
Template

发布 1

Hongyi Wu(吴鸿毅)

2019 年 10 月 17 日

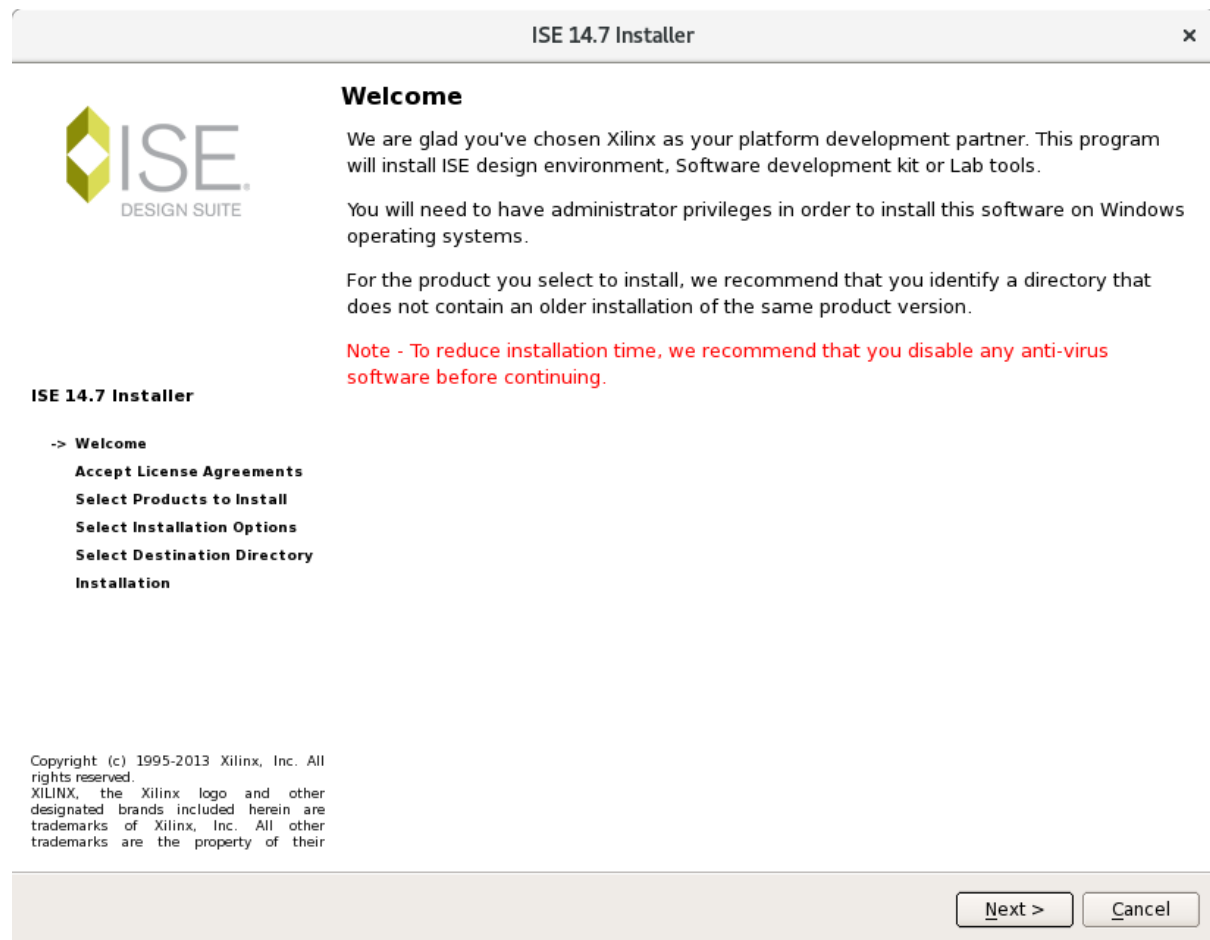
Contents:

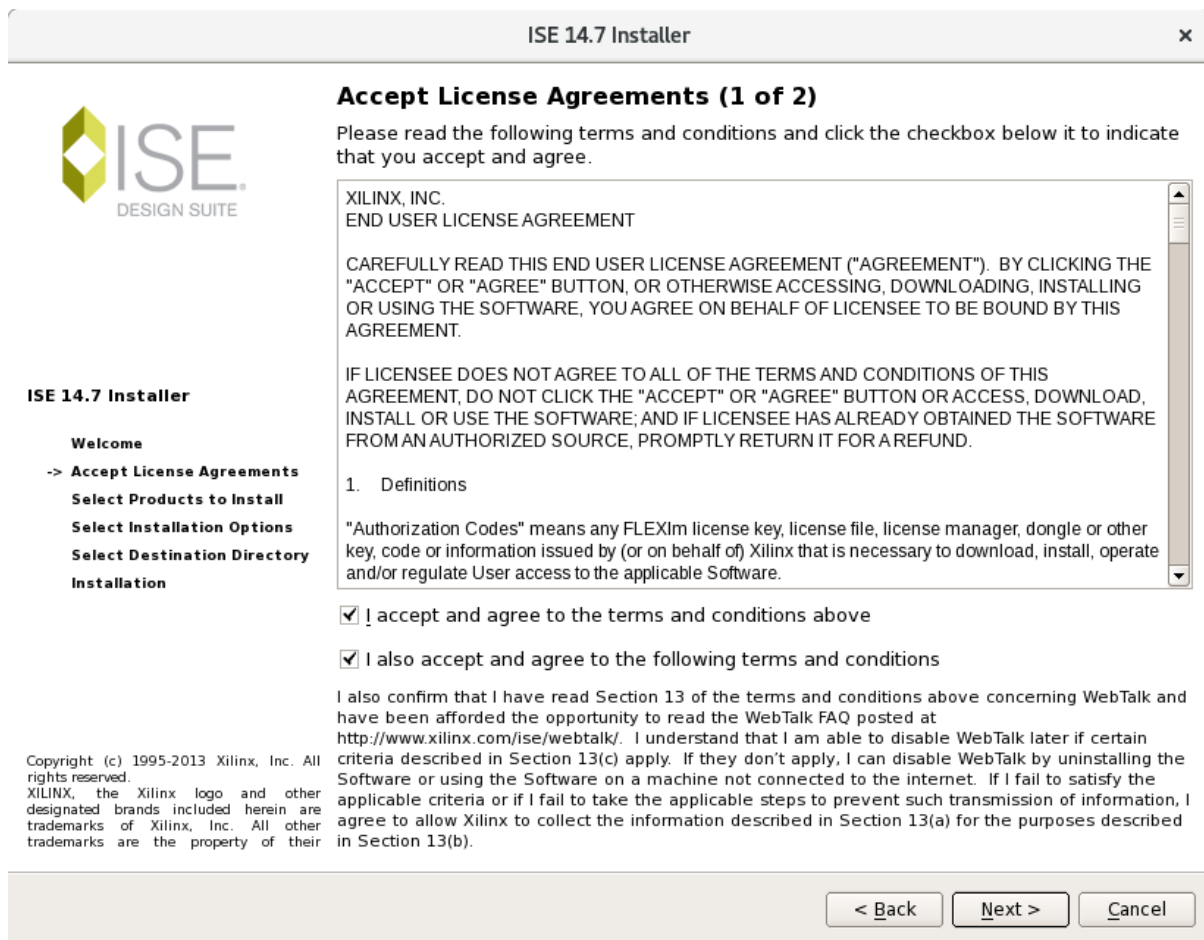
1	ISE/Altera/Vivado 软件	1
1.1	软件安装	1
2	硬件介绍	27
3	计数器	29
3.1	计数器	29
4	状态机	35
4.1	状态机	35
5	FIFO	37
5.1	FIFO	37
6	经验总结	39
6.1	经验总结	39
7	Indices and tables	41

