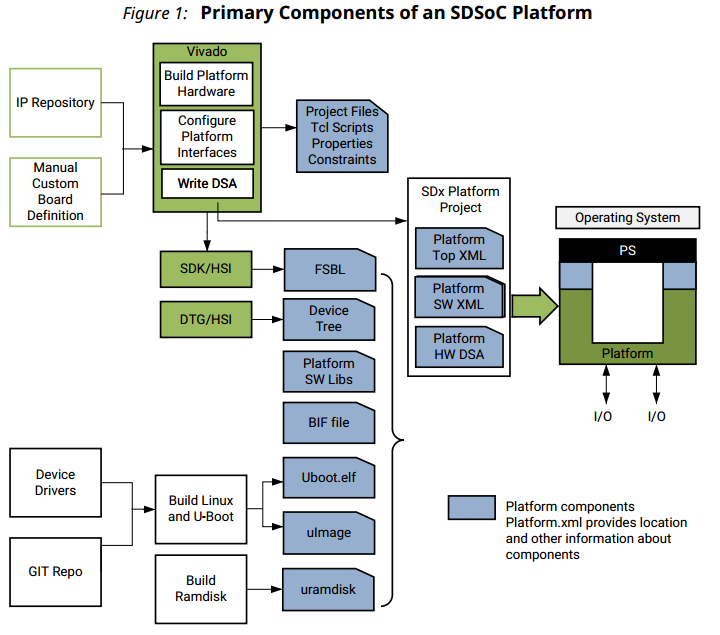
参考UG1146的内容，主要描述SDsoc平台的搭建。Sdsoc的开发前提是必须有一个底层平台。这个平台的搭建包括硬件平台和软件平台数据的创建。下面主要介绍创建的流程和注意事项。



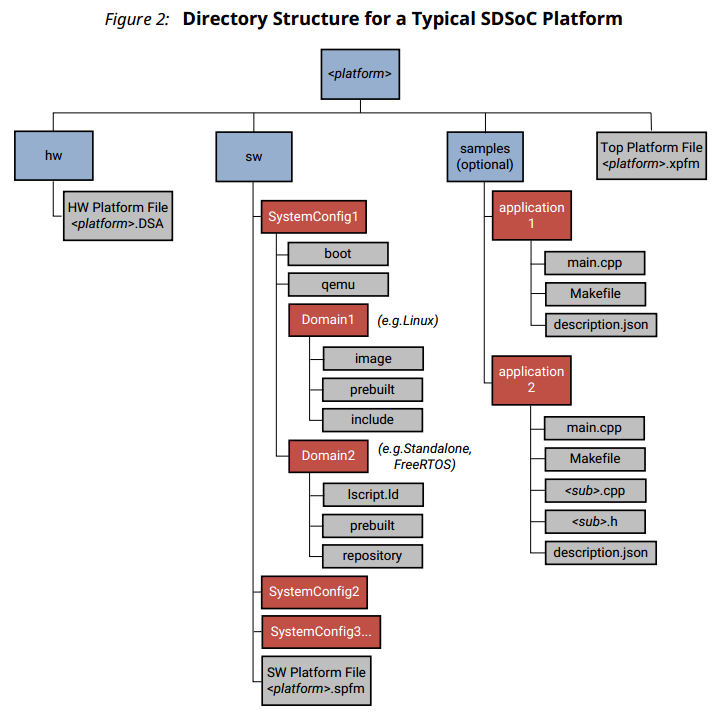
一个SDsoc主要由下面两个部分组成：

1. 硬件文件夹：

主要是生成DSA文件。（Device support archive），

同时硬件平台是用vivado生成的，还要还要导出hdf给SDK等等。

1. 软件文件夹：软件比较复杂裸机的和跑OS的都不一样。不同OS的也可能不同，这里大概列一下。



下面将根据UG1146的方法操作生成sdsoc的平台文件。但是顺序不是按照UG1146来的，而是先说硬件平台的创建，再说软件部分各文件的生成，最后说SDSOC平台的生成。这里主要以zynq平台来描述，不涉及microbliaze等其他架构。

1. **硬件平台生成流程和各环节注意事项：**

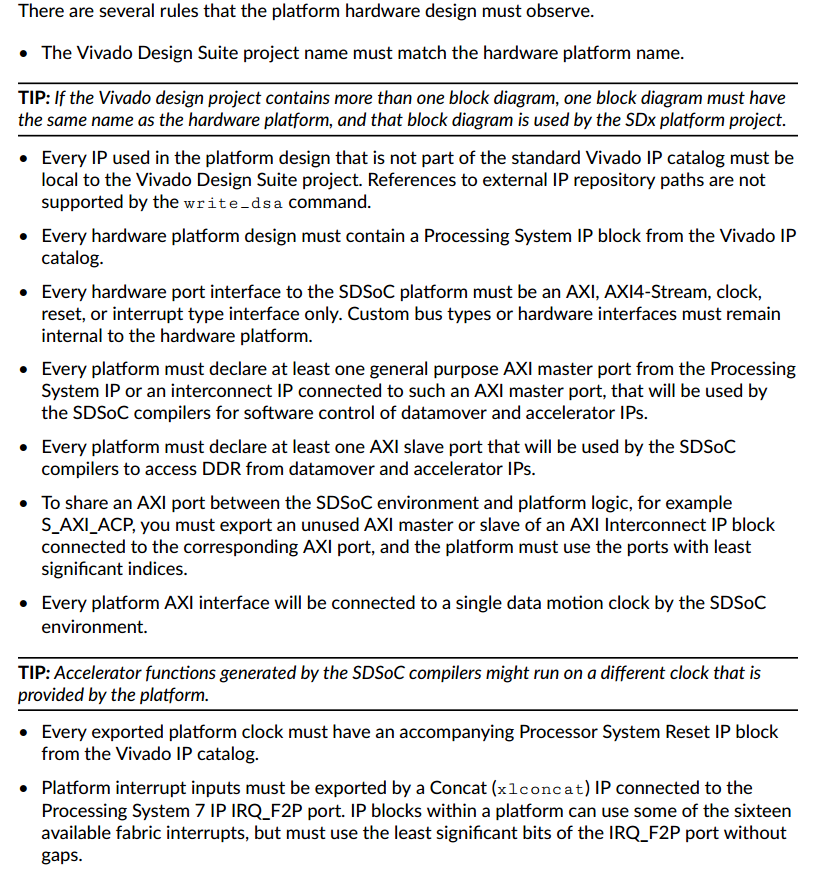
主要基于vivado和TCL操作。生成SDA文件为主。

* 1. 主要步骤就三步，但是每个步骤里面都有很多注意细节需要把握。
     1. Vivado建立硬件平台
     2. 很重要：tcl配置平台参数以及各种接口参数。
     3. 写入DSA文件。
  2. 重点注意事项
     1. Vivado工程名称和block design的硬件平台名称要一致。
     2. 工程中用到的任何自定义IP核路径必须在该VIVADO工程目录内。否则最后“write\_dsa”命令不支持。
     3. 必须包含一个PS或者microblaze。
     4. Sdsoc的平台每个模块的接口只能是 AXI,AXI4-STREAM，clock,reset,interrupt这几种。定制的接口或者总线啥的只能出现在自定义IP的内部。（怀疑：如果有其他形式的接口或者与或非、常量等IP，是不是应该也不影响SDA的生成，只是sdsoc不能使用他们而已。）
     5. 每个platform至少有一个PS端的GP AXImaster口或者它的interconnect IP互联矩阵，用于datamover和加速IP模块。
     6. 每个平台必须生命至少一个 AXI-slave端口，用于sdsoc编译器访从datamover问DDR和IP核的各种加速。（可以联想到为什么Xilinx的demo会预留很多空的时钟，复位模块，推测当用HLS做加速函数时候可以在PL端实现功能IP）
     7. 为了在sdsoc环境和平台逻辑之间共享一个AXI端口，比如S\_AXI\_ACP，就必须用一个无用的AXI互联矩阵IP连接到相应的AXI端口上。
     8. 每个平台的AXI接口将被sdsoc开发环境连接到一个独立数据迁移时钟上。

Tip:sdsoc编译器生成的加速函数功能块也可能运行在一个不同的时钟上，该时钟也是platform提供的。

* + 1. 每个导出的平台时钟必须包含一个相应的由PS的reset IP模块产生的复位信号。（就是一个时钟配合一个复位，但是这个复位是由特定的IP核产生的。）
    2. 所有中断最好通过concat IP核捕获并连接到PS端。

