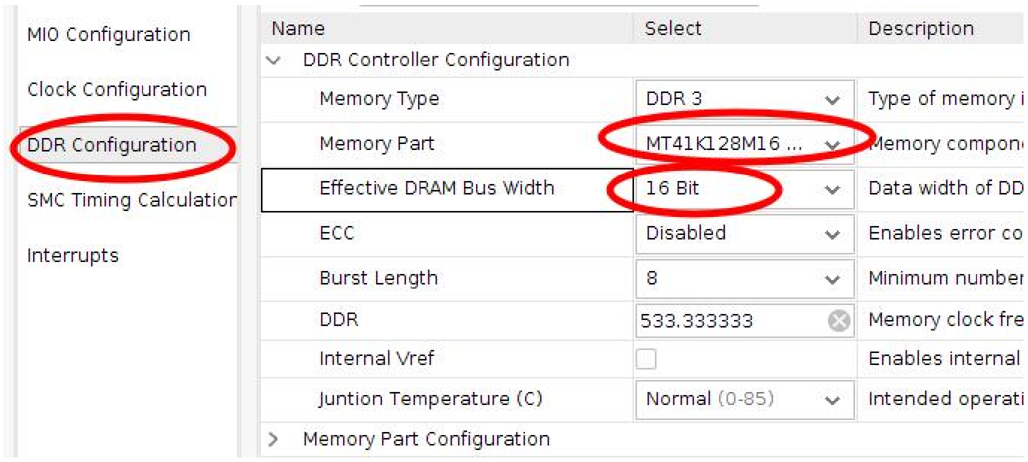


Clock Configuration
DDR Configuration
SMC Timing Calculation
Interrupts

Component	Clock Source	Requested Fre...	Actual Freque...
> Processor/Memory Clocks			
⌄ IO Peripheral Clocks			
SMC	IO PLL	100	100.000000
QSPI	IO PLL	200	10.000000
ENET0	External	100 Mbps	25.000000
ENET1	External	1000 Mbps	125.000000
SDIO	IO PLL	100	100.000000
SPI	IO PLL	166.666666	10.000000

vivado

- Set up DDR. Fortunately, ours `MT41K128M16` has default parameter configuration:

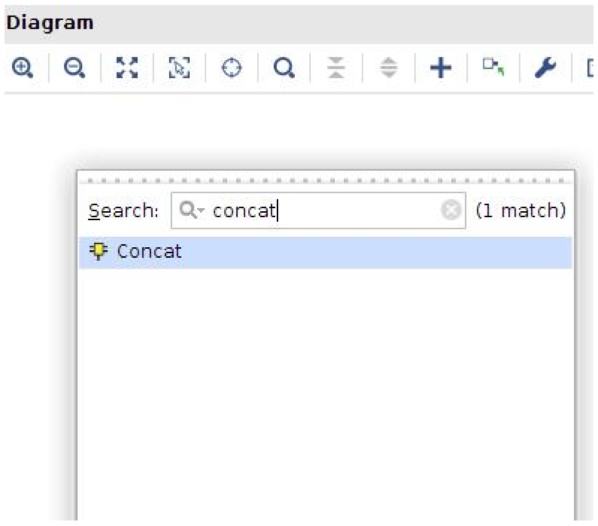


MIO Configuration
Clock Configuration
DDR Configuration
SMC Timing Calculation
Interrupts

Name	Select	Description
⌄ DDR Controller Configuration		
Memory Type	DDR 3	Type of memory i
Memory Part	MT41K128M16 ...	Memory compon
Effective DRAM Bus Width	16 Bit	Data width of DD
ECC	Disabled	Enables error co
Burst Length	8	Minimum number
DDR	533.333333	Memory clock fre
Internal Vref	☐	Enables internal
Juntion Temperature (C)	Normal (0-85)	Intended operati
> Memory Part Configuration		