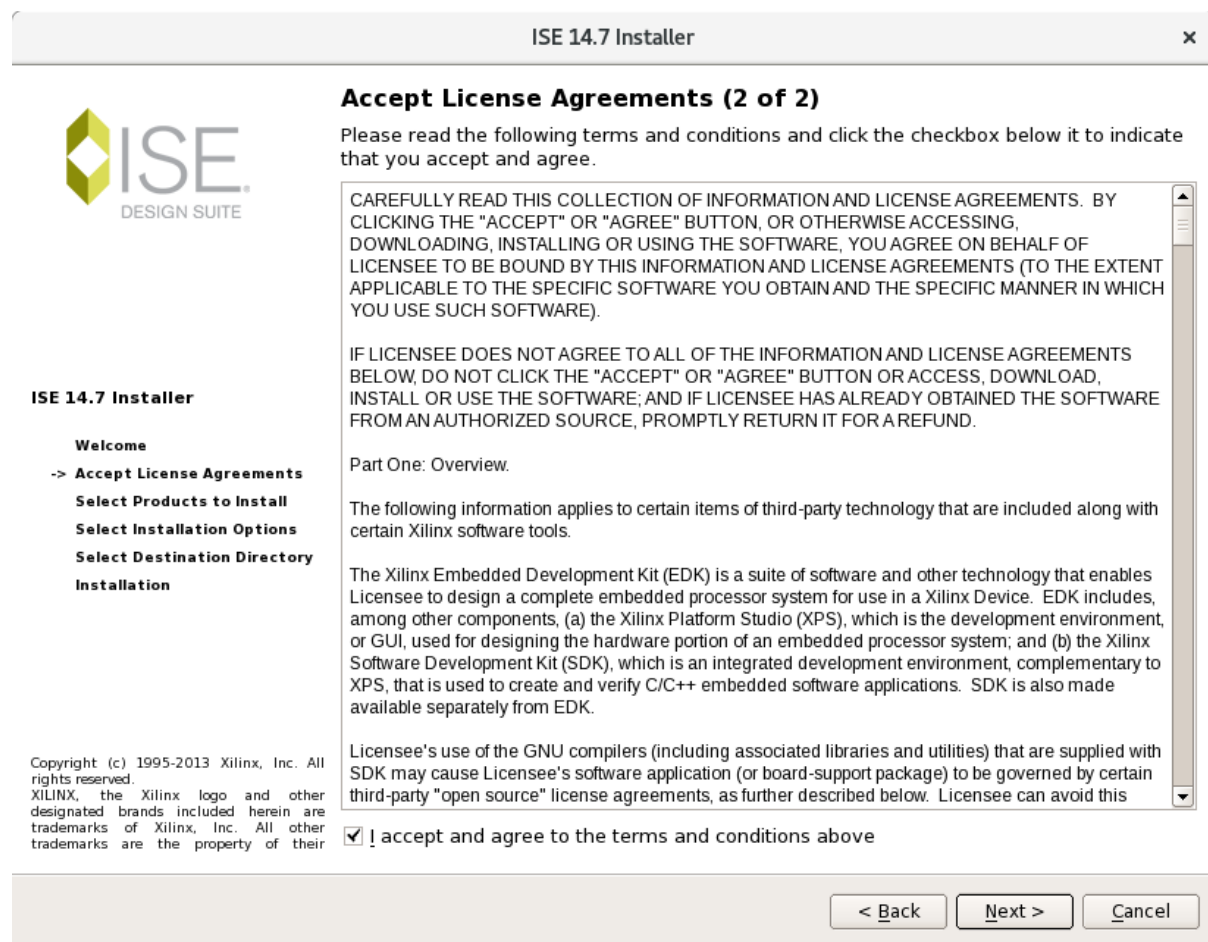
1.1 软件安装

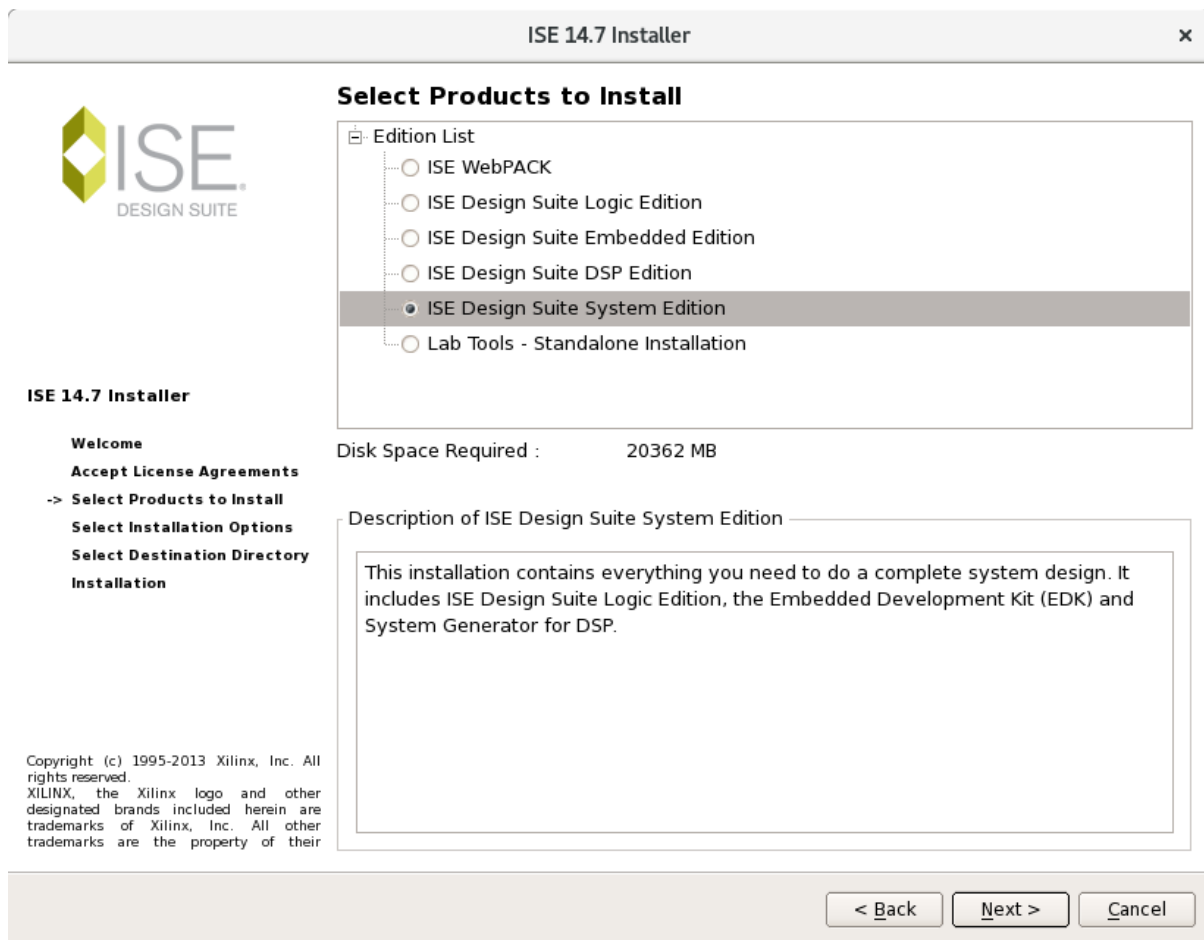
1.1.1 ISE 安装

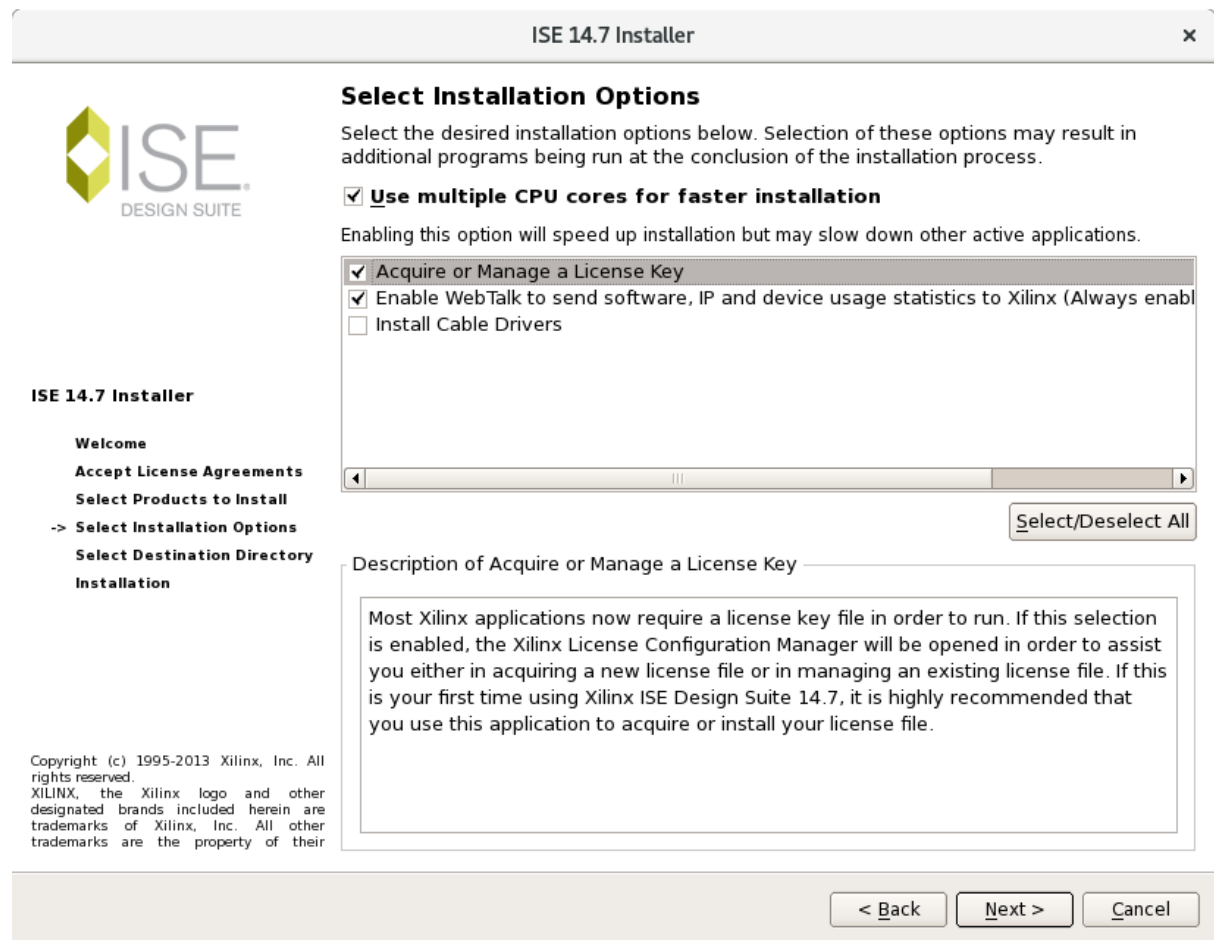
```
tar -xvf Xilinx_ISE_DS_Lin_14.7_1015_1.tar
cd Xilinx_ISE_DS_Lin_14.7_1015_1
./xsetup
```












ISE 14.7 Installer



Select Destination Directory

Select the directory where you want the software installed.

/home/wuhongyi/Xilinx

Browse...

Install location(s) :
/home/wuhongyi/Xilinx/14.7/ISE_DS

Disk Space Required : 20362 MB

Disk Space Available : 371143 MB

ISE 14.7 Installer

Welcome

Accept License Agreements

Select Products to Install

Select Installation Options

-> Select Destination Directory

Installation

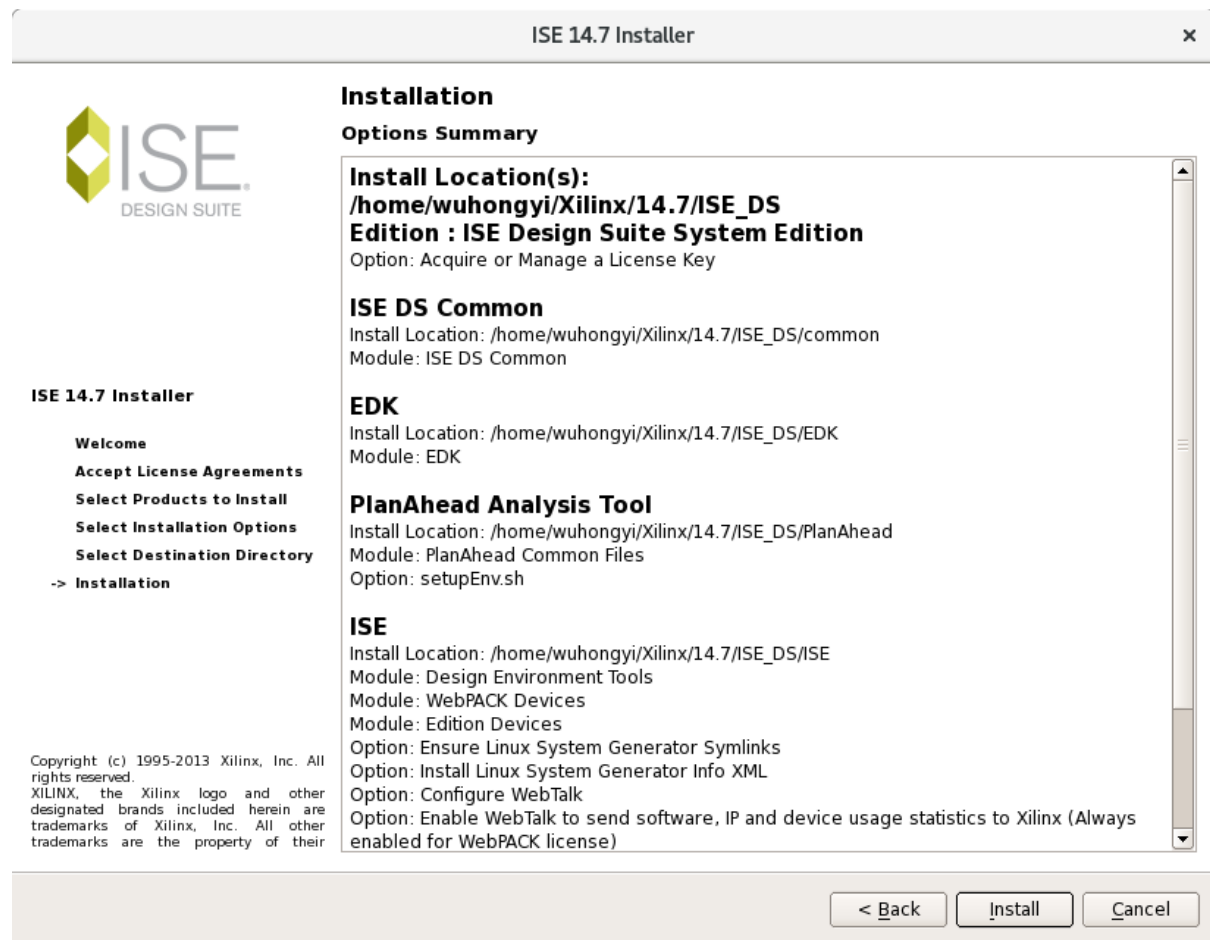
☒ Import tool preferences from previous version

