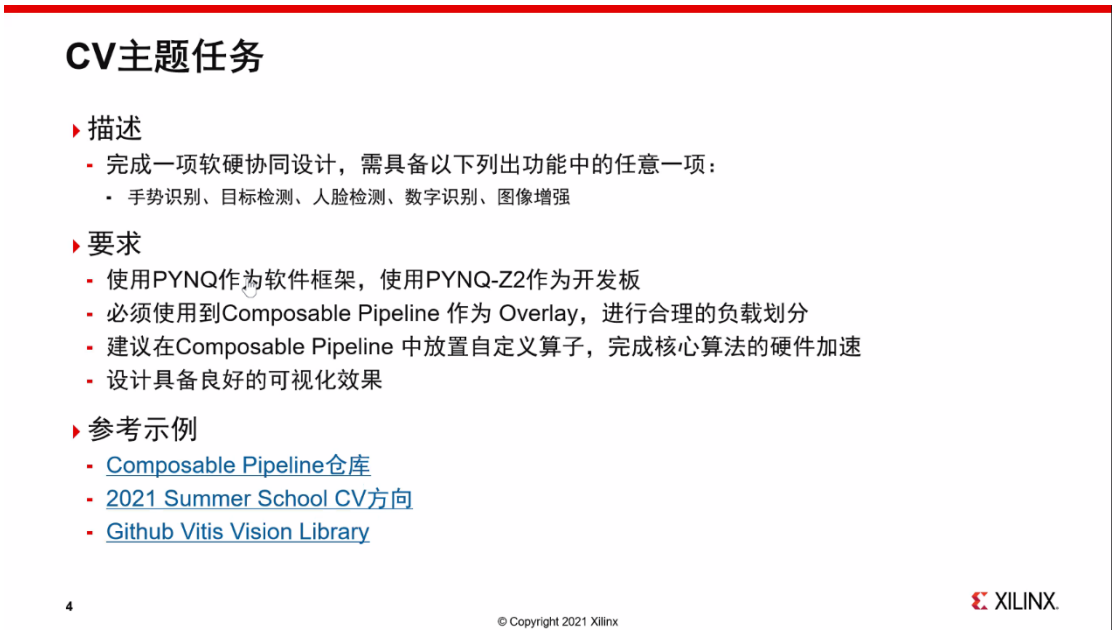
目　录

CV 方向要求



需要完成的功能：

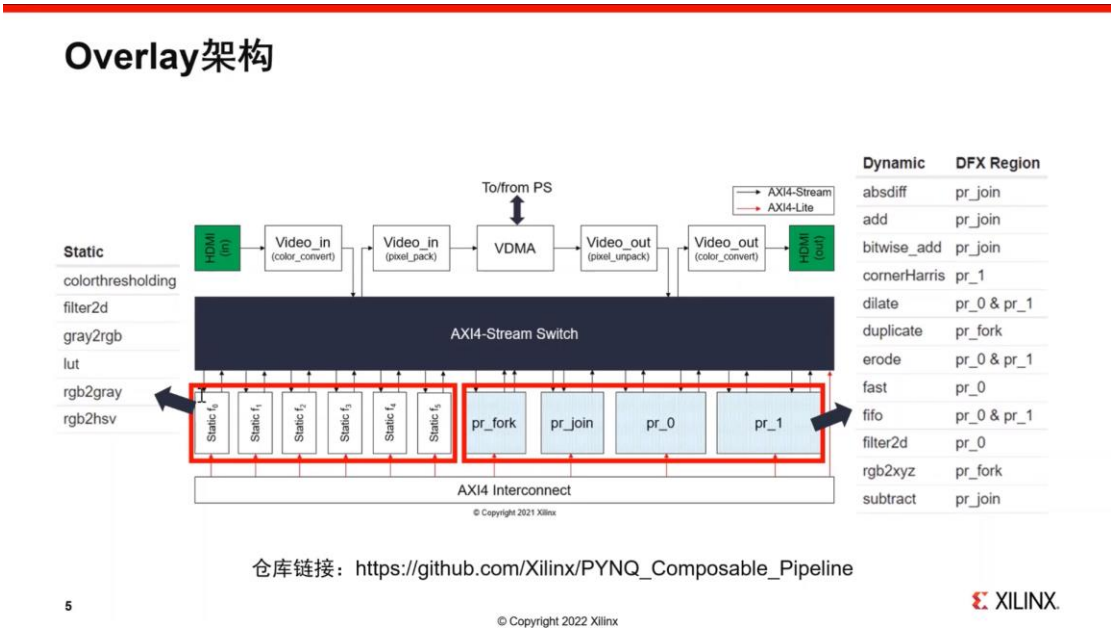
手势识别、目标检测、人脸检测、数字识别、图像增强。
完成软硬件协同设计

完成要求

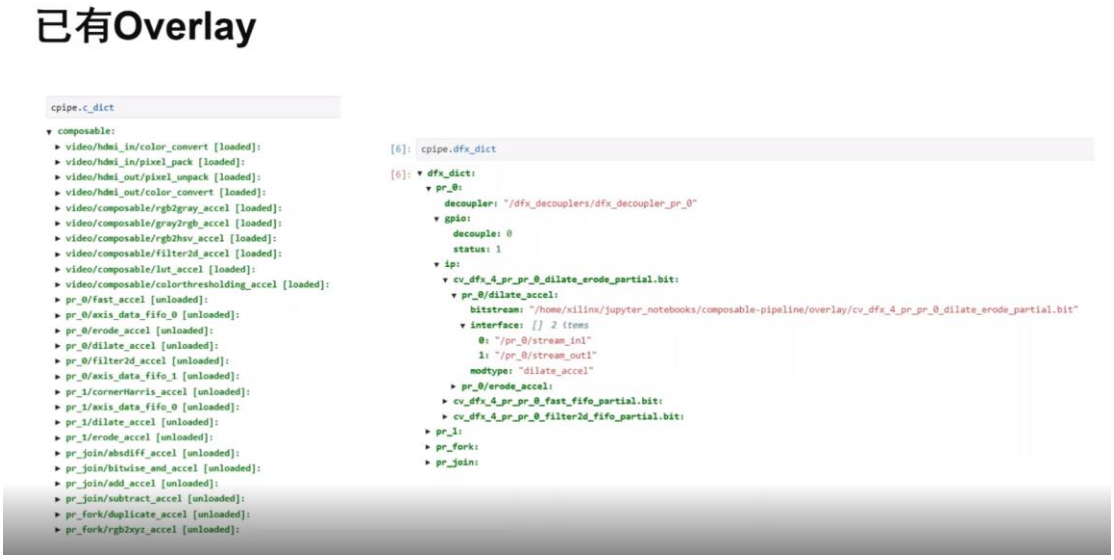
- 使用 PYNQ 以及 PYNQ-Z2；
- 使用 Composable Pipeline 作为 Overlay；
- 在 Composable Pipeline 中放置自定义算子，完成核心算法的硬件加速，可以修改 static 区域，也可以修改 DFX 区域；
- 也可以导入 VDMA 之后，进行 Python 的处理；
- 良好的可视化效果；
- 在 GitHub Vitis Vision Library 中有 Xilinx 优化过的 IP 可以进行使用。

基础知识

Overlay 架构



6 个 static 静态算子都是常见的，在旁边的动态区可以放置动态重构的算子，可以进行自定义配置和切换。VDMA 是 PS 和 PL 之间一个交互的方式，AXI4-Stream Switch 类似一个转换机。DFX 在左边浅蓝色的动态区。



可以通过在 Jupyter 中输入最上面那一条命令来查看 DFX 的有关信息。

静态 Static IP 调用

下面是静态区调用 composable pipeline。

1. static ip调用

- 目标：实现rgb，hsv与灰度图相互转化，2d滤波等简单功能
- 第一步：下载Overlay
- 第二步：获取功能函数ip

```
video_in_in = cpipe.video.hdmi_in.color_convert
video_in_out = cpipe.video.hdmi_in.pixel_pack

filter2d = cpipe.video.composable.filter2d_accel
rgb2gray = cpipe.video.composable.rgb2gray_accel
gray2rgb = cpipe.video.composable.gray2rgb_accel
rgb2hsv = cpipe.video.composable.rgb2hsv_accel
colorthr = cpipe.video.composable.colorthresholding_accel
lut = cpipe.video.composable.lut_accel
```

- 第三步：先实现rgb图与灰度图的转化

```
video_pipeline = [video_in_in, rgb2gray, video_in_out]

cpipe.compose(video_pipeline)

cpipe.graph
```

12

XILINX.

1. static ip调用

- 第四步：**替换**ip，将rgb2gray替换为rgb2hsv

```
cpipe.replace((rgb2gray, rgb2hsv))

cpipe.graph