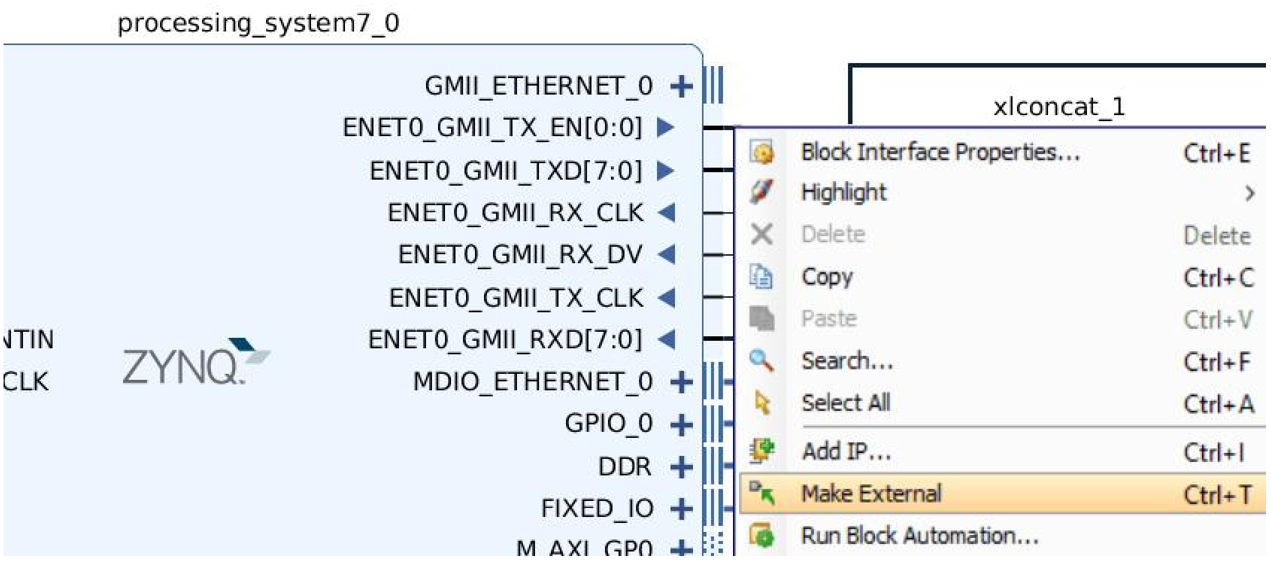
vivado

Because the network port of the board is led out through EMIO, we need to manually assign the pins one by one. But this IP is a GMII Gigabit Ethernet port, TX RX has 8 bits, while the MII interface TX RX used by a 100M network card has only 4 bits, **you must explicitly add two concat modules to convert 8 bits to 4 bits** , otherwise it will be redundant If the pin is led out but no IO port is allocated, an error will be reported when the bitstream is finally generated.



vivado

Lead the TX and RX of GMII to their respective concat modules, and lead other pins of GMII and MDIO: right click `Make External`