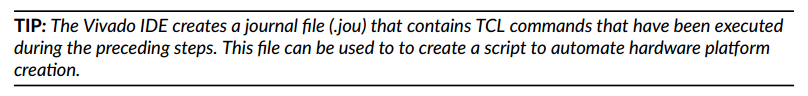
以下是上述中文描述所对应的UG1146原文描述，有可能中文描述不准确。



* 1. 创建vivado平台工程
     1. vidado建立工程，一切正常操作。有几个细节需要注意：

1. 注意creat block design的名称要和工程名称一样。
2. PS中SD卡必选，另外把TTC,SWDT也勾上为了配合Petalinux的配置要求。
3. 也可以添加自己的IP，如自定义IP,AXI GPIO,VDMA等等。完成一些自己习惯用的，或者必要的接口电路设计，这些不影响sdsoc的平台搭建。
4. 如果自建的IP有用到中断，也可以通过XLCONCAT IP先添加到PS上，后面在配置预留给sdsoc的中断属性的tcl命令时，可以修改下参数即可。如果所有中断都不用，而是留给sdsoc后期使用，则在用xlconcat IP添加中断时，要把输入的数量改成1，默认是2个。主要是配合后面的tcl命令操作，如果会修改命令了，这里可以再自定义吧。
   * 1. 在vivado搭建好以后，先validate Design一下，看看设计有没有错误，再进行下一步。
     2. 最关键的是通过PFM properties配置接口参数。可以通过图形化也可以用TCL命令。下面蓝色字体部分做一个简单的概述和demo。另外设置完tcl命令后，在2018.2版本后，可以通过**Window → Platform Interfaces→**Enable platform interfaces中查看各种属性配置是否OK，也可以通过这里进行图形化设置属性，不用TCL命令的。（最后生成DSA文件的路径的默认路径（C:\Users\usr\AppData\Roaming\Xilinx\Vivado），但是下面用TCL操作的会指定到工程路径下）
     3. 再次Validate the block design to ensure everything is correct, and save the design。
     4. 创建xdc管脚约束or时序约束等等文件 > Generate outputproducts > create hdl wrapper > export hardware。Bitstream生成与否不做强制要求，不过最好生成一下，然后在SDK里面做一些测试工程看看vivado工程搭建是否OK。

这一步操作后，在SDK目录下会生成个.hdf文件，这个在Petalinux创建过程中会用到。同时SDK的fsbl工程，linker script生成工程都是基于这个的。所以，如果修改vivado工程后，最好重新操作一遍以上的步骤。

* + 1. Write and validate the DSA using “write\_dsa”and “validate\_dsa”.sdsoc的定制平台硬件参数是基于这个DSA文件搭建的。
    2. 
  1. 定义平台属性（PFM）——语法参考UG1146

在完成硬件平台设计工程后，必须执行PFM来定义平台名称，配置平台接口例如时钟，中断，总线等等。这些属性只要设置一次后就存在工程当中。如果修改vivado工程后，建议重新输入一遍。基本配置主要有如下几点：

1. 设置platform的名称，该名称符合VLNV格式：

语法：set\_property PFM\_NAME string [get\_files design.bd]

Demo：

set\_property PFM\_NAME "xilinx.com:asdx:asdx:1.0" [get\_files C:/Users/Administrator/Desktop/alinx\_sdx/asdx/asdx.srcs/sources\_1/bd/asdx/asdx.bd]

1. 设置platform的接口属性，一共有如下四个属性：
   1. 1 PFM.AXI\_PORT
   2. 2 PFM.AXIS\_PORT
   3. 3 PFM.CLOCK
   4. 4 PFM.IRQ
   5. 语法：

set\_property PFM.AXI\_PORT { <port\_name> {parameters} \

<port2> {parameters} ...} [get\_bd\_cells <cell\_name>]

set\_property PFM.AXIS\_PORT { <port\_name> {parameters} \

<port2> {parameters} ...} [get\_bd\_cells <cell\_name>]

set\_property PFM.CLOCK { <port\_name> {parameters} \

<port2> {parameters} ...} [get\_bd\_cells <cell\_name>]

set\_property PFM.IRQ { <port\_name> {} <port2> {} ...} \

[get\_bd\_cells <cell\_name>]

* 1. DEMO：

set\_property PFM.AXI\_PORT { \

M\_AXI\_GP0 {memport "M\_AXI\_GP"} \

M\_AXI\_GP1 {memport "M\_AXI\_GP"} \

S\_AXI\_ACP {memport "S\_AXI\_ACP" sptag "ACP" memory "processing\_system7\_0 ACP\_DDR\_LOWOCM"} \

S\_AXI\_HP0 {memport "S\_AXI\_HP" sptag "HP0" memory "processing\_system7\_0 HP0\_DDR\_LOWOCM"} \

S\_AXI\_HP1 {memport "S\_AXI\_HP" sptag "HP1" memory "processing\_system7\_0 HP1\_DDR\_LOWOCM"} \

S\_AXI\_HP2 {memport "S\_AXI\_HP" sptag "HP2" memory "processing\_system7\_0 HP2\_DDR\_LOWOCM"} \

S\_AXI\_HP3 {memport "S\_AXI\_HP" sptag "HP3" memory "processing\_system7\_0 HP3\_DDR\_LOWOCM"} \

} [get\_bd\_cells /processing\_system7\_0]

set\_property PFM.AXIS\_PORT {AXIS\_P0 {type "S\_AXIS"}} \

[get\_bd\_cells /zynq\_ultra\_ps\_e\_0]

set\_property PFM.CLOCK { \

clk\_out1 {id "0" is\_default "true" proc\_sys\_reset "proc\_sys\_reset\_0" } \

clk\_out2 {id "1" is\_default "false" proc\_sys\_reset "proc\_sys\_reset\_1" } \

clk\_out3 {id "2" is\_default "false" proc\_sys\_reset "proc\_sys\_reset\_2" } \

clk\_out4 {id "3" is\_default "false" proc\_sys\_reset "proc\_sys\_reset\_3" } \

} [get\_bd\_cells /clk\_wiz\_1]

set\_property PFM.CLOCK { \

FCLK\_CLK0 {id "4" is\_default "false" proc\_sys\_reset "rst\_ps7\_0\_100M" } \

} [get\_bd\_cells /processing\_system7\_0]

set intVar []

for {set i 0} {$i < 16} {incr i} {

lappend intVar In$i {}

}

set\_property PFM.IRQ $intVar [get\_bd\_cells /xlconcat\_0]

1. 生成SDA文件 使该DSA文件生效

write\_dsa -force C:/Users/lhb/Desktop/system/system.srcs/sources\_1/bd/system/system.dsa

validate\_dsa C:/Users/lhb/Desktop/system/system.srcs/sources\_1/bd/system/system.dsa

1. **基于SDK生成SDsoc定制平台的standlone必要文件生成流程：**

**Sdsoc定制平台的搭建是基于软件和硬件两部分完成的，vivado先生成一个基本的硬件框架，以DSA文件形式倒给sdsoc，同时vivado还生成支持SDK或者Petalinux创建相应配套软件的hdf文件。SDK或者Petalinux基于这个会生成sdsoc需要的软件元素。下面说下不用petalinux 生成standlone生成sdx配置文件，主要靠SDK环境。基于这些软硬件文件，SDsoc能直接生成在SD卡中启动的最终运行固件，而如下基于SDK的软件仅支持sdsoc生成裸机运行的SD卡启动固件。而基于Petalinux既可以生成裸机启动文件也能生成基于Petalinux运行的带系统的固件，后期补充：**

* 1. **基于SDK生成sdsoc必要的软件文件有如下几个：**
     1. **fsbl.elf**
     2. **standlone.bif**
     3. **BOOT.BIN**
     4. **linker script**
     5. **一个独立文件夹：/xxx\_boot，把上面四个文件放在里面。**
  2. **fsbl.elf + standlone.bif + BOOT.BIN文件生成：**
     1. **创建FSBL工程，**
     2. **直接Create Boot Image；修改.bif文件名，改成standlone.bif**
     3. **不要添加bitstream文件，确保只含有fsbl.elf，点击Create Image。**

**这样以后，在SDK > FSBL的文件夹目录下，就能找到fsbl.elf ， standlone.bif ， BOOT.BIN三个文件。打开相应的.bif文件，按照下面的格式内容编辑并保存:**



/\* standalone \*/

the\_ROM\_image:

{

[bootloader]<fsbl.elf>

<bitstream>

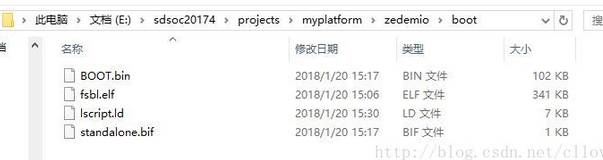
<elf>

}

* 1. **创建linker script文件：**

**在SDx中，平台创建过程需要一个新的链接器脚本。这个链接器脚本基本上确保了针对自定义平台的任何应用程序代码驻留并运行在执行板上的DDR内存中。**

* + 1. **我们将创建一个空应用程序empty\_app。**
    2. **右击点击empty\_app文件夹->Generate Linker Script ，在Heap Size中输入402653184（384M）；在Stack Size中输入262144（256K）（这些值可以更改，如果DDR更大，可以把堆栈值都改大，输入的数值是Byte单位的，如果要设置512MB大小，就输入512\*1024\*1024的结果）；点击Generate。**
  1. **在vivado工程以外的地方，建一个/xxx\_boot文件夹，把上述4个文件都放到里面，作为创建sdx的custom platform的standlone应用的元素。**

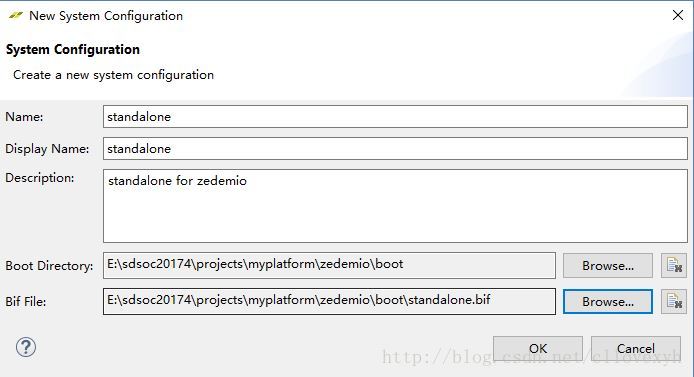


1. **使用SDx IDE创建定制的平台：**
   1. **打开 SDx IDE,关闭欢迎页面（否则不显示创建的platform project页面）创建SDX工程中选择platform project。**
   2. **指定到DSA文件的路径（vivado\_pro/system.srcs/sources\_1/bd/system/system.dsa）。然后就进入软件配置了，如下图：**



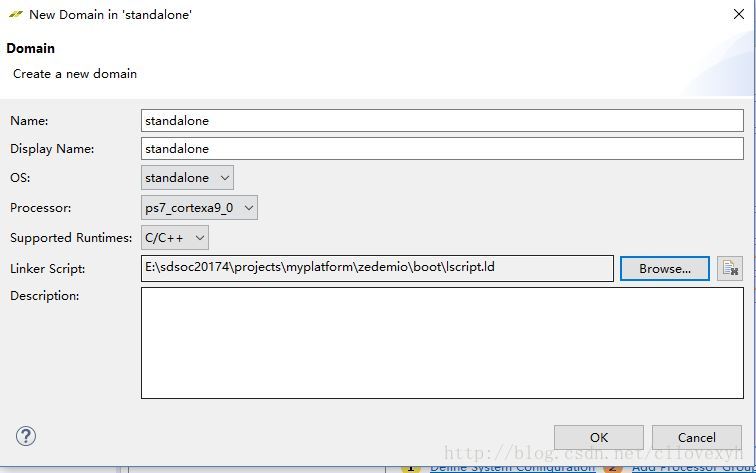
**一下1/2/3/4步操作都可以重复操作的。比如同时创建支持裸机、Petalinux、freeOS等等方式。一个platform可以支持多种应用开发方式的。下面值讲standlone的过程**

* 1. 点击Define System Configuration：



**在接下来的弹框中填入以下信息，Name和Display Name表示系统的名字，这里叫standalone（裸机）；Description随便写吧；Boot Dictionary就是前面新建的boot文件夹；Bif File就是在boot文件夹中的standaalone.bif文件；点击OK。**

* 1. 点击Add Processor Group/Domain：



1. 在弹框中填入以下信息，其中Name和Display Name就是系统的名字，OS选择standalone（裸机），Linker Script选择boot文件夹中的lscript.ld;
2. 配置BSP支持包路径：指定到SDK创建的FSBL\_bsp下的.mss文件（其实任何SDK下的工程的bsp目录都可以。）
3. 配置Libraries路径指定到SDK创建的FSBL\_bsp下的include文件夹（其实任何SDK下的工程的bsp目录都可以。）



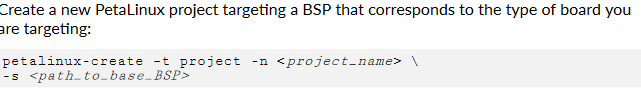


点击OK。

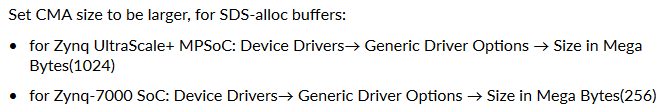
* 1. 接着点击Generate Platform产生平台
  2. 接着点击Add to Custom Repositories，添加这个平台

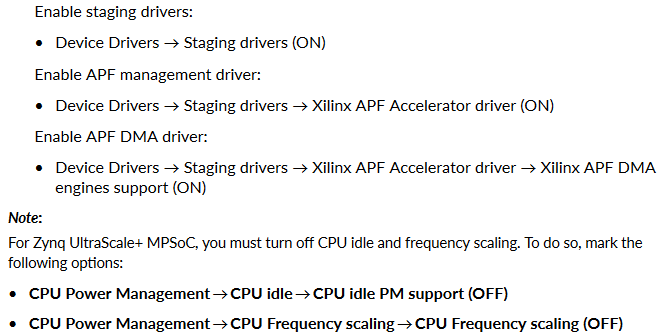
1. **基于定制平台的app开发：**

**如下还不会写加速app开发，只是把基于SDK开发的过程写一下。具体需要摸索**

1. 新建一个SDSoc Application Project，并取名；
2. 在接下来的选择平台界面，就出现了刚才创建的这个平台，选择这个平台，点击Next；
3. 接着选择系统配置，由于只添加了standalone，所有只有standalone可选
4. 由于没有添加例程，只有选择创建一个Empty Application；点击Finish。
5. 之后就可以把SDK下的应用工程在这里开发，当然这样只能跑vivado下添加的IP功能。如果要开发带硬件加速的功能，还要继学习sdsoc加速模块写法，并把相关功能添加到硬件加速中去编译。这才是sdsoc的核心，后面将进入这部分的学习。
6. 在配置第3,4步时，有个使用hadware acc硬件加速的选项要勾选。这样在完成第5步编译后，会在该sdsoc工程目录下生成一个SD文件夹，里面包含SD卡启动和运行的所有文件。如果不勾选，将不能生成启动文件。这有elf文件。毕竟原理上是不会生成bitstream的，这个elf和用vivado下的sdk开发的效果一样的。
7. 在底5步完成后，在sdsoc的工程下不管是debug还是release版本下都会生成相应的vivado工程， 如果你用了硬件加速模块，那么该vivado工程会在你原来vivado工程的基础上添加生成相应的加速电路模块。如果没有引用，则和原来vivado工程一样。
8. **基于Petalinux的SDSoC平台创建文件和SDSoC平台创建方法：ug1146~P83**
   1. 首先要熟悉Petalinux的创建方法，在这基础上结合下面说的内容才可行。这里以ZYNQ为例，不涉及UltraScale+MPSoC.(另外根据个人经验，可以自己去linaro下载通用的linaro文件系统添加下面必要的库后，也可以支持SDSoC在Linux下运行的，后面会细说)
   2. 首先要有个意识就是(libsds\_lib.so)以前是静态链接库，而现在是动态链接库。必须在Petalinux工具生成该平台的文件系统的时候就要添加进去。放置的路径默认在文件系统的/usr/lib下面。这个库有的添加方法是从SDx(SDSoC)安装路径下的指定目录下<SDX\_Install\_Dir>/target/<architecture>-linux/lib拷贝到你的Petalinux生成的文件系统里。Petalinux支持其他linaro文件系统的，你也可以自己下载文件系统并拷贝进去。
   3. 创建一个PetaLinux工程，目的是生成对应vivado工程的BSP。虽然这里给的command和PTALinux创建工程的有所不一样，但本质上和PetaLinux里创建工程是一样的。以及后续的导入hdf等操作。
   4. **配置SDSoC特殊要求的PetaLinux工程，常规PetaLinux配置依旧需要，这里说下SDSoC的特殊配置需求：**

petalinux-config -c kernel后，有如下两个操作：





petalinux-config -c rootfs有如下配置要求：

重点添加 stdc++libs,(可以添加opencv，build essential多个库。方便开发)路径：



如果是第三方linaro文件系统，也可以按照网上通常方法安装这个库，有可能linaro自带。

（**另外特别说明**：针对自定义的SDSoC系统，PetaLinux在配置rootfs时可以添加很多第三方库，比如OPENCV库，但根据实际经验，这个OPENCV库并不完整，如果SDSoC生成的执行文件xxx.elf用到opencv库，则有可能在ZYNQ上运行不起来。最好是找一个或者cmake一个完整的opencv库，是可以运行的。）