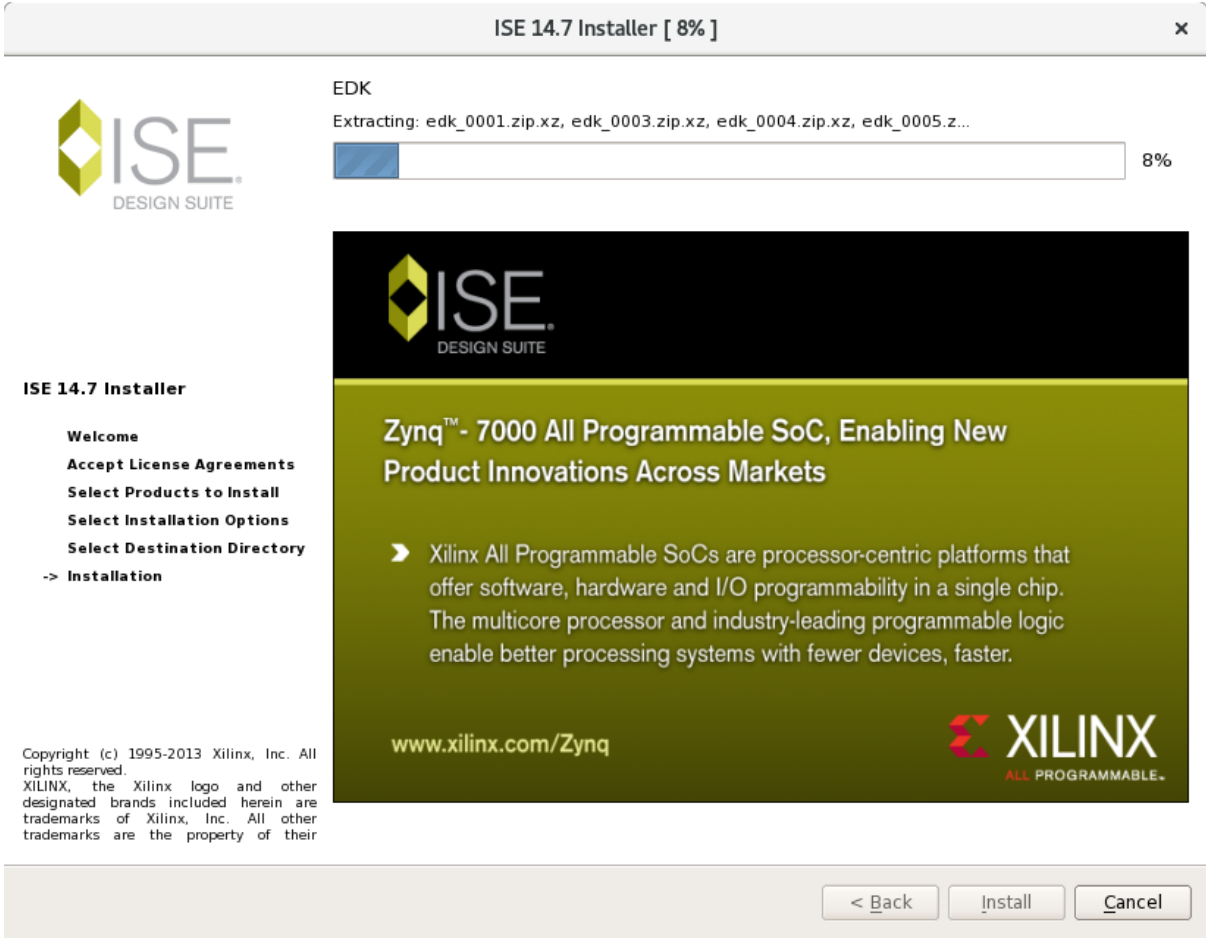
Copyright (c) 1995-2013 Xilinx, Inc. All rights reserved.
XILINX, the Xilinx logo and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their