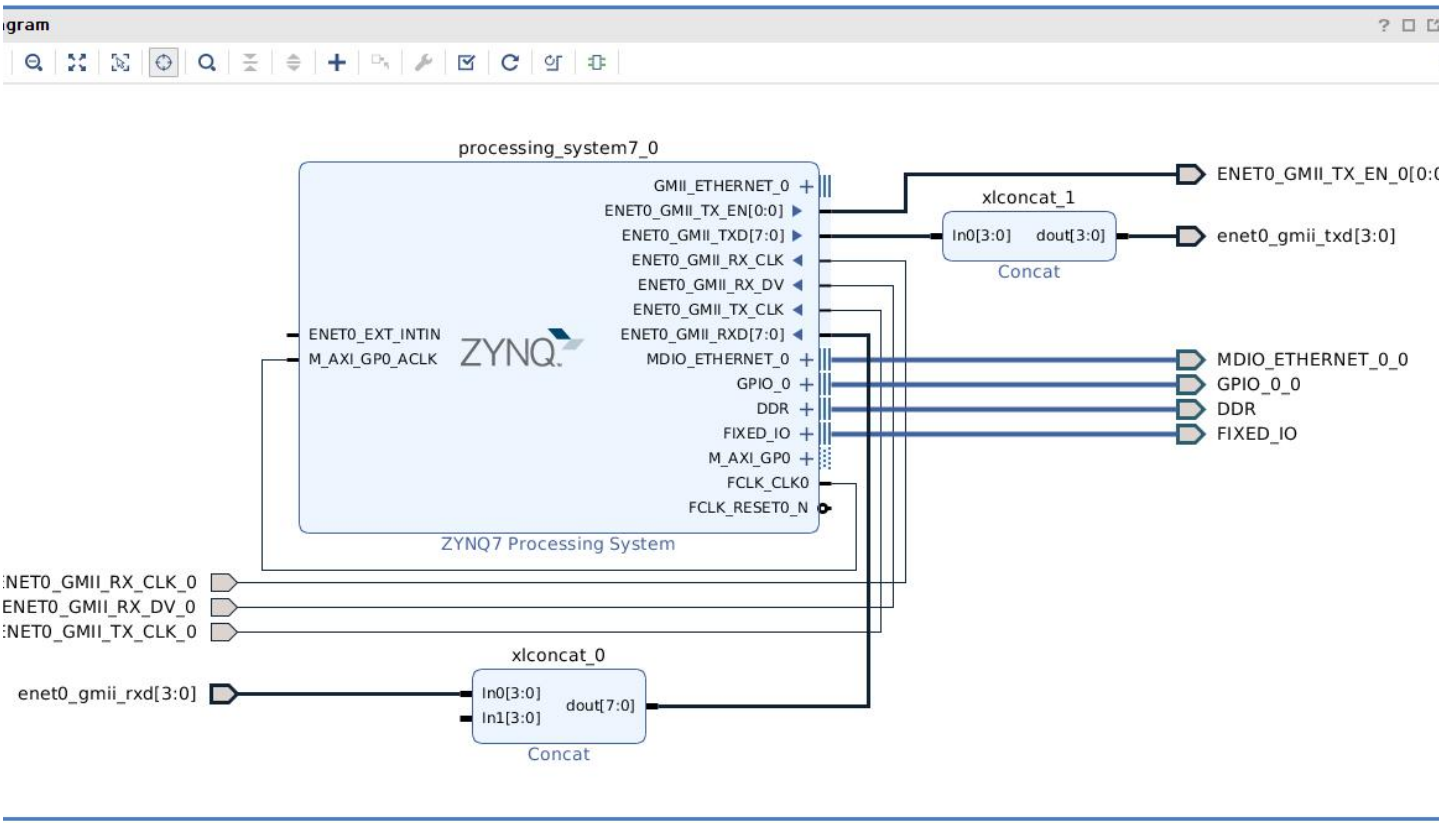


vivado

The `FCLK_CLK0` follow `M_AXI_GP0_ACLK` all together, and finally click on the top bar `Run Block Automation` do the rest.

vivado

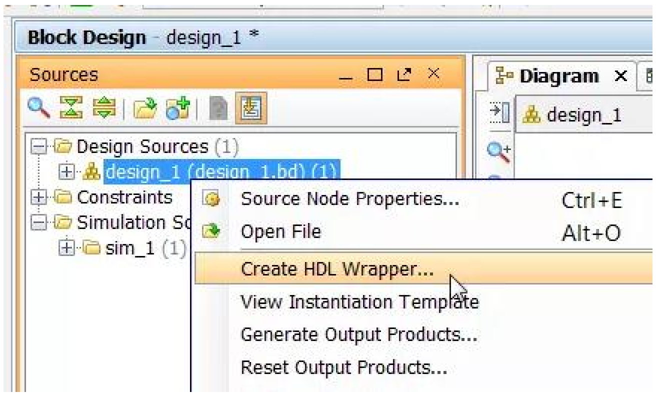
The effect after completion:



vivado

Generate Block Design on the left column

First `Generate Block Design` , and then right `Source` in the box below bd file `Create HDL Wrapper` :