设备树需要添加如下节点：

路劲在PetaLinux的如下路径：

/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi

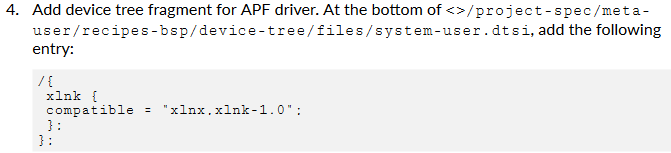
/{

xlnk {

compatible = "xlnx,xlnk-1.0";

};

};



最后Build the PetaLinux image:

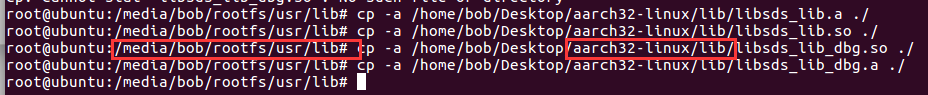
petalinux-build完即可，不需要生成BOOT.BIN因为SDSoC做硬件加速是会改变PL的，既bit文件是变化的。而且SDSoC其中一个任务就是需要这里的文件来自己生成BOOT.BIN。但是你也可以合成一下生成BOOT.BIN。就当做常规Petalinux使用，看看系统能否正常启动，外设是否正常工作。做个校验工作。

Petalinux**文件系统的二次加工——添加第三方库：**

1. PetaLinux在build后会生成rootfs文件，而且有很多压缩格式，这里以rootfs.tar.gz这个最常用的为例。这里都是以第二分区启动为例。ramdisk的比较麻烦因为buid完后文件系统和内核还有设备树被合成成了一个.ub文件，不好添加第三方库。
2. 添加opencv（建议用opencv3，而opencv4有些目录可能调整了，旧代码可能会要修改。）库和libsds\_lib库。方法有两种：
   1. 对于自定义SDSoC平台过程中，其实在PetaLinux配置rootfs的步骤中就可以预先添加opencv库，但是貌似这个opencv库不完整，而且不支持自定义的SDSoC平台，所以不能添加，要自己添加。
   2. 在PC上解压rootfs.tar.gz后，把这两个库放到指定目录下，然后可以再次压缩打包这个文件系统成一个新的，后期可以多次使用。
   3. 直接拷贝或者同步rootfs.tar.gz到SD卡的第二分区并解压。然后再添加这两个库。
   4. 这两个库的路径：
      1. libsds\_lib放到/usr/lib就可以了。
      2. opencv一般放到/usr/local下，但还要等PetaLinux起来后配置一下环境变量。具体操作见OPENCV的移植方式,下面概述一下。
         1. 在linux下解压一个rootfs文件夹，并解压一个编译好的opencv文件包到一个文件夹。
         2. 把OPENCV这个文件夹整个放到rootfs的/usr目录下。
         3. 重新打包这个rootfs为rootfs.tar.gz备份，并解压到SD卡第二分区里，作为现在的文件系统，简单打包指令如下，sudo权限：

zip SDSoC\_PetaLinux\_rootfs.zip ./SDSoC\_PetaLinux\_rootfs/\*

对于libsds\_lib，我把这个库拷贝到linux下后，再挨个文件拷贝到SD卡解压好文件系统的目录下的。做个演示截图



1. 个人推荐去linaro官网下一个最新的linaro文件系统，添加上面的两个库。根据个人经验，Petalinux裁剪太厉害，自己要配置的东西太多。经常出现比如apt-get ,update，yum等等常用指令都没有。不方便自己扩展。其实文件系统只要是完整的，PETALINUX生成的Image.ub都能支持启动的。

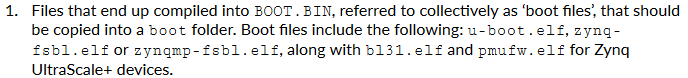
* 1. 导入PetaLinux生成的各种需要的文件给SDSoC创建平台使用：
     1. 在PetaLinux工程的如下路径<petalinux\_project>/images/linux/下有一些重要的文件，并将它们分别放在两个目录下，命名和架构如下：

1，/boot :(用于让SDSoC生成BOOT.BIN文件的基本文件,注意不需要bit文件。)

u-boot.elf

zynq-fsbl.elf

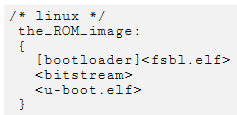
如果是mpsoc,有所不同，这里不细说。



2，/image : (主要是一些要放在SD卡上的文件，你也可以放一些自己的文件。PetaLinux主要提供的是image.ub文件。就是linux内核了)

image.ub

3，BIF文件一个：用于合成BOOT.BIN用的，文件名“boot.bif”，可以放到一个文件夹里：



/\* linux \*/

the\_ROM\_image:

{

[bootloader]<fsbl.elf>

<bitstream>

<u-boot.elf>

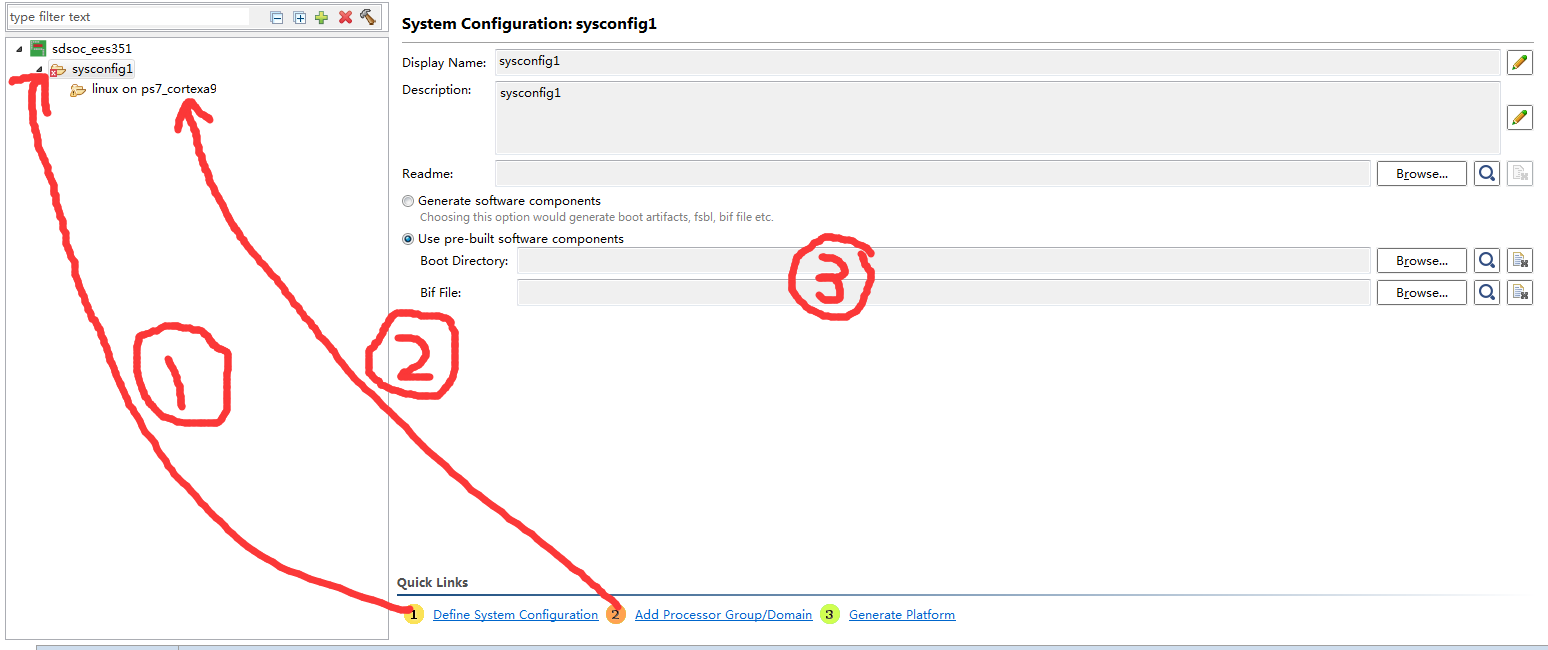
}

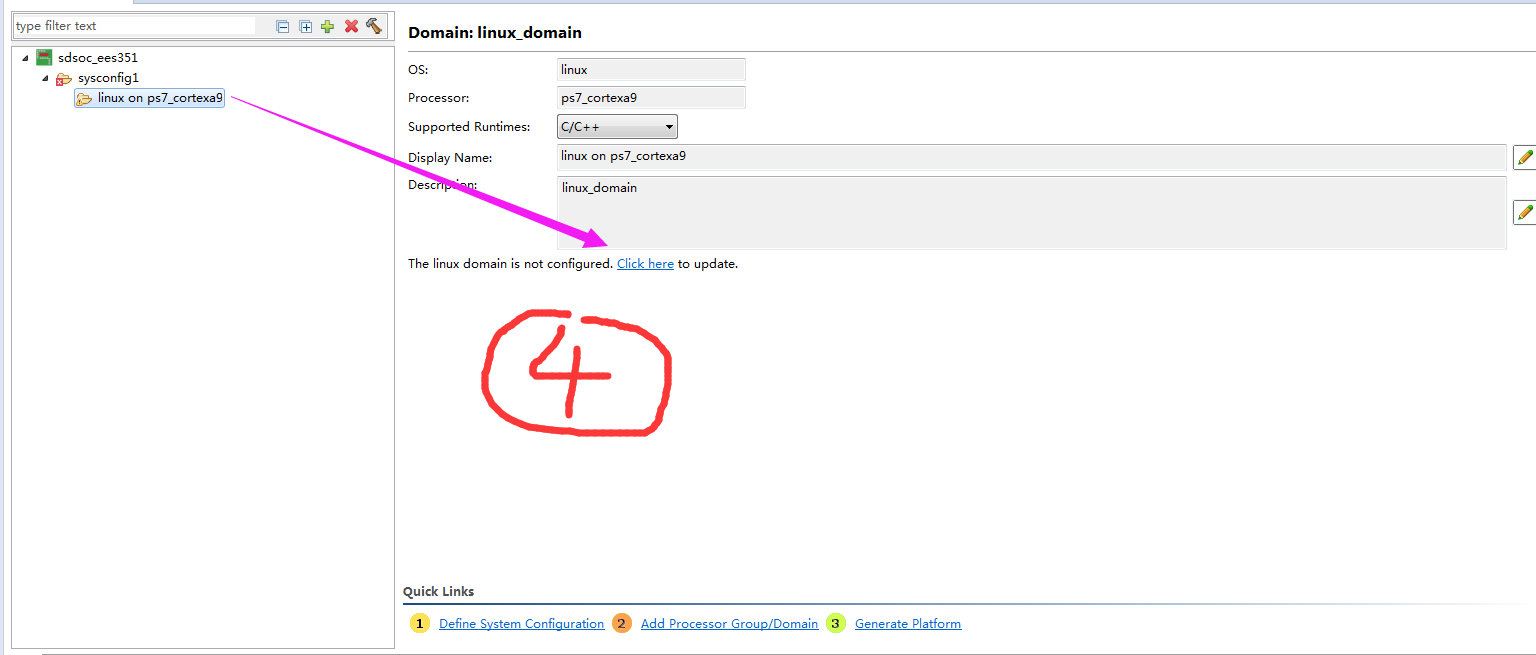
4，DSA文件：在建立vivado工程时生成的DSA文件也需要，可以拷贝出来放到一个文件夹里：

* + 1. 把以上4种文件放到一个共同的文件夹里作为SDSoC创建LinuxOS所需要的文件，如果有强迫症，可以把配置好的rootfs文件系统也放到一起。以后就是一套完整的搭建资料了。创建platform的流程参考standalone的流程。

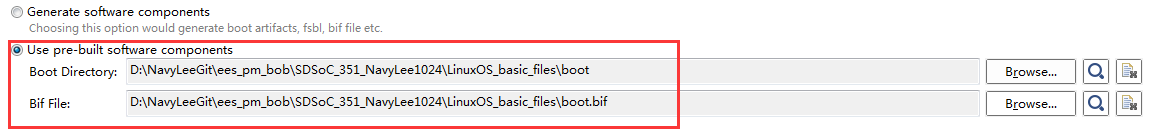
这里说下SDSoC2018.3的创建流程，和2018.2略有区别。主要是理清下几个层级目录，以截图大概说下：

在导入DSA后，会出现如下，其实你不需要再重复按照软件下面那三个1,2,3彩色标签的步骤操作。默认自动帮你创建了sysconfig1方案，已经执行了1,2的操作。你只要把手写的3号和4号区域里面的路径配置好就行。这块看的有点糊涂，所以特别说明下。

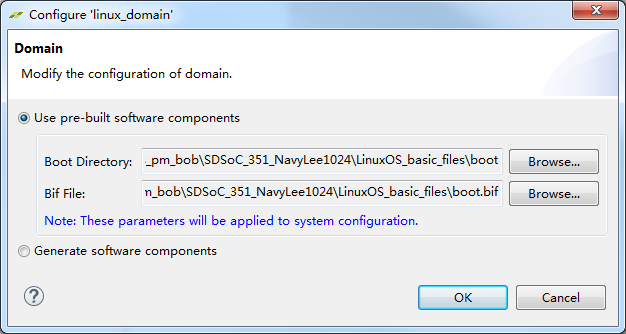




在sysconfig1的右侧编辑框里面选择如下：主要指定boot文件夹和bif文件：



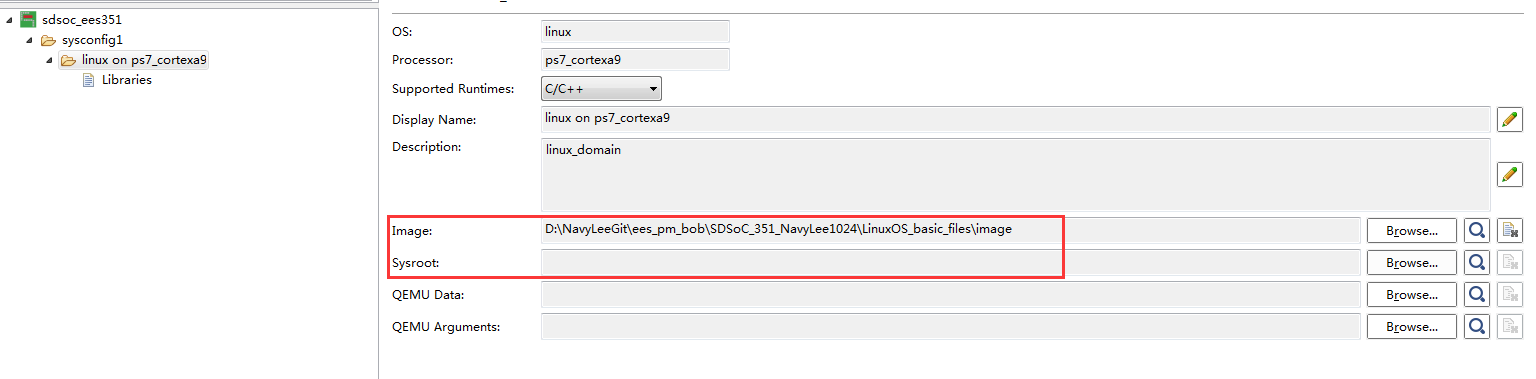
在linux\_on\_ps7\_cortexa9的右侧编辑框里先点“click here”,见手写的4区域。会有弹框，如下图做选择：