< Back

Next >

Cancel



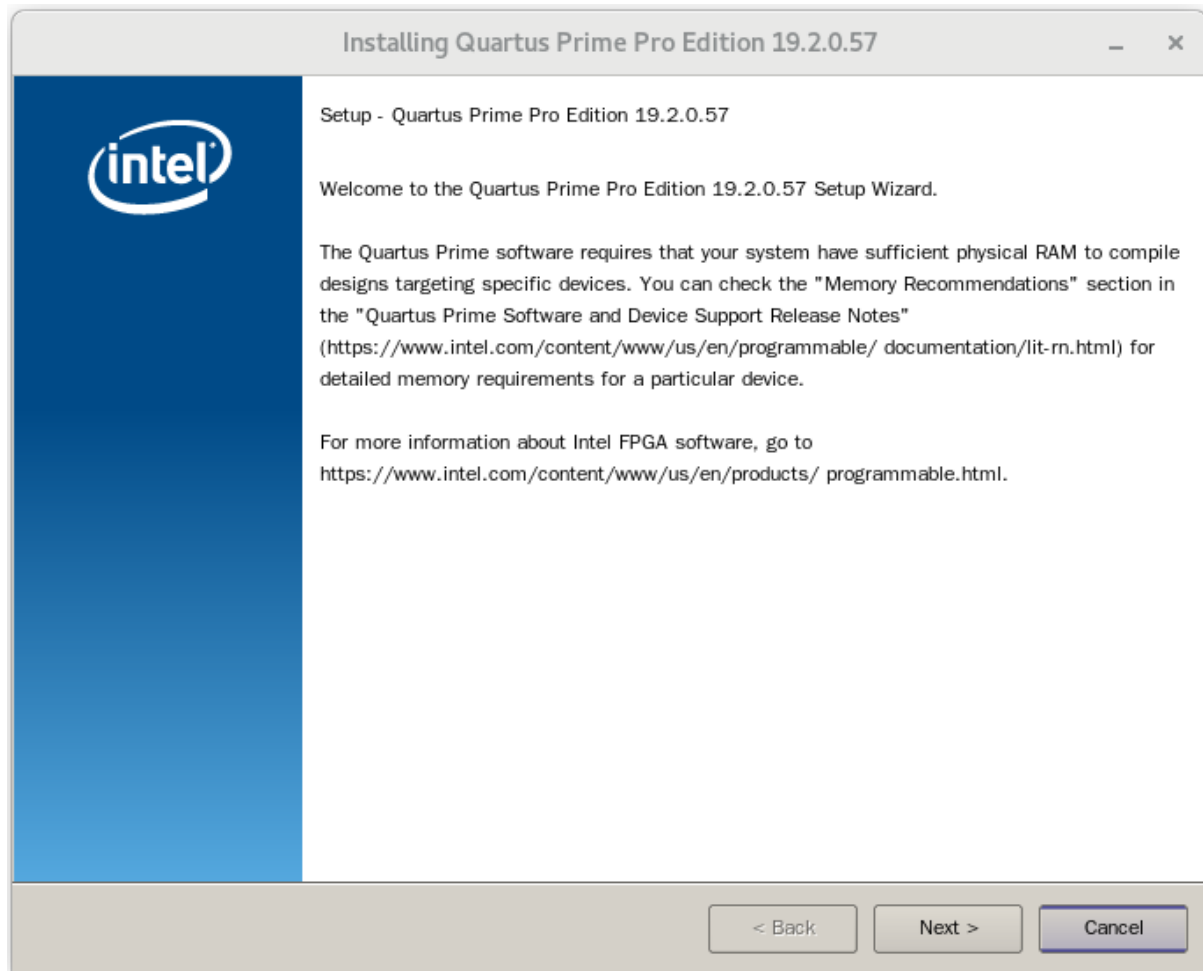


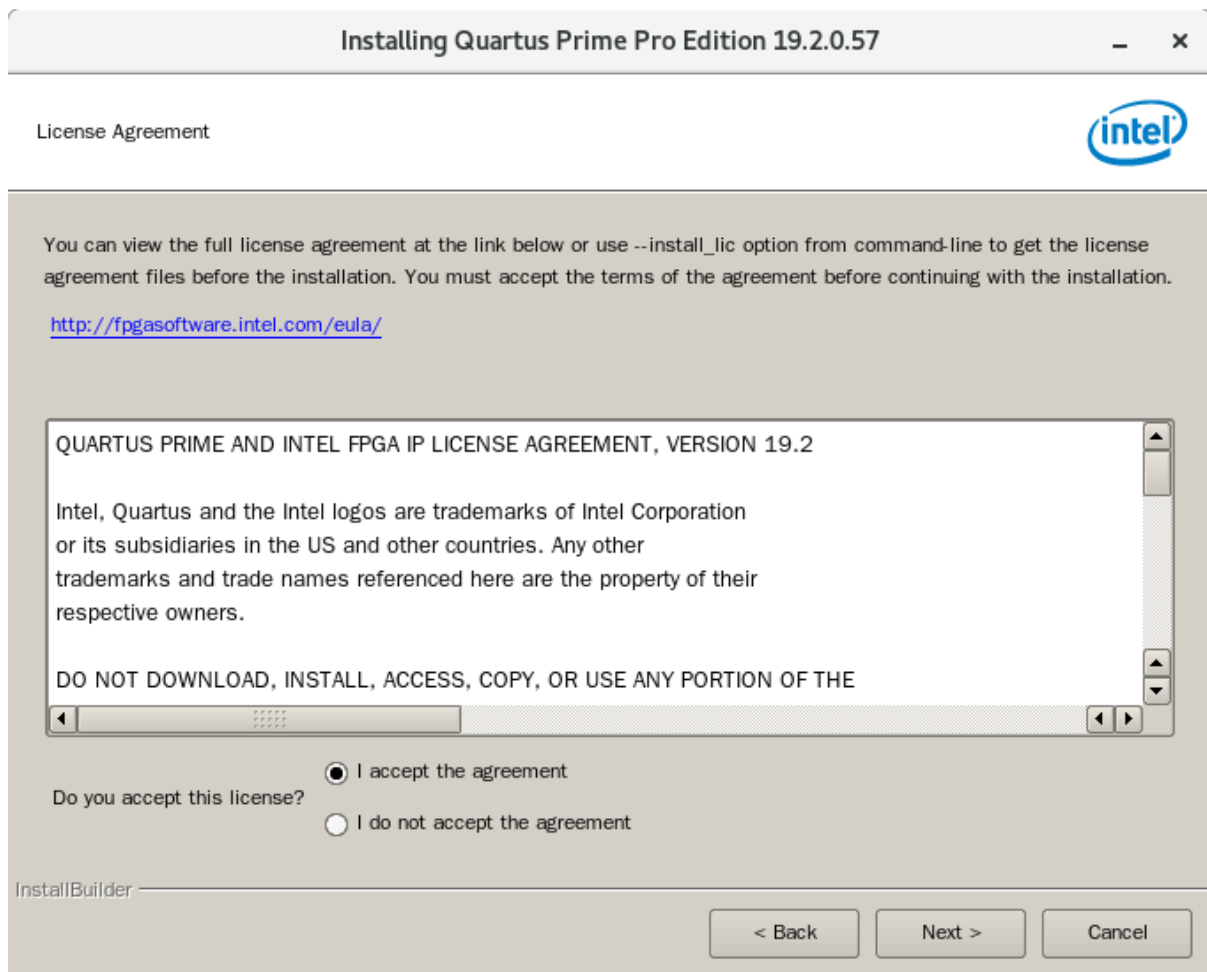
1.1.2 Altera 安装

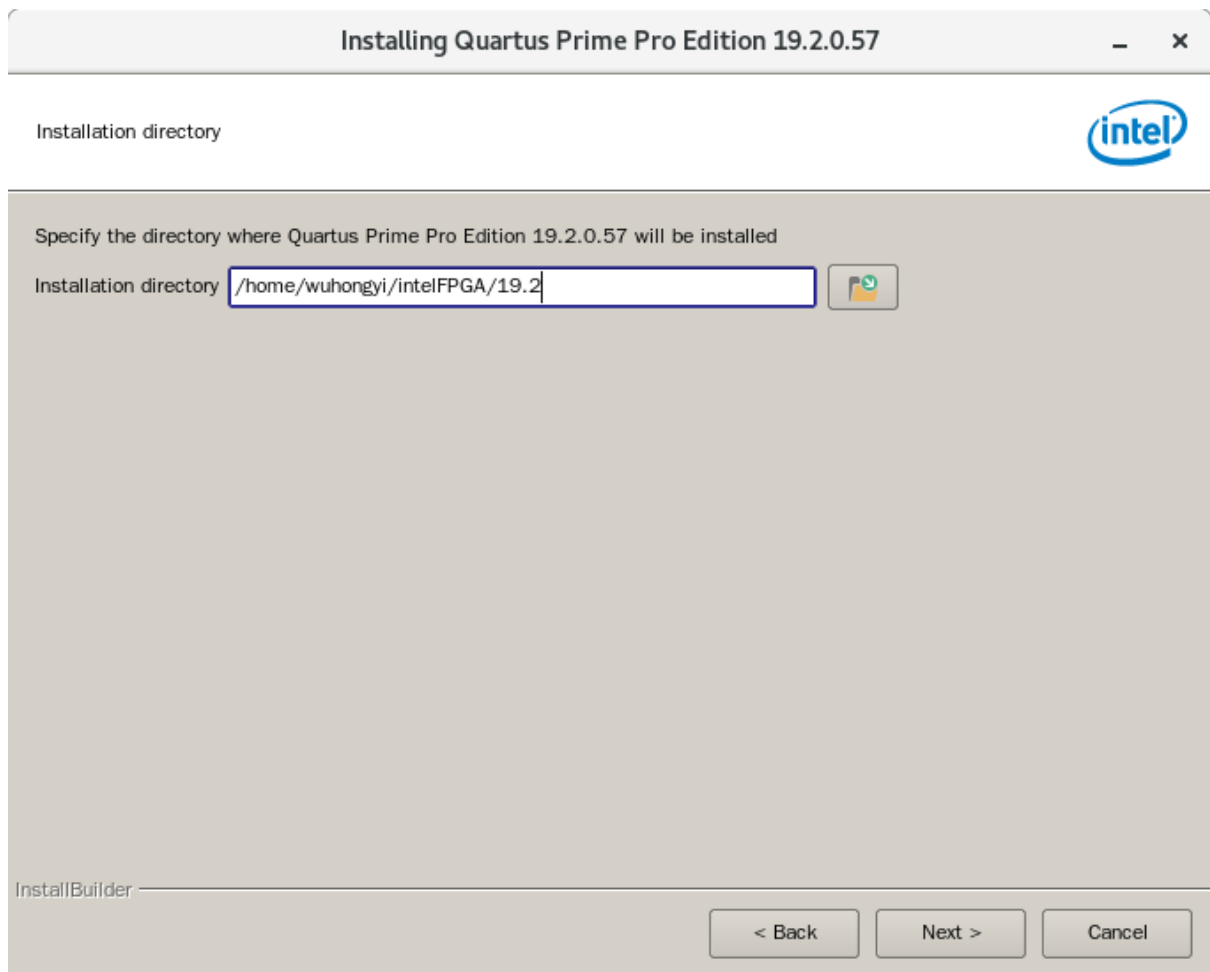
下载需要的所有文件，放在一个目录下

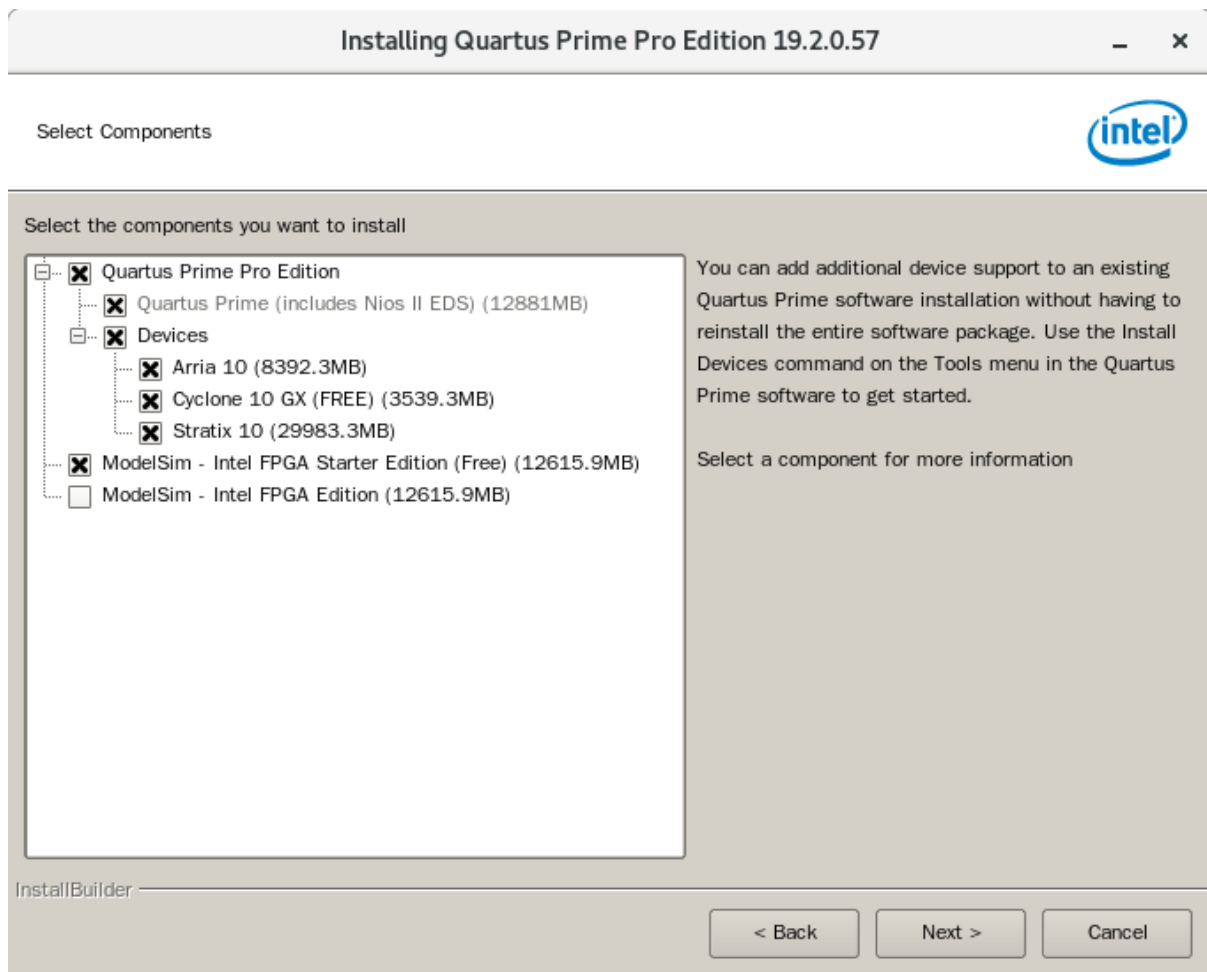
```
arria10-19.2.0.57.qdz      ModelSimProSetup-19.2.0.57-linux.run
cyclone10gx-19.2.0.57.qdz  QuartusProSetup-19.2.0.57-linux.run
modelsim-part2-19.2.0.57-linux.qdz  stratix10-19.2.0.57.qdz
```

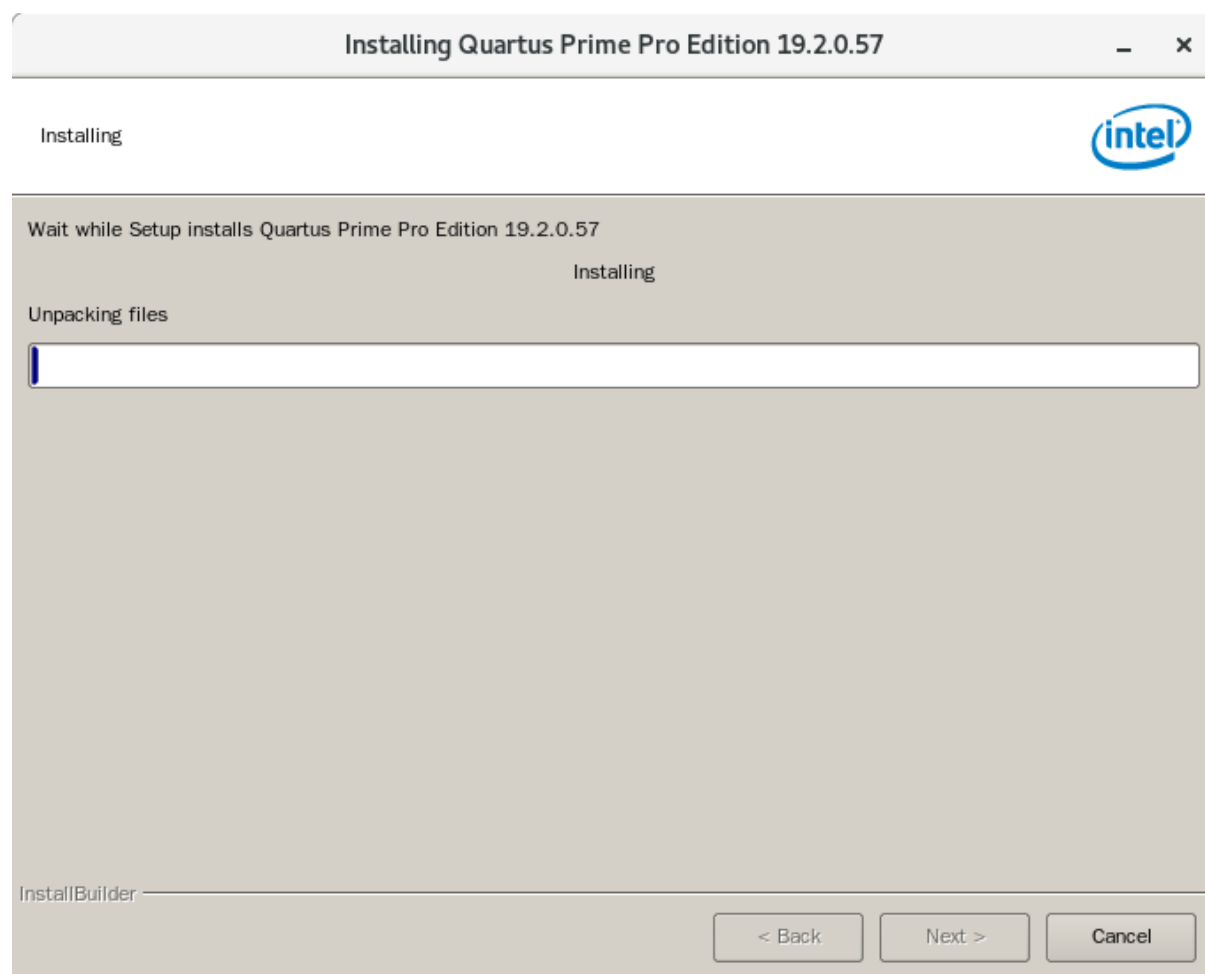
```
chmod +x QuartusProSetup-19.2.0.57-linux.run
./QuartusProSetup-19.2.0.57-linux.run
```