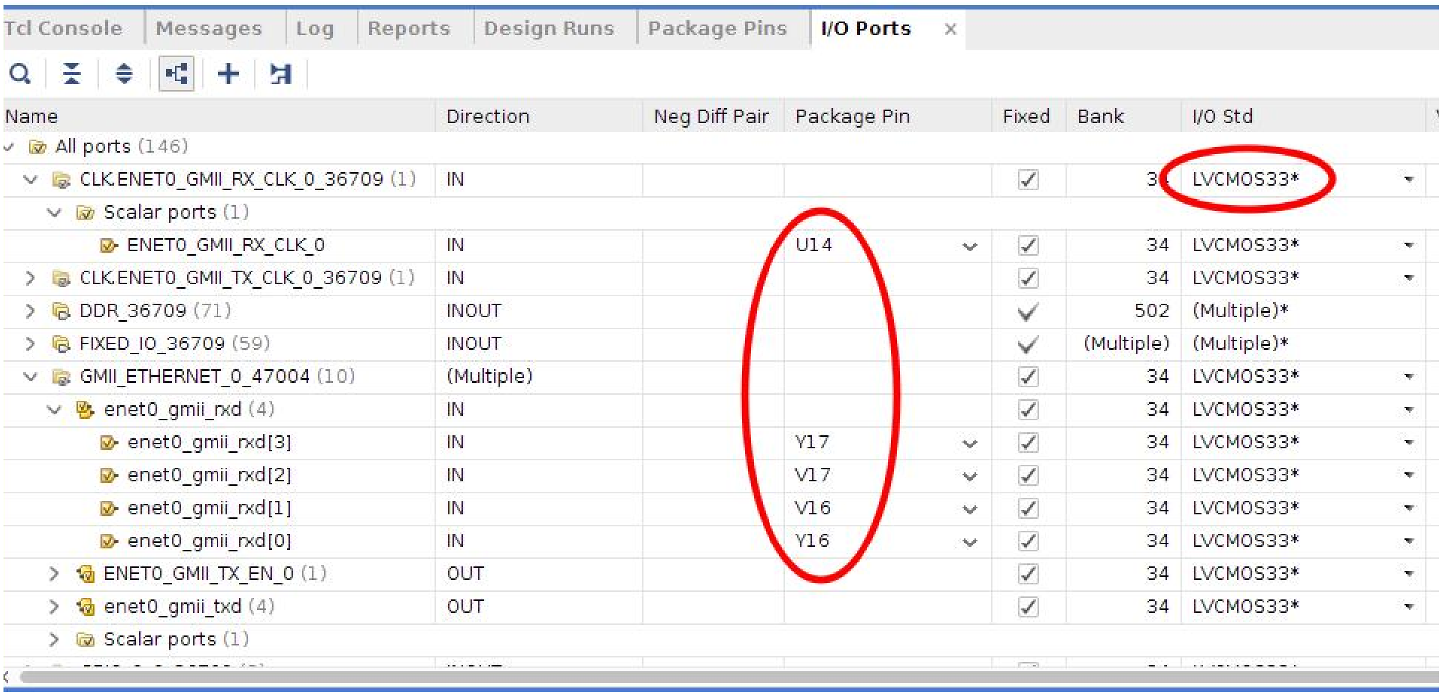

Block Design - design_1 *
Sources
Diagram
design_1
Design Sources (1)
design_1 (design_1.bd) (1)
Constraints
Simulation Sc
sim_1 (1)
Source Node Properties... Ctrl+E
Open File Alt+O
Create HDL Wrapper...
View Instantiation Template
Generate Output Products...
Reset Output Products...

vivado

Run Synthesis on the left column

First synthesize once, then turn on `Open Synthesized Design` -> `Constraint Wizard` to assign pins. At this time, you need to create a new constraint file.

Set all the pins of the network port to `LVCMOS33` level, and then assign the pins one by one. . .



Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (146)						
CLK.ENET0_GMII_RX_CLK_0_36709 (1)	IN			✓	34	LVCMOS33*
Scalar ports (1)						
ENET0_GMII_RX_CLK_0	IN		U14	✓	34	LVCMOS33*
CLK.ENET0_GMII_TX_CLK_0_36709 (1)	IN			✓	34	LVCMOS33*
DDR_36709 (71)	INOUT			✓	502	(Multiple)*
FIXED_IO_36709 (59)	INOUT			✓	(Multiple)	(Multiple)*
GMII_ETHERNET_0_47004 (10)	(Multiple)			✓	34	LVCMOS33*
enet0_gmii_rxd (4)	IN			✓	34	LVCMOS33*
enet0_gmii_rxd[3]	IN		Y17	✓	34	LVCMOS33*
enet0_gmii_rxd[2]	IN		V17	✓	34	LVCMOS33*
enet0_gmii_rxd[1]	IN		V16	✓	34	LVCMOS33*
enet0_gmii_rxd[0]	IN		Y16	✓	34	LVCMOS33*
ENET0_GMII_TX_EN_0 (1)	OUT			✓	34	LVCMOS33*
enet0_gmii_txd (4)	OUT			✓	34	LVCMOS33*
Scalar ports (1)						