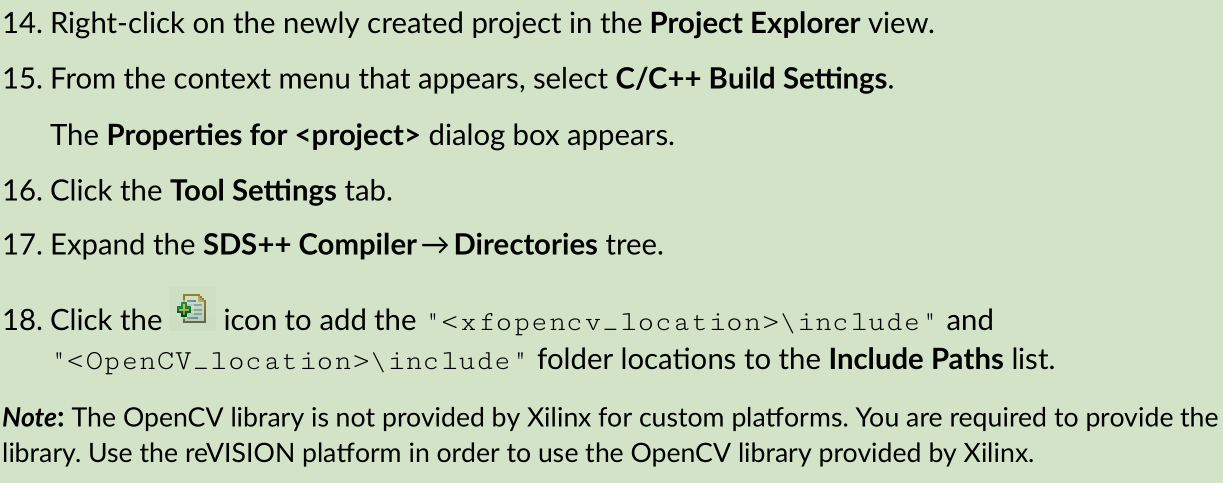
再指定Image和Sysroot的路径。Image必须要指定，否则无法创建。其实这两步只是SDSOC在后期帮你生成SD卡启动的一套文件做准备：

Image和rootfs就是在Petalinux下生成的image.ub，SDSoC不会修改他，只会copy到你相应的工程下。

SDSOC主要是根据工程生成新的bit.再根据bif文件把boot文件夹下面的文件和这个bit文件合成一个BOOT.BIN文件。



* 1. SDSoC必要配置库安装：
     1. **交叉编译的opencv库：**根据2018.3版本的ug1233说明，SDx自带的opencv库是不支持自定义板卡的自定义SDSoC平台的。这里说下步骤（这个库也是Petalinux文件系统下需要安装的交叉编译库）：
        1. 有一个自定义的platform，并建立一个需要用到opencv的sdsoc工程。
        2. **指定Compiler下opencv的include路径**。参见2018.3版本的ug1233的28页添加方法，后面会说xfopencv添加方法，和这边一样：一定要看2018.3版本哦！！！2018.2版本没说，很坑爹！后面有新版本再补充说明，特别还要留意感慨下Note小字。

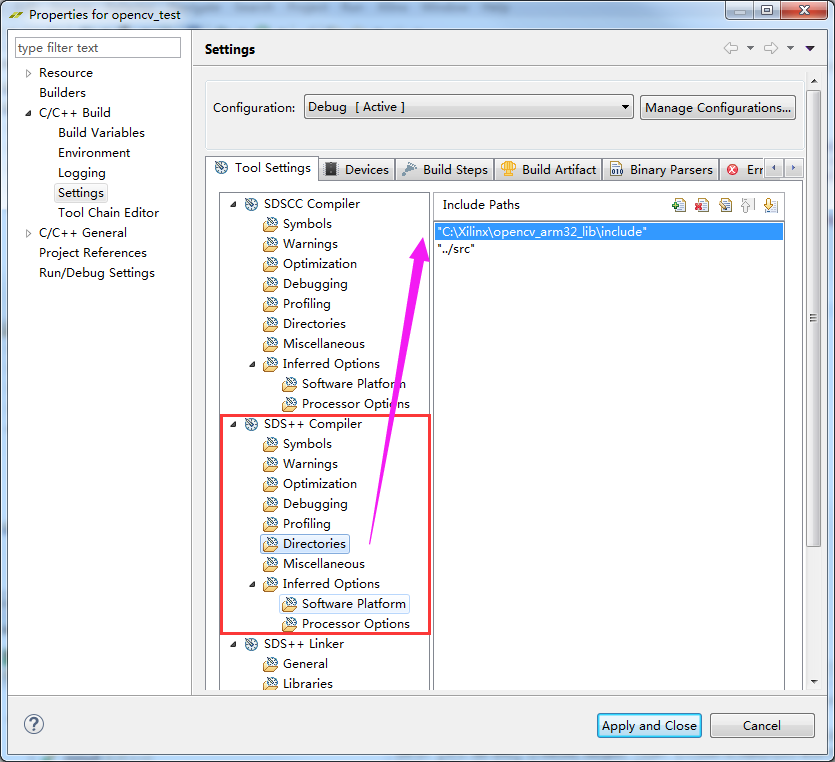


这个include的路径有三个，经验证后有两个地方可用，：

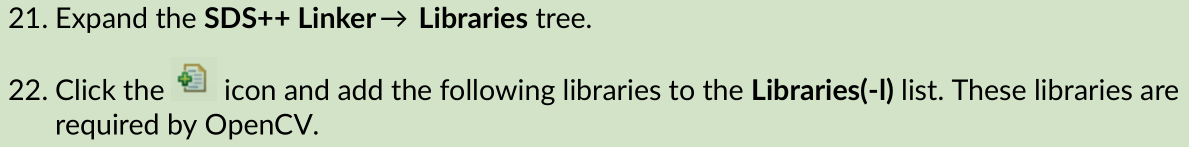
1. 在Xilinx的SDK安装路径下面有opencv，这里选arrch32针对非mpsoc：C:\Xilinx\SDK\2018.3\data\embeddedsw\ThirdParty\opencv\aarch32\include
2. 在Xilinx的SDX安装路径下面有opencv：C:\Xilinx\SDx\2018.3\include
3. 自交叉编译的opencv库，你可以放到任意路径：

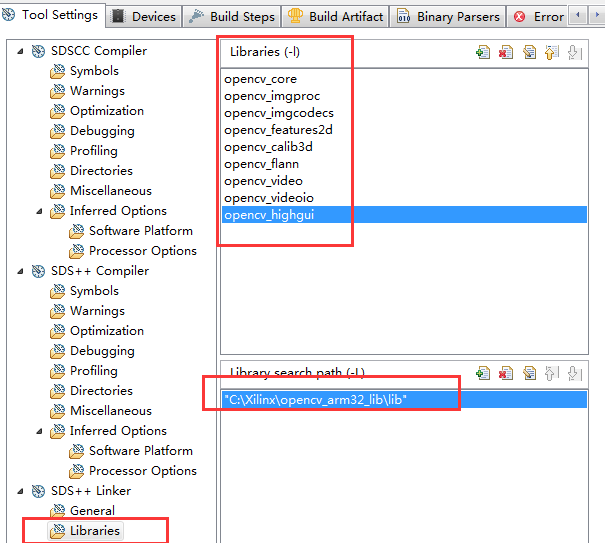
C:\Xilinx\opencv\_arm32\_lib\include

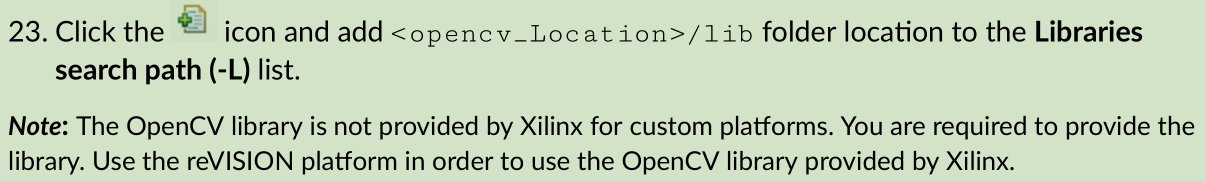
这里只有1和3的openv的include可以用。2不可以。估计是给他特定平台用的。推荐用3，毕竟下面一步就是指定opencv链接库了。对于自定义的平台智能用自编译的opencv库。干脆都指定同一个地方。