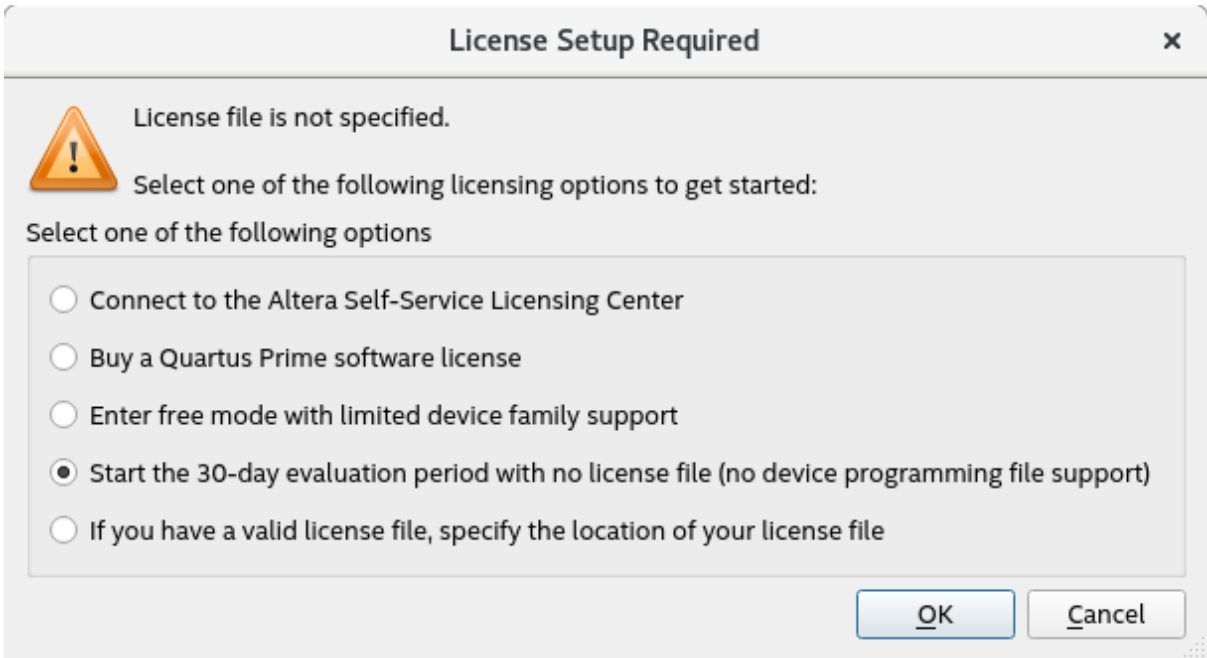
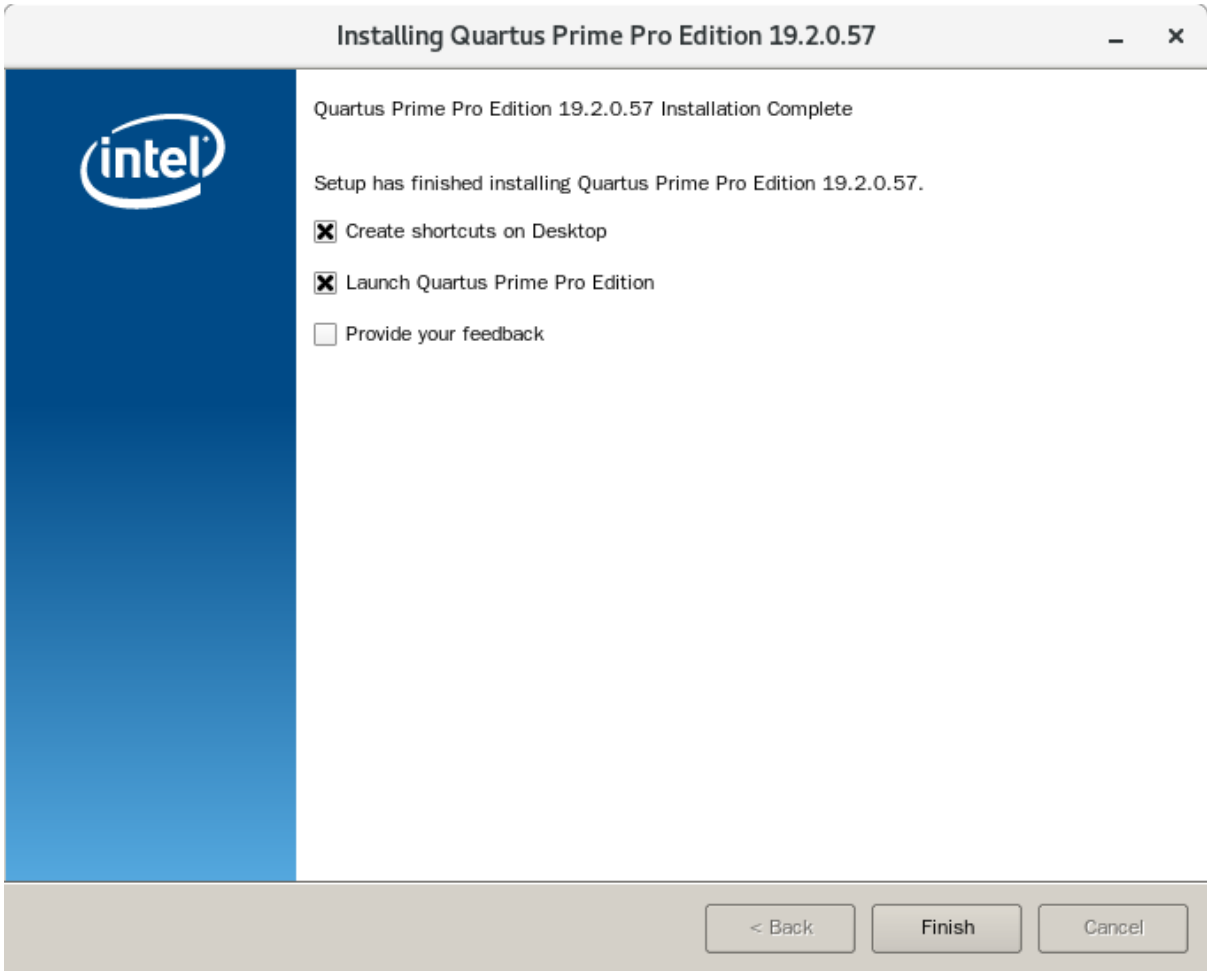












在安装路径下有以下文件

devdata	licenses	modelsim_ase	qsys	syscon
ip	logs	nios2eds	quartus	uninstall

quartus/bin 文件夹内存放 quartus 启动的脚本

```
./quartus
```

modelsim_ase/bin 文件夹内存放 modelsim 启动的脚本

```
./vsim
```

linux usb blaster 权限的设置

对于错误 error (209053): unexpected error in jtag server – error code 89, 它产生的原因在于, 在 linux 系统下, Quartus ii 的驱动 USB-Blaster 只能有 root 用户使用, 而普通用户是无权使用的。解决思路是更改 USB-Blaster 的使用权限, 使得普通用户也能使用。

因为 usb 默认只有 root 才有权限访问, 所以只要把权限修改一下即可, usb blaster 链接上电脑

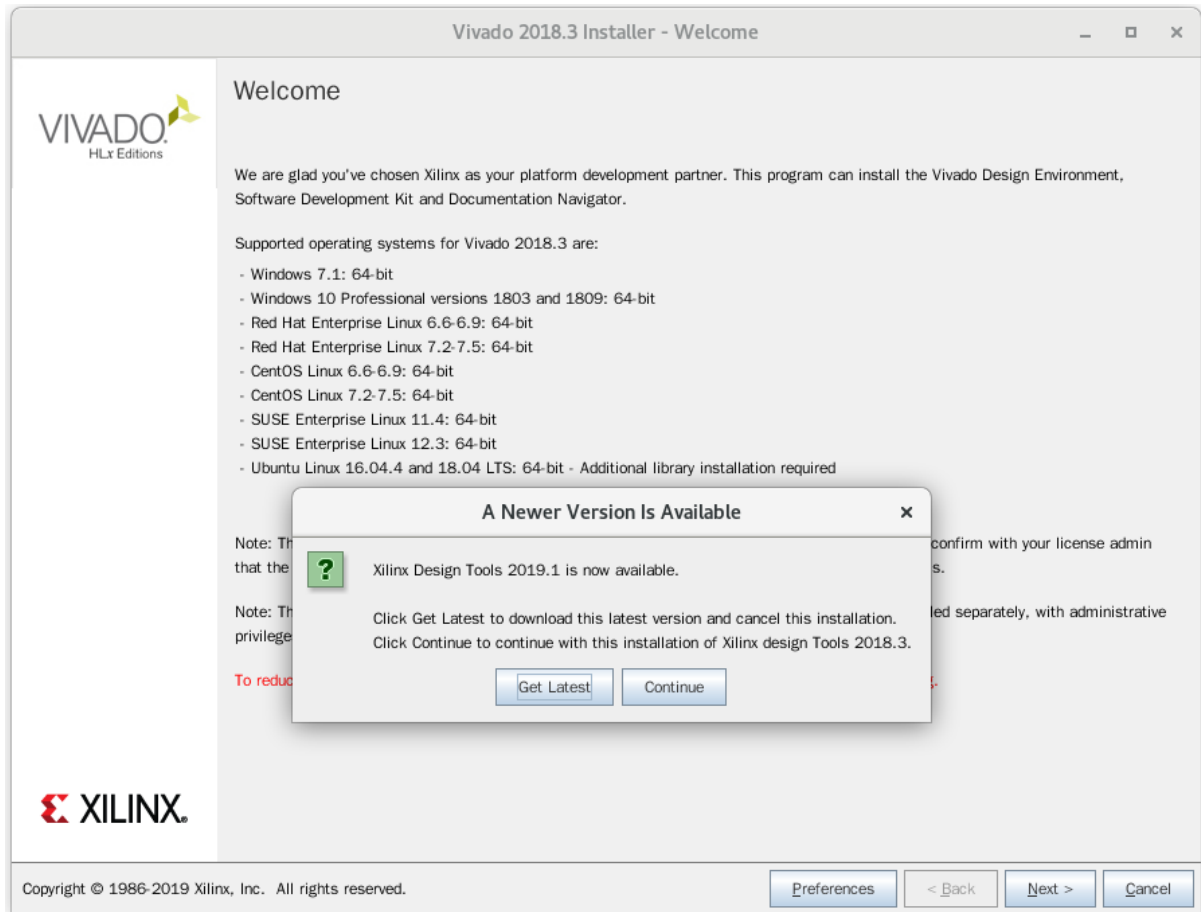
```
[root@localhost 003]# lsusb
Bus 002 Device 002: ID 8087:8000 Intel Corp.
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 8087:8008 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 004: ID 0bda:0184 Realtek Semiconductor Corp. RTS5182 Card Reader
Bus 003 Device 013: ID 09fb:6001 Altera Blaster
Bus 003 Device 003: ID 046d:c077 Logitech, Inc. M105 Optical Mouse
Bus 003 Device 002: ID 413c:2107 Dell Computer Corp.
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

说明 /dev/bus/usb/003/013 这个文件现在就是我们的 Altera Blaster 设备