vivado

`Ctrl-S` Save after setting , the constraint file is generated as follows:

```
set_property IOSTANDARD LVCMOS33 [get_ports ENET0_GMII_RX_CLK_0]
...
set_property PACKAGE_PIN U14 [get_ports ENET0_GMII_RX_CLK_0]
set_property PACKAGE_PIN U15 [get_ports ENET0_GMII_TX_CLK_0]
set_property PACKAGE_PIN W19 [get_ports {ENET0_GMII_TX_EN_0[0]}]
set_property PACKAGE_PIN W18 [get_ports {enet0_gmii_txd[0]}]
set_property PACKAGE_PIN Y18 [get_ports {enet0_gmii_txd[1]}]
set_property PACKAGE_PIN V18 [get_ports {enet0_gmii_txd[2]}]
set_property PACKAGE_PIN Y19 [get_ports {enet0_gmii_txd[3]}]
set_property PACKAGE_PIN W16 [get_ports ENET0_GMII_RX_DV_0]
set_property PACKAGE_PIN W15 [get_ports MDIO_ETHERNET_0_0_mdc]
set_property PACKAGE_PIN Y14 [get_ports MDIO_ETHERNET_0_0_mdio_io]
set_property PACKAGE_PIN Y16 [get_ports {enet0_gmii_rxd[0]}]
set_property PACKAGE_PIN V16 [get_ports {enet0_gmii_rxd[1]}]
set_property PACKAGE_PIN V17 [get_ports {enet0_gmii_rxd[2]}]
set_property PACKAGE_PIN Y17 [get_ports {enet0_gmii_rxd[3]}]
```

Then synthesize it again. . .

Run Implementation and Generate Bitstream in the left column

It takes a while to run. . .

Output design

`File` -> `Export` -> `Export Hardware`

Remember to tick it `Include Bitstream` .