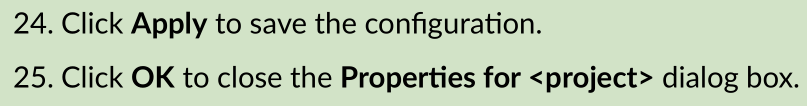


* + - 1. **指定Linker下opencv的链接库路径，**参见2018.3版本的ug1233的29~30页：

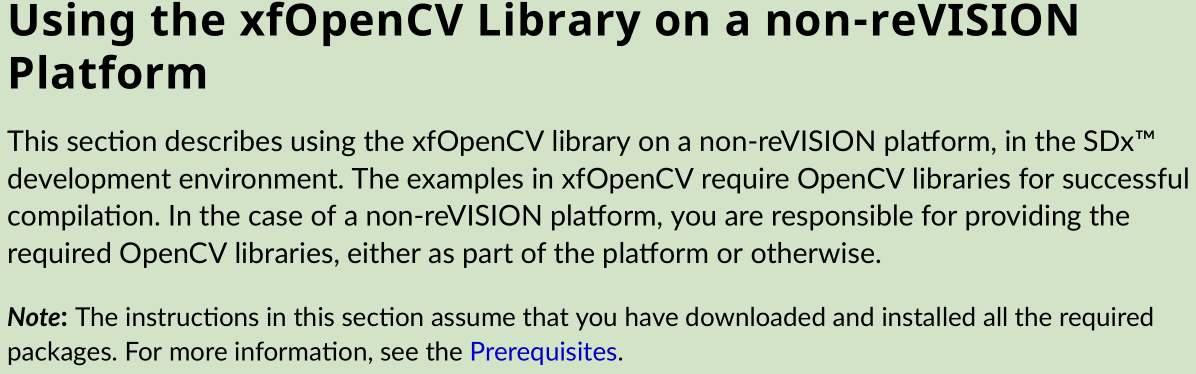




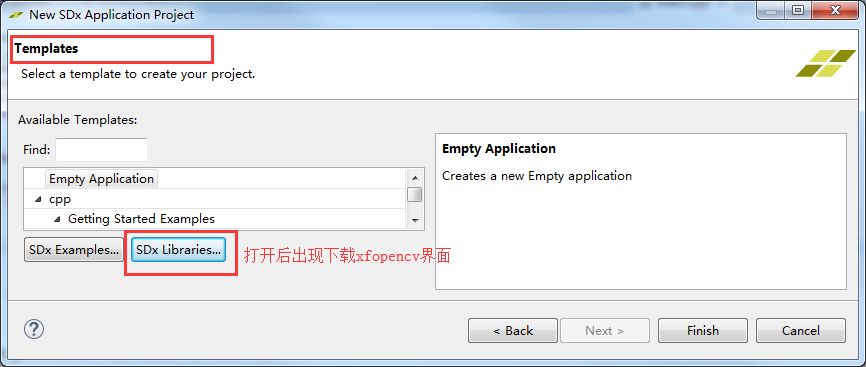


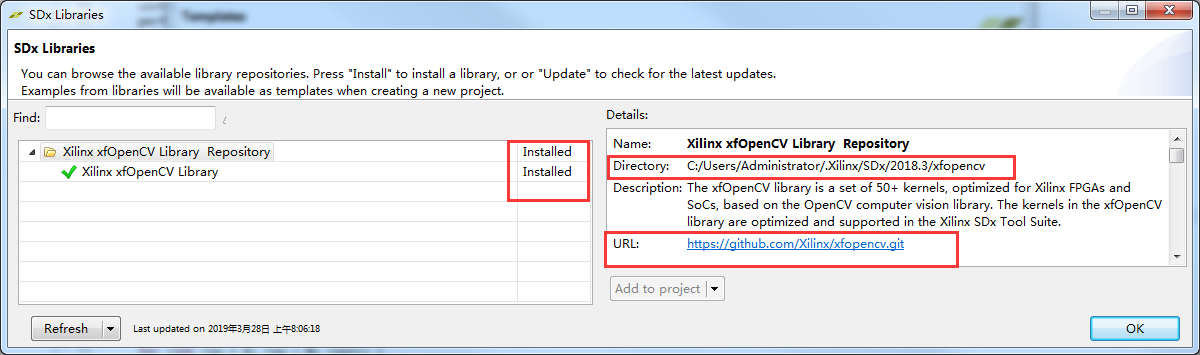


* + 1. 自定义平台添加xfopencv库方法，参见2018.3版本的ug1233从P26页到P33页。这里主要是自定义平台，没用reVISION包的。上面的opencv也是参考这段操作的：这里直接上截图：

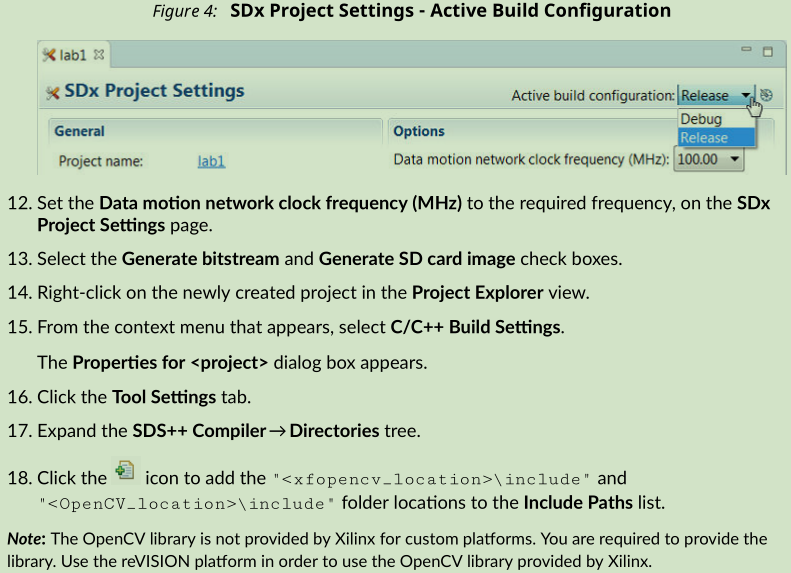
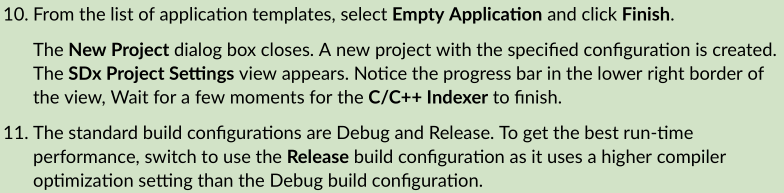


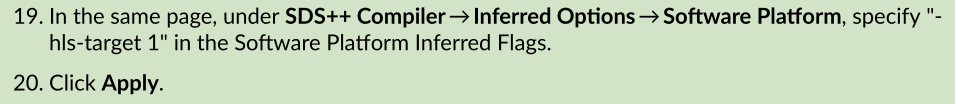
自定义好平台后，要求下载好xfopencv，其实在第一次创建工程时候在导入Templates 的界面就可以直接下载xfopencv，下载方法和路径如下图所示：如果已下载直接建工程