```
cd /dev/bus/usb/003
chmod 666 013
```

1.1.3 Vivado 安装

```
tar -zxvf Xilinx_Vivado_SDK_2018.3_1207_2324.tar.gz
cd Xilinx_Vivado_SDK_2018.3_1207_2324
./xsetup
```



点击 continue 选择不下载最新版本，然后点击 Next 进入下一步

Vivado 2018.3 Installer - Accept License Agreements

Accept License Agreements

Please read the following terms and conditions and indicate that you agree by checking the I Agree checkboxes.

Xilinx Inc. End User License Agreement

By checking "I AGREE" below, or OTHERWISE ACCESSING, DOWNLOADING, INSTALLING or USING THE SOFTWARE, YOU AGREE on behalf of licensee to be bound by the agreement, which can be viewed by [clicking here](#).

☒ I Agree

WebTalk Terms And Conditions

By checking "I AGREE" below, I also confirm that I have read [Section 13 of the terms and conditions](#) above concerning WebTalk and have been afforded the opportunity to read the WebTalk FAQ posted at <https://www.xilinx.com/products/design-tools/webtalk.html>. I understand that I am able to disable WebTalk later if certain criteria described in Section 13(c) apply. If they don't apply, I can disable WebTalk by uninstalling the Software or using the Software on a machine not connected to the internet. If I fail to satisfy the applicable criteria or if I fail to take the applicable steps to prevent such transmission of information, I agree to allow Xilinx to collect the information described in Section 13(a) for the purposes described in Section 13(b).

☒ I Agree

Third Party Software End User License Agreement

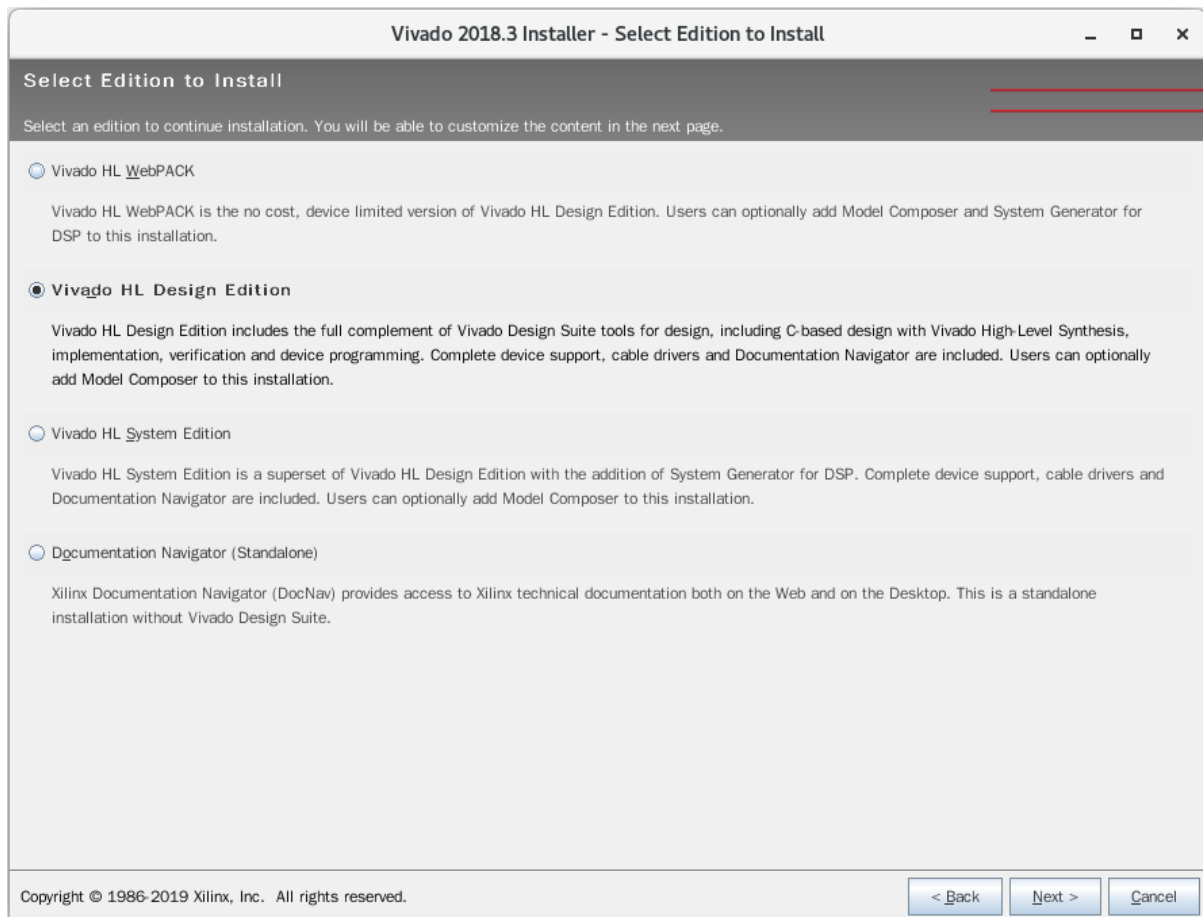
By checking "I AGREE" below, or OTHERWISE ACCESSING, DOWNLOADING, INSTALLING or USING THE SOFTWARE, YOU AGREE on behalf of licensee to be bound by the agreement, which can be viewed by [clicking here](#).

☒ I Agree

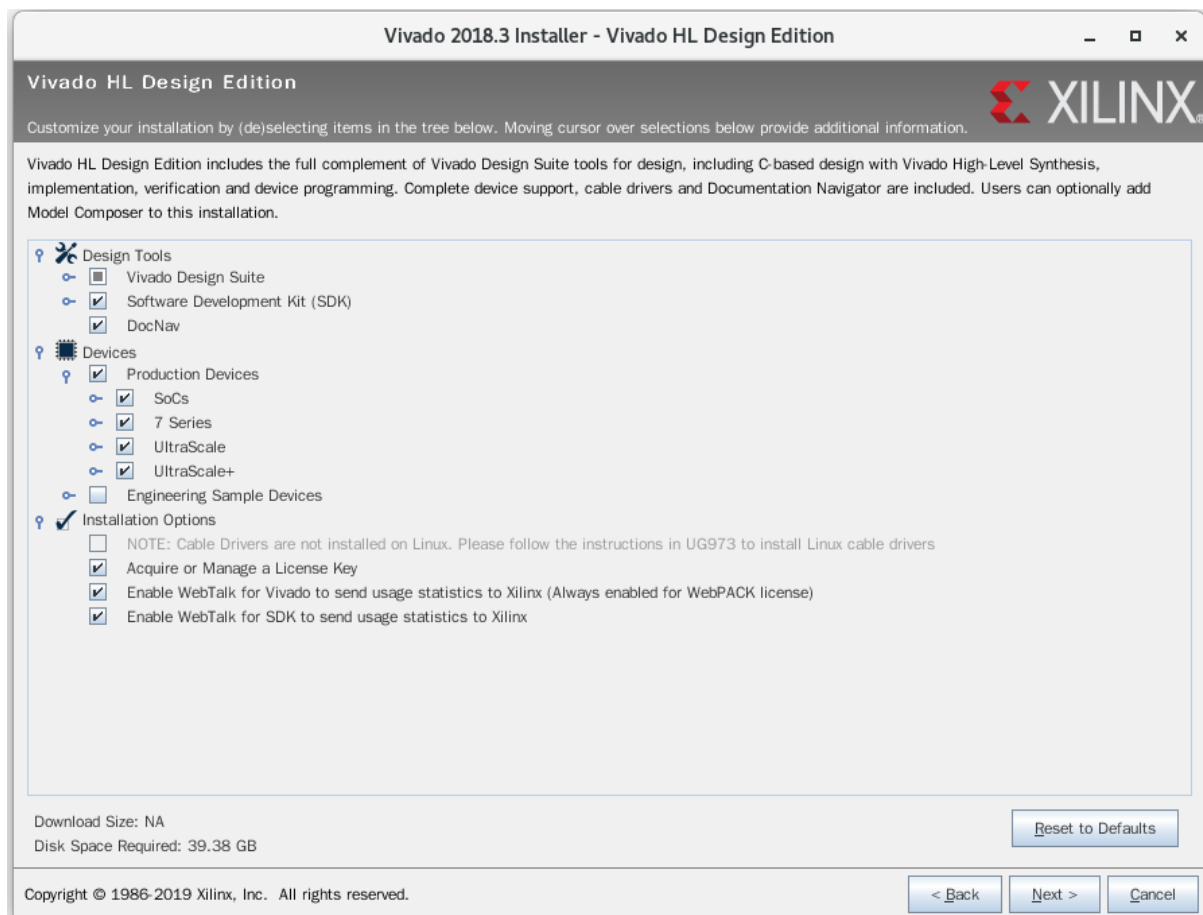
Copyright © 1986-2019 Xilinx, Inc. All rights reserved.

< Back Next > Cancel

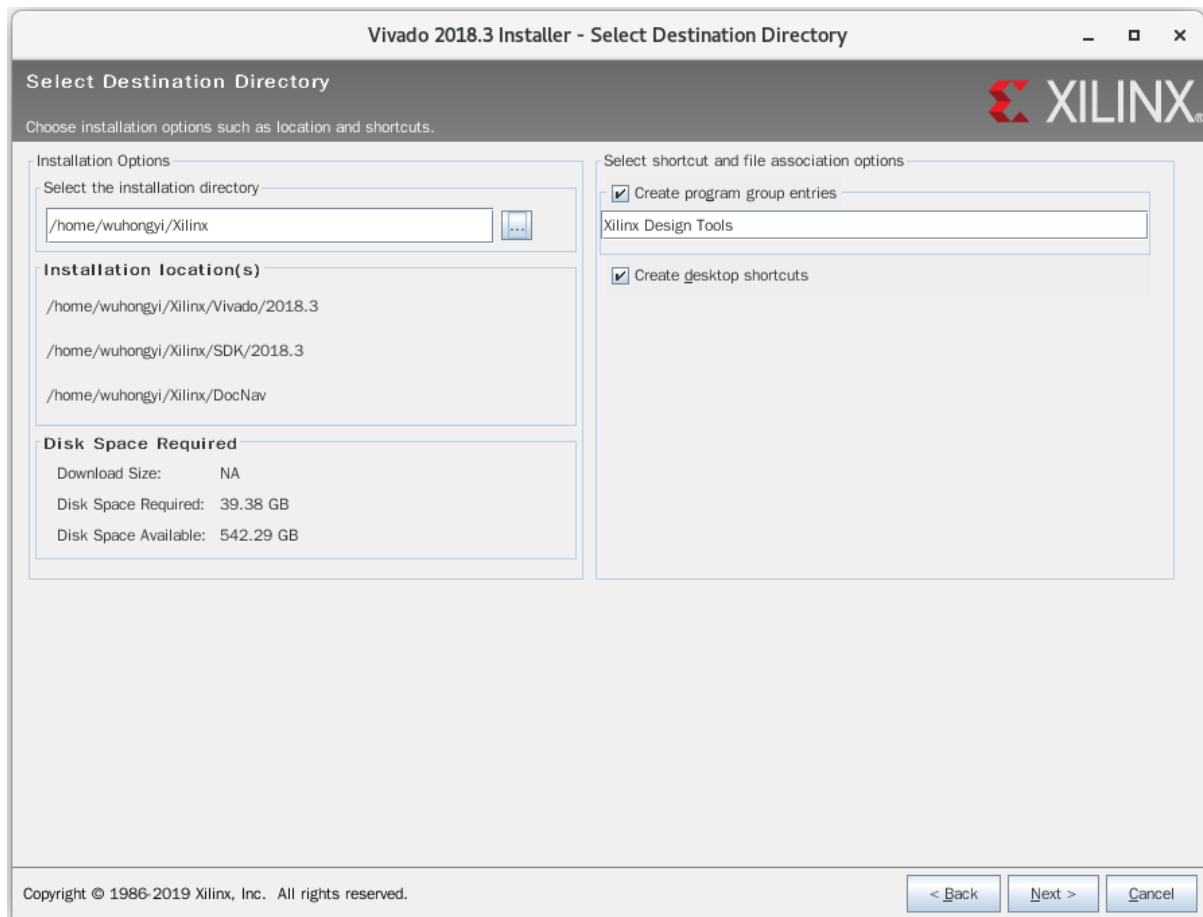
点击三个可选框，然后点击 Next 进入下一步



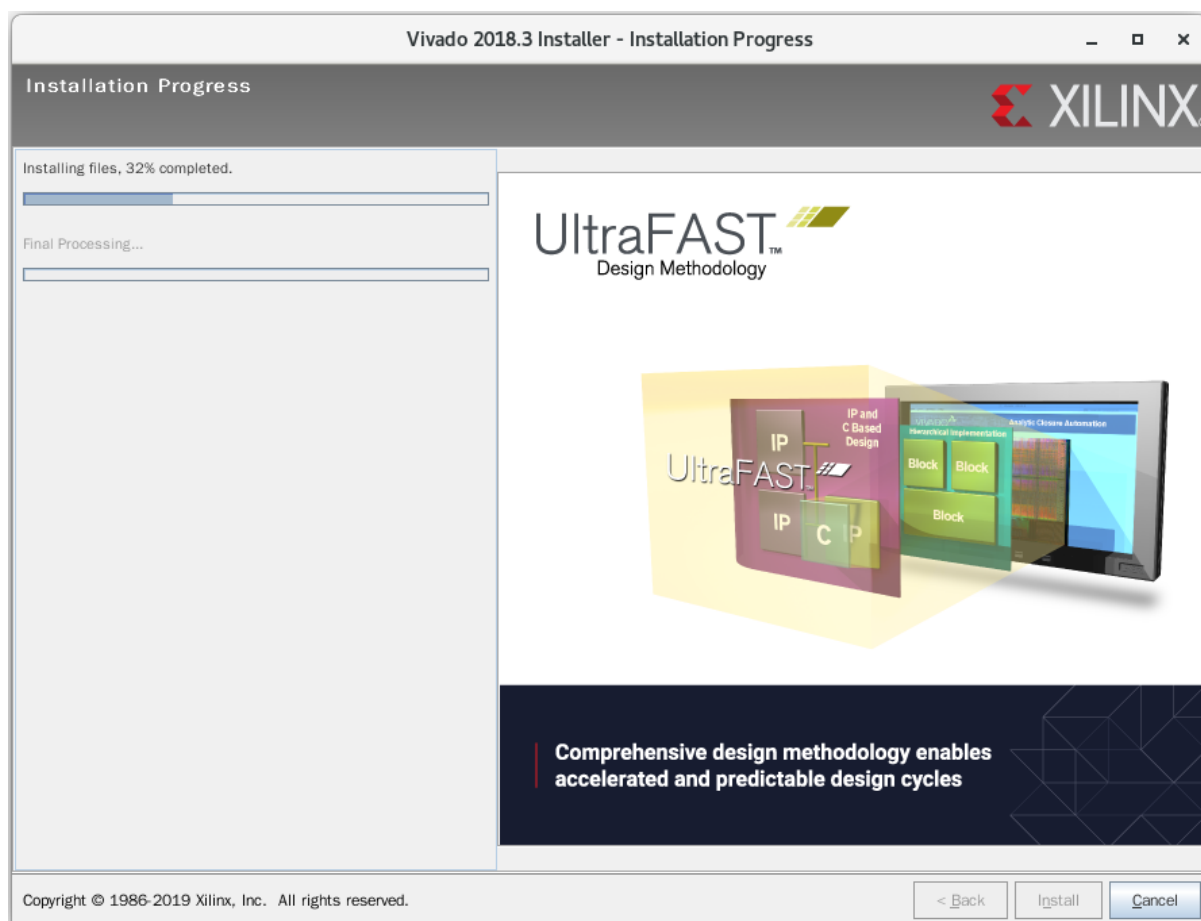
选择 Vivado HL Design Edition，然后点击 Next 进入下一步



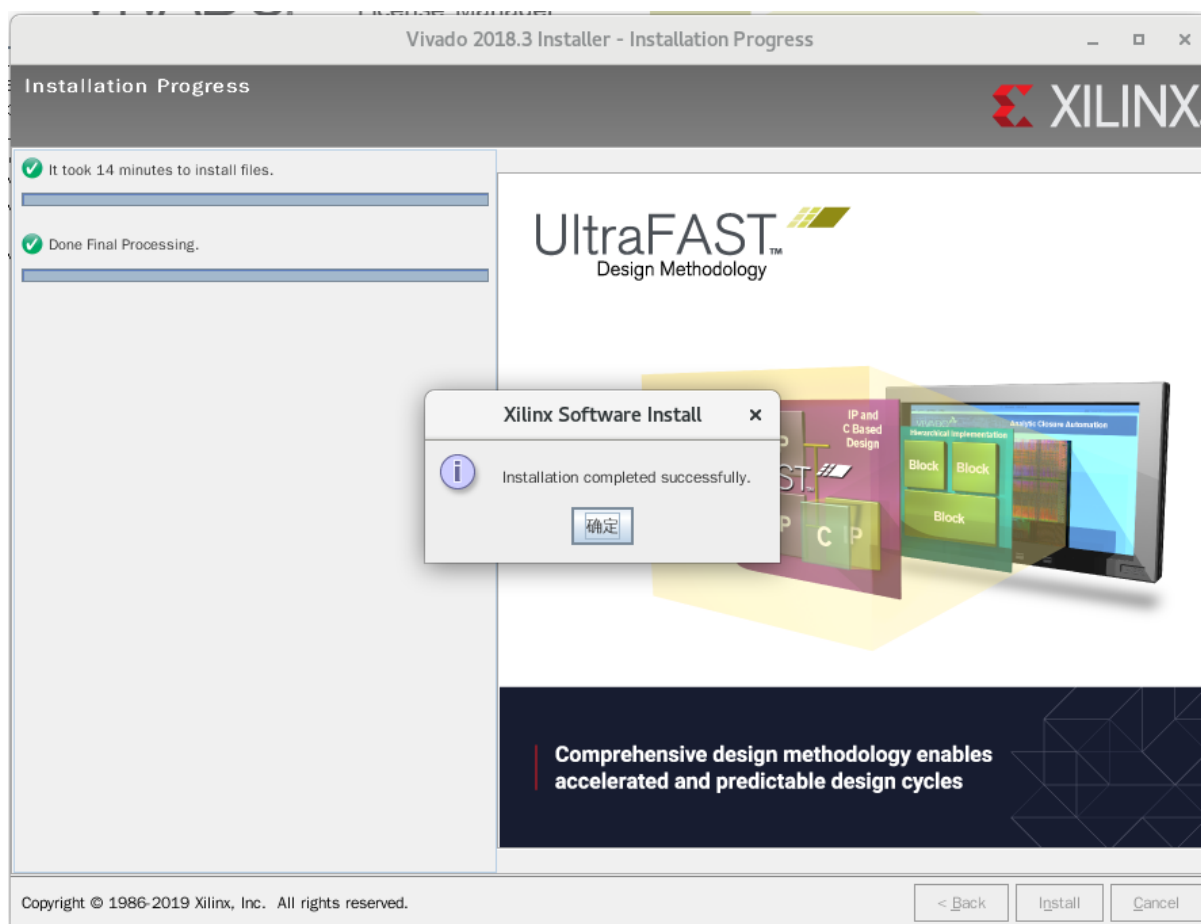