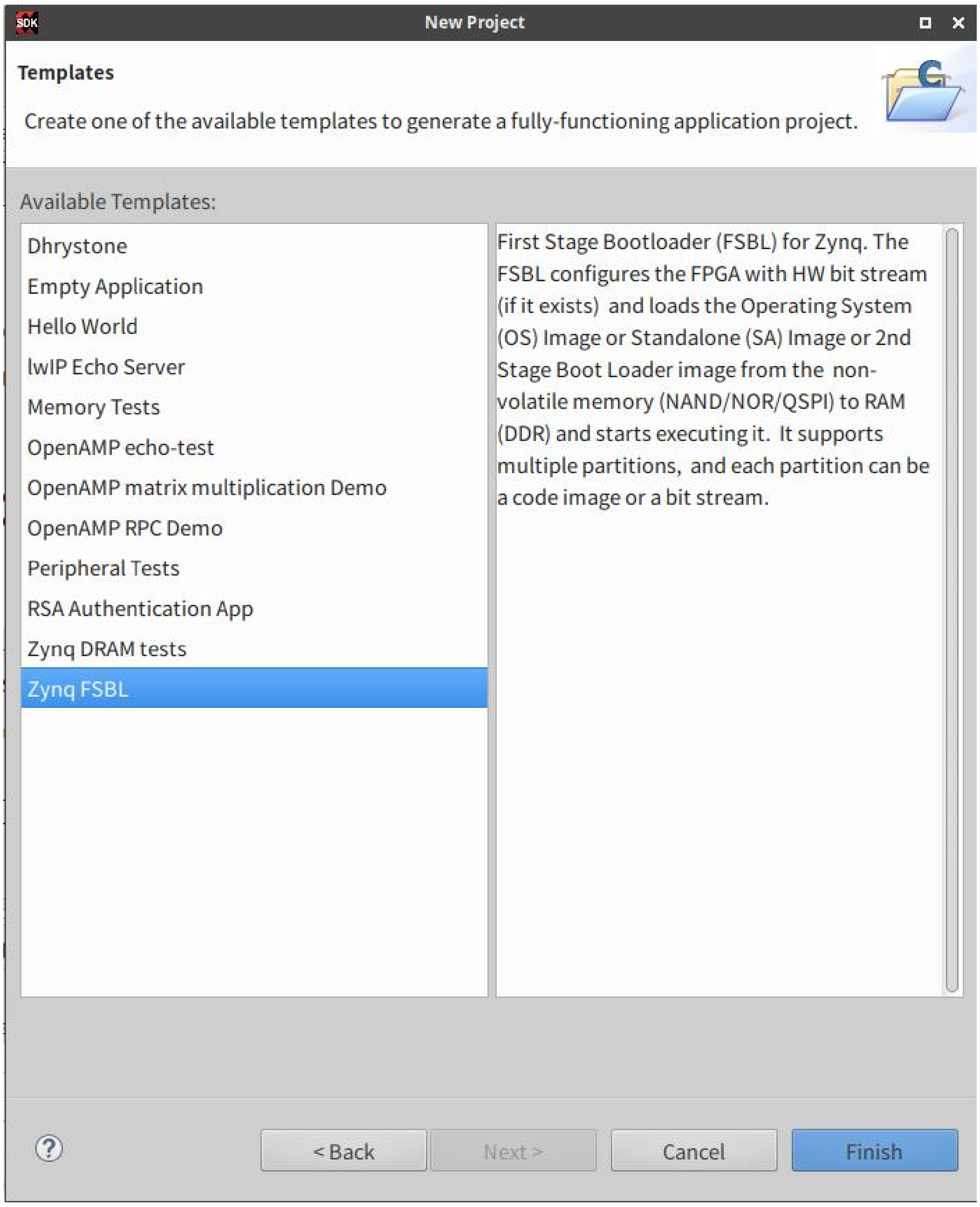
Enter SDK

`File` -> `Launch SDK`

SDK operation process

First you need to create a new FSBL.

`File` -> `New` -> `Application Project` , Next to `Templates` , choose `Zynq FSBL` :


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

sdk

Then it will automatically start to compile. . .

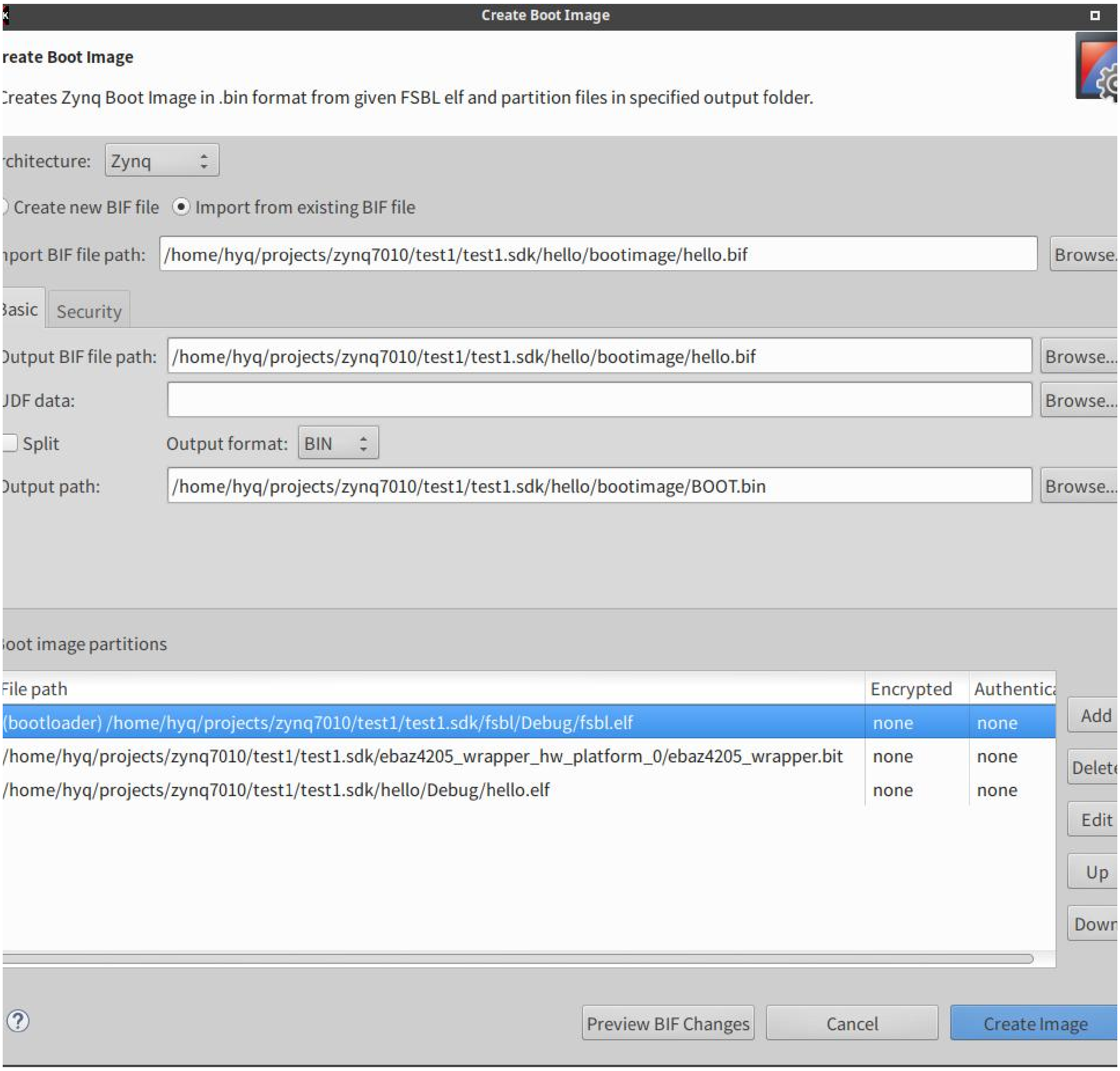
Then create a new helloworld. .