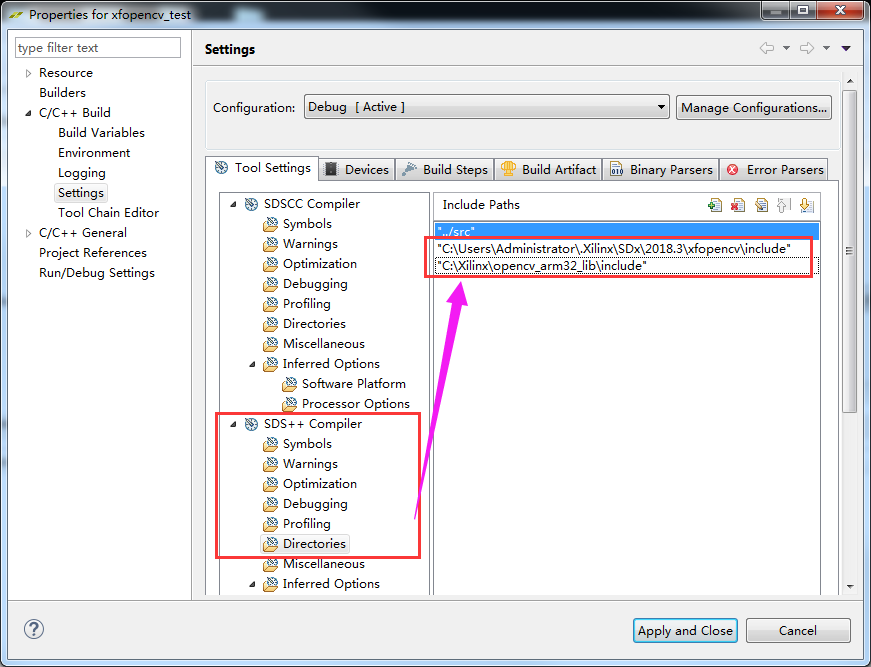


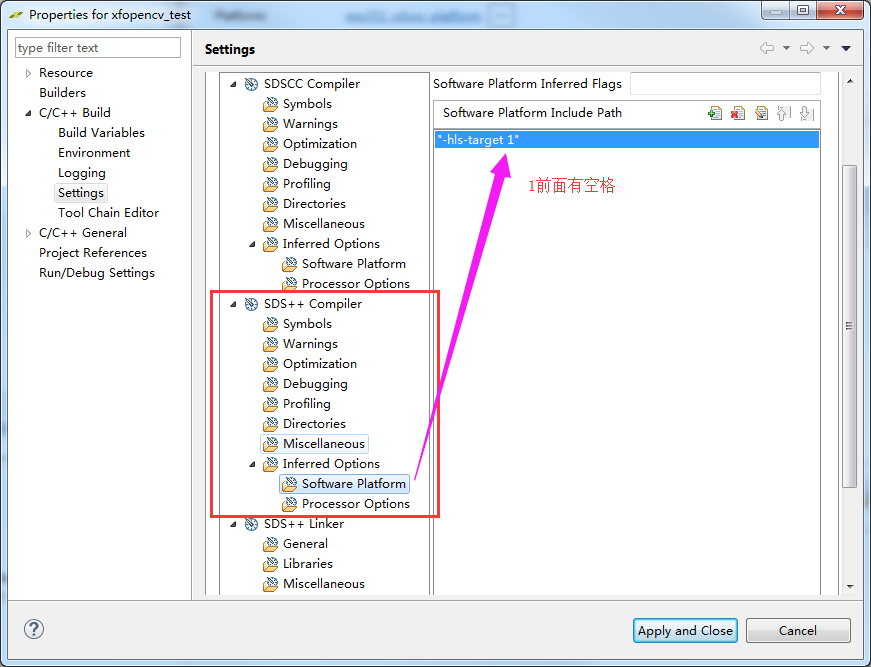


下面就是建空工程，并配置硬件加速参数，注意选择release或者debug后，分别都要配置的。添加opencv和xfopencv的相应目录的路径

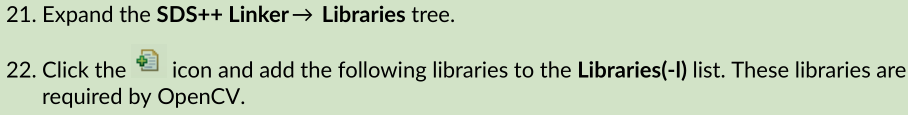


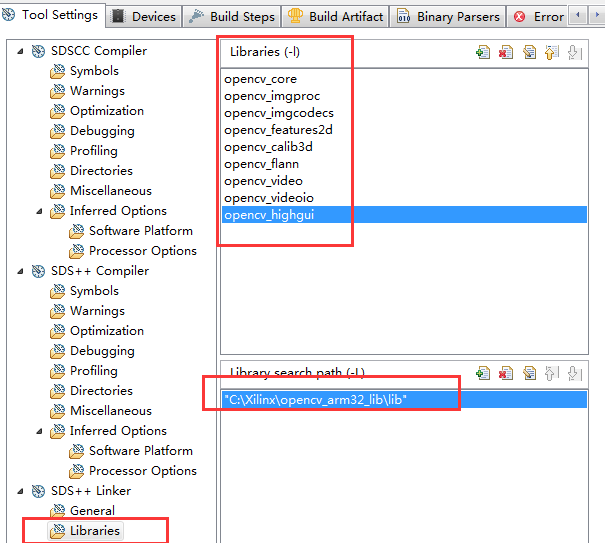


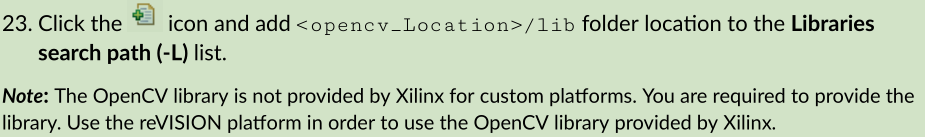


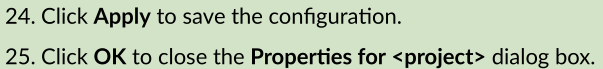


以下是添加opencv的链接库，这里再记录一遍

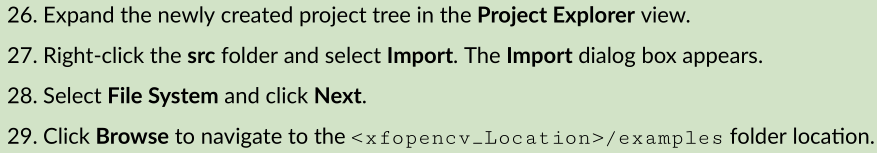




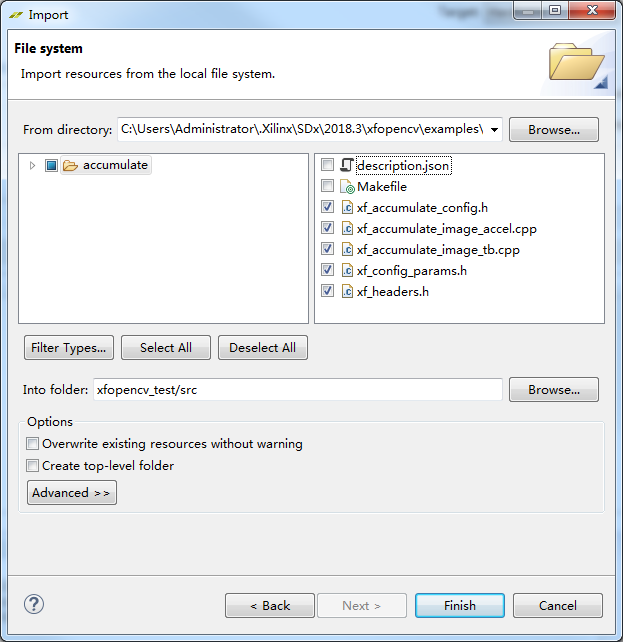




下面是导入一个xfopencv的案例

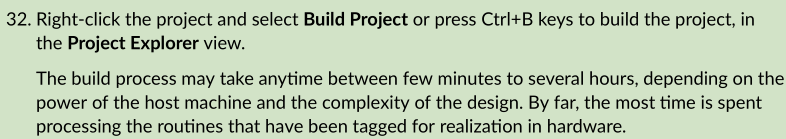


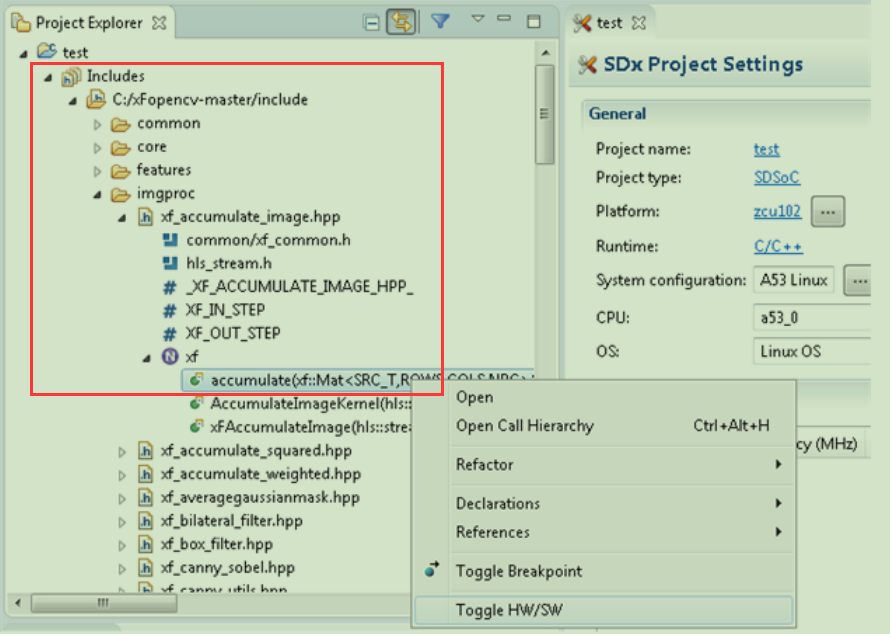




下面是添加要硬件加速的函数，这里要注意的是，加速的是xfopencv函数，不是自己写的。这里可以思考下！







下面的操作是拷贝生成的elf和boot.bin文件到SD卡里，放到ZYNQ板卡上执行就行了。注意第35条的Note也强调了下位机的文件系统里要有opencv的库，同时要配置好环境变量！