直接点击 Next 进入下一步



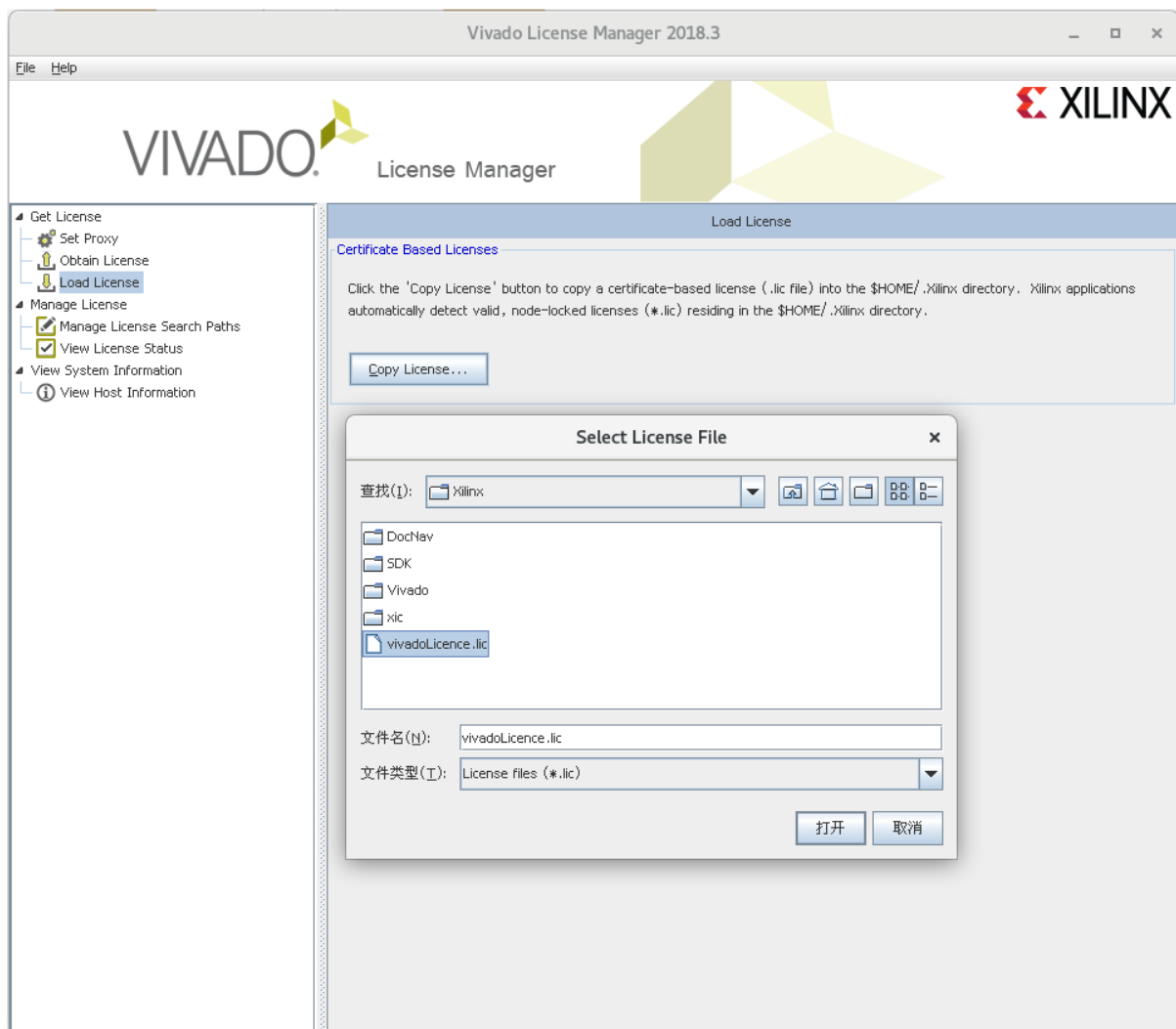
选择安装目录，这里我选择安装到 /home/wuhongyi/Xilinx，然后点击 Next 进入下一步



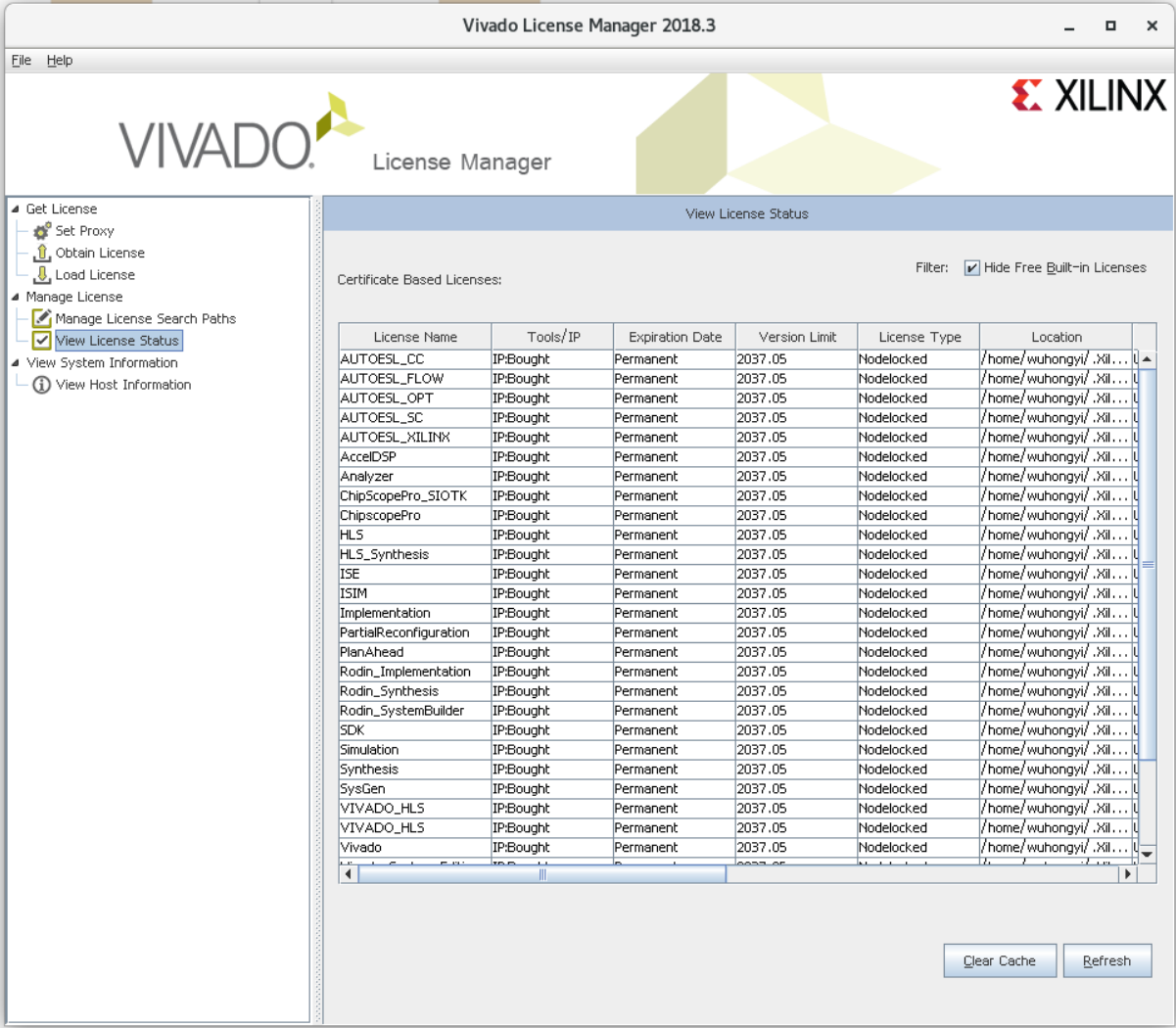
等待安装完成



将 vivadoLicence.lic 文件复制到安装目录，这里为 /home/wuhongyi/Xilinx
安装完成之后会弹出以下界面



点击左上方的 Load License，选择我们的 vivadoLicence.lic 文件
然后点击左上方的 View License Status 可查看破解的 IP 核



CHAPTER 2

硬件介绍

3.1 计数器

3.1.1 计数器规则

计数器规则 1: 计数器逐一考虑三要素-初值、加 1 条件、结束条件

任何计数器都有三个要素：初值、加 1 条件、结束值

- 初值：计数器的默认值或者开始计数的值
- 加 1 条件：计数器执行加 1 条件
- 结束值：计数器计数周期的最后一个值设计计数器，要逐一考虑这三个要素，一般是先考虑初值，再考虑加 1 条件，最后再考虑结束值。

计数器规则 2: 计数初值必须为 0

计数器的默认值和开始值一定要为 0。这是我们规范的统一要求。我们知道一般编程语言计数都是从 0 开始的，0 表示第 1 个，1 表示第 2 个，。。。这里我们也参考这种做法，计数器都从 0 开始计数。

所有计数器都统一从 0 开始计数，有助于我们阅读理解、方便使用，从而不用从头看具体代码，就能清楚这个数值的含义。

计数器规则 3: 使用某一计数值，必须同时满足加 1 条件

计数器从 0 开始计数，计数器的默认值，也就是复位值也是 0。当计数器值为 0 时，如何判断这是计数器的第一个值还是还没开始计数的默认值呢？

可以通过加 1 条件来判断。当加 1 条件无效时，计数器值为 0 表示未开始计数的默认值；当加 1 条件有效时，计数器值为 0 表示计数器的第 1 个值。以此类推，当 `cnt==x-1` 时，不表示数到 `x`；只有当 `cnt==x-1` 时，并且加 1 条件有效时，才表示数到 `x`。

例如：当加 1 条件为 `add_cnt`，且 `add_cnt && cnt==4` 时，表示计数到 5 个；而当 `add_cnt==0 && cnt==4` 时，不表示计数到 5 个。

计数器规则 4：结束条件必须同时满足加 1 条件，且结束值必须是 x-1 的形式。

计数器的结束条件必须同时满足加 1 条件。例如假设要计数 5 个，那么结束值是 4，但是结束条件不是 `cnt==4` 而是 `add_cnt && cnt==4`。因为 `cnt==4` 不表示计数到 5 个，只有 `add_cnt && cnt==4` 时，才表示计数到 5 个。

为了更好地阅读代码，我们这里规定结束值必须是 x-1 的形式，即 `add_cnt && cnt==4` 要写成 `add_cnt && cnt==5-1`。这里的“5”表示希望计算的个数，“-1”则是固定格式。有了这个约定后，计数的边界就明确了。

计数器规则 5：当取某个数时，assign 形式必须为：（加 1 条件）&&（cnt== 计数值-1）

当要从计数器取某个数时，例如要取计数器的第 5 个点，就很容易写成 `cnt==5-1`，这是不正确的。正确的写法时：（加 1 条件）&&（cnt== 计数值-1），如 `add_cnt && cnt==5-1`。

计数器规则 6：结束后必须回到 0

每轮计数周期结束后，计数器变回 0，这是为了使计数器能够循环重复计数。

计数器规则 7：若需要限定范围，则推荐使用“>=”和“<”两种符号

设计时，考虑边界值通常要花费一些心思，而且容易出错。为此，我们约定：若需要限定范围，则推荐使用“>=”和“<”两种符号。例如要取前 8 个数，那么就取 `cnt>=0 && cnt<8`。注意，一定是“大于或等于”和“小于”符号，而不使用“大于”和“小于或等于”符号。

该规则参考编程里的 for 循环语句。假如要循环 8 次，for 循环的条件通常会写成“`i==0; i<8; i++`”，前面的 0 表示开始值，后面的 8 表示循环次数。当然，也可以写成“`i==0; i<=7; i++`”，但是这数字的意义就实在令人费解了，虽然知道 7 是从 8-1 得来的，但多一个“-1”的思考，就纯属画蛇添足了。

计数器规则 8：设计步骤

设计步骤：先写计数器的 always 段，条件用名字代替；然后用 assign 写出加 1 条件；最后用 assign 写出结束条件

我们的计数器模版代码包括三段。

第一段，写出计数器的 process/always 段

VHDL