`File` -> `New` -> `Application Project` , Next `Templates` , choose `Hello World` . It will also start to compile automatically, and it will be over if nothing happens.

Finally, the startup file BOOT.bin is generated

Right-left column helloworld project `Create Boot Image` . You can see that this startup file includes three parts:

- fsbl
- bitstream
- application


Create Boot Image

reate Boot Image

:reates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

chitecture: Zynq

Create new BIF file [✓ Import from existing BIF file]

nport BIF file path: /home/hyq/projects/zynq7010/test1/test1.sdk/hello/bootimage/hello.bif

Basic | Security

Output BIF file path: /home/hyq/projects/zynq7010/test1/test1.sdk/hello/bootimage/hello.bif

JDF data:

Split   Output format: BIN

Output path: /home/hyq/projects/zynq7010/test1/test1.sdk/hello/bootimage/BOOT.bin

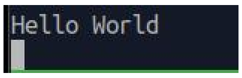
oot image partitions

File path	Encrypted	Authentic
(bootloader) /home/hyq/projects/zynq7010/test1/test1.sdk/fsbl/Debug/fsbl.elf	none	none
/home/hyq/projects/zynq7010/test1/test1.sdk/ebaz4205_wrapper_hw_platform_0/ebaz4205_wrapper.bit	none	none
/home/hyq/projects/zynq7010/test1/test1.sdk/hello/Debug/hello.elf	none	none

sdk

.ight up the board

Format an SD card as fat file system, `bootimage/BOOT.bin` throw it in, plug it in and start it up.

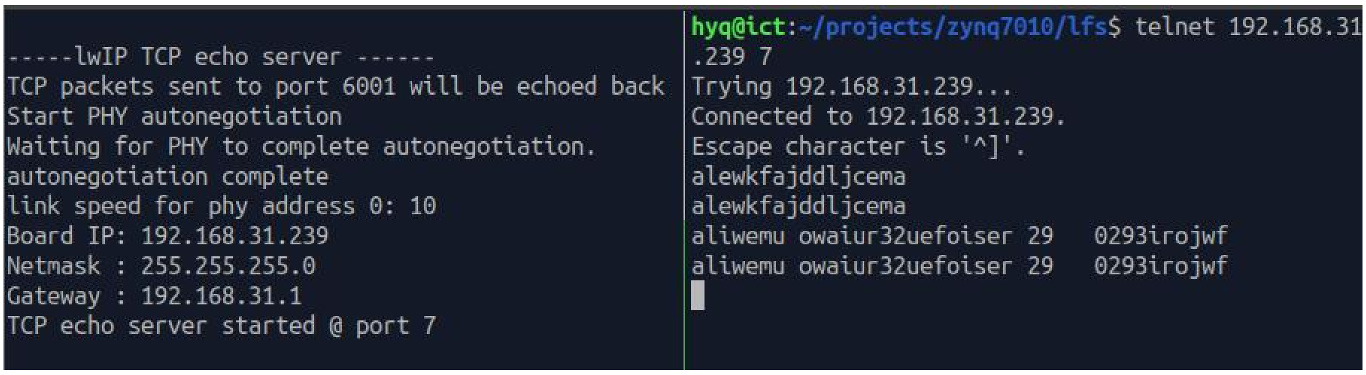


```
Hello World
```

sdk

Test network port

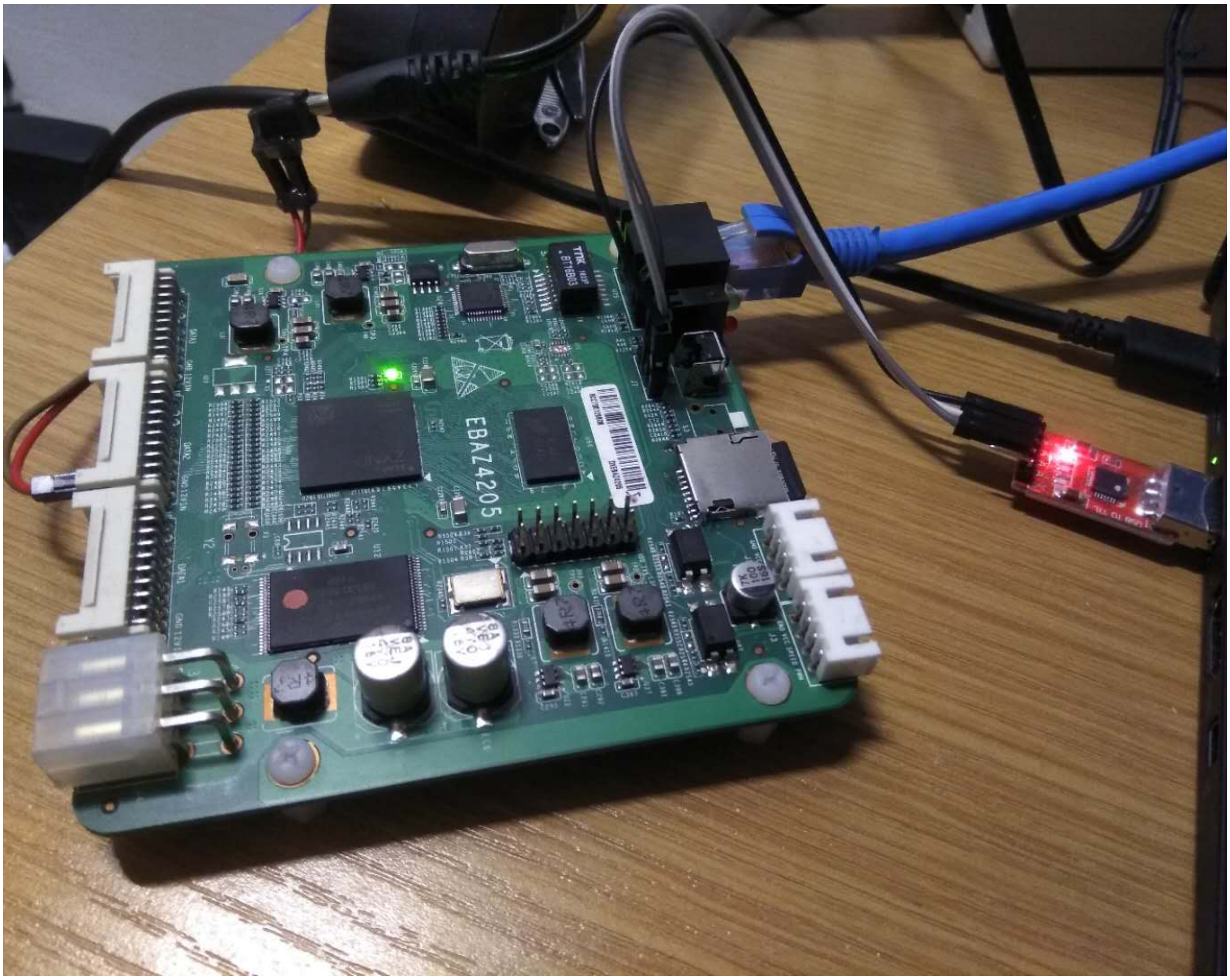
Create a new `lwIP Echo Server` project in the SDK , wait for it to compile, generate BOOT.bin, and copy it to the SD card. Plug in the network cable, `telnet` enter its No. 7 port in the terminal , enter a line, press Enter, and then it will return what you entered. . .



```
-----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
link speed for phy address 0: 10
Board IP: 192.168.31.239
Netmask : 255.255.255.0
Gateway : 192.168.31.1
TCP echo server started @ port 7
```

```
hyq@ict:~/projects/zynq7010/lfs$ telnet 192.168.31
.239 7
Trying 192.168.31.239...
Connected to 192.168.31.239.
Escape character is '^]'.
alewkfajddljcema
alewkfajddljcema
aliwemu owaiur32uefoiser 29    0293irojwf
aliwemu owaiur32uefoiser 29    0293irojwf
```

sdk



debugging

debugging