```
process(clk,rst_n)
begin
    if(rst_n = '0')then
        cnt <= 0;
    elsif(clk'event and clk = '1')then
        if(加 1 条件)then
            if(结束条件)then
                cnt <= 0;
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end if;
end process;
```

verilog

```

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        cnt <= 0;
    end
    else if(加 1 条件) begin
        if(结束条件)
            cnt <= 0;
        else
            cnt <= cnt + 1;
        end
    end
end

```

大家有没有发现上述模版的特点？这个模版只需要填两项内容：加一条件和结束条件。如果为加 1 条件和结束条件定义一个信号名，例如 add_cnt 和 end_cnt，则代码变成：

VHDL

```

process(clk,rst_n)
begin
    if(rst_n = '0')then
        cnt <= 0;
    elsif(clk'event and clk = '1')then
        if(add_cnt )then
            if(end_cnt)then
                cnt <= 0;
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end if;
end process;

```

verilog

```

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        cnt <= 0;
    end
    else if(add_cnt) begin
        if(end_cnt)
            cnt <= 0;
        else
            cnt <= cnt + 1;
        end
    end
end

```

第二段，用组合逻辑写出加 1 条件

在此阶段，只需要想好一个点，就是计数器的加 1 条件。假设计数器的加 1 条件为 a==2，则代码如下：

VHDL

```
add_cnt <=  a=2;--add 1
```

verilog

```
assign add_cnt = a==2;//add 1
```

第三段，用组合逻辑写出结束条件

在此阶段，只需要想好一个点，就是计数器的结束值。参考计数器规则 5 的要求，结束条件的形式一定是：（加 1 条件）&&（cnt== 计数值-1）。假设计数器要计数 10 个，则代码如下：

VHDL

```
end_cnt <= add_cnt and (cnt == 10-1);--end
```

verilog

```
assign end_cnt = add_cnt && cnt == 10-1; //end
```

至此，就完成了计数器代码的设计。总结一下这段代码的特点：每次值考虑一件事，按这要求去做，就非常容易完成代码设计。

以下是我们完整的模版：

VHDL

```
signal cnt : integer range 0 to ;--max number
signal add_cnt : boolean;
signal end_cnt : boolean;

process(clk,rst_n)
begin
    if(rst_n = '0')then
        cnt <= 0;
    elsif(clk'event and clk = '1')then
        if(add_cnt)then
            if(end_cnt)then
                cnt <= 0;
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end if;
end process;

add_cnt <= ;--add 1 dout_tmp='1'
end_cnt <= add_cnt and (cnt == -1);--end
```

以上模版中，只需要补充三个地方，

```
signal cnt : integer range 0 to ;--在 to 后面补充计数器的最大计数范围
add_cnt <= ;--补充加 1 条件
end_cnt <= add_cnt and (cnt == -1);--补充计数器数多少数
```

verilog

```
reg [ :0] cnt ;
wire add_cnt;
wire end_cnt;

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        cnt <= 0;
    end
    else if(add_cnt) begin
        if(end_cnt)
            cnt <= 0;
        else
            cnt <= cnt + 1;
        end
    end
end

assign add_cnt = ;//condition: add 1
assign end_cnt = add_cnt && cnt == -1; //End condition, last value
```

以上模版中，只需要补充三个地方，

```
reg [ :0] cnt ;//补充计数器的最大计数范围
assign add_cnt = ;//补充加 1 条件
assign end_cnt = add_cnt && cnt == -1; //补充计数器数多少数
```

计数器规则 9：加 1 条件必须与计数器严格对齐，其它信号一律向计数器对齐

我们设计出计数器，但一般计数器不是最终的目的，最终的目的是输出各种信号。设计计数器是为了方便产生这些输出信号（包括中间信号），并能从计数器获取变化条件。例如：信号 dout 在计数到 6 时拉高，则其变 1 的条件是：add_cnt && cnt==6-1。

假设有两个信号：dout0 在计数到 6 时拉高；dout1 在计数到 7 时拉高。一种做法是 dout0 变 1 的条件是 add_cnt && cnt==6-1，dout1 变 1 的条件是 dout0==1。这个 dout1 就是间接与计数器对齐。这是非常不好的方法。这里我们建议一律向计数器对齐，dout1 变 1 的条件应该为 add_cnt && cnt==7-1。

计数器规则 10：命名必须符合规范

比如：add_cnt 表示加 1 条件；end_cnt 表示结束条件

如无特别说明，计数器的命名都要符合规范，加 1 条件的前缀为“add_”，结束条件的前缀为“end_”。

4.1 状态机

4.1.1 状态机规则

状态机规则 1：使用四段式写法

四段式不是指四个 `always` 代码，而是四段代码。另外需要注意的是，四段式状态机并非固定不变。如果没有输出信号就只有三段代码（两个 `always`）；如果有多个输出信号，那么就会有多个 `always`。

第一段，同步时序的 `always` 模块，格式化描述次态迁移到现态寄存器。

5.1 FIFO

5.1.1 FIFO 规则

FIFO 规则 1：使用 Show-ahead 都模式

根据 FIFO 的读模式，一般有两种使用模式：Show-ahead 和 Nornal 模式。这两种模式的区别在生成 FIFO IP 核额步骤中说明。

FIFO 规则 2：读、写隔离规则

读、写隔离规则是指：读控制和写控制是独立的，它们之间除了用 FIFO 交流信息外，不能有任何信息传递。因此，既不能根据 FIFO 的读状态或者读出的数据来决定写行为，也不能根据 FIFO 的写状态和写入的数据来决定读行为。

FIFO 规则 3：读使能必须判断空状态，并且用组合逻辑产生

rdreq 必须由组合逻辑产生，原因与 empty 有关。

FIFO 规则 4：处理报文时，把指示信号与数据一起存入 FIFO

FIFO 规则 5：读、写时钟不同时，必须用异步 FIFO

6.1 经验总结

6.1.1 计数器

- 画出输入、输出波形（根据功能要求、画出输入和输出的波形）
- 画出计数器结构
- 确定加 1 条件（计数器数什么，加 1 条件不足则加 `flag_add`）
- 确定计数器结束条件（数多少个，个数不同时，用变量法，即用 `x` 代替；`x` 不足以区分时则加 `flag_sel`）
- 如果有更新波形
- 其它信号变化点条件（其它信号：即输出或内部信号；变化点：0 变 1、1 变 0）
- 写出计数器代码
- 写出其它信号代码

6.1.2 状态机

- 明确功能
- 输出分析
- 状态合并
- 状态转移条件
- 转移条件
- 完整性检查
- 状态机代码
- 功能代码

6.1.3 FIFO

- FIFO 的写使能写数据，同时用组合逻辑或者同时用时序逻辑。
- FIFO 的读使能，用组合逻辑。
- 数据的输出用组合逻辑。

6.1.4 波形

- 时序逻辑的波形观看方法：时钟上升沿前看输入，时钟上升沿后看输出。
- 组合逻辑的波形观看方法：输入变输出即刻变。

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`