```

- 第五步：**删除**ip，将rgb2hsv这个ip删除

```
cpipe.remove([rgb2hsv])

cpipe.graph
```

- 第六步：**插入**ip，将filter2d和gray2rgb插入，其中数字为插入该ip的位置

```
cpipe.insert(([filter2d, lut],1))

cpipe.graph
```

```
cpipe.insert(([gray2rgb],3))

cpipe.graph
```

- 第七步：停止pipeline，否则将会影响下次使用

```
video.stop()
```

13

XILINX.

动态 DFX Overlay 调用

下面是 DFX Overlay 的调用

2. DFX Overlay调用

- 目标：调用DFX区域的Overlay，并实现有分支（并行）的pipeline

- 第一步：下载Overlay

```
from pynq.lib.video import *
from composable_pipeline import ComposableOverlay
from composable_pipeline.libs import *

ol = ComposableOverlay("../overlay/cv_dfx_4_pr.bit")

cpipe = ol.composable

video = HDMIVideo(ol)
video.start()
```

2. DFX Overlay调用

- 第二步：载入函数dilate_accel和erode_accel的ip

```
cpipe.c_dict.loaded
cpipe.loadIP([cpipe.pr_1.dilate_accel])
cpipe.c_dict.loaded
```

- 第三步：搭建pipeline

```
video_in_in = cpipe.video.hdmi_in.color_convert
video_in_out = cpipe.video.hdmi_in.pixel_pack

dilate = cpipe.pr_1.dilate_accel
cpipe.compose([video_in_in, dilate, video_in_out])

cpipe.graph
```

2. DFX Overlay调用

- 第四步：搭建有分支的pipeline

```
cpipe.loadIP(['pr_fork/duplicate_accel', 'pr_join/subtract_accel', 'pr_0/filter2d_accel'])
filter2d = cpipe.video.composable.filter2d_accel
duplicate = cpipe.pr_fork.duplicate_accel
subtract = cpipe.pr_join.subtract_accel
fifo = cpipe.pr_0.axis_data_fifo_1
filter2d_d = cpipe.pr_0.filter2d_accel

filter2d.sigma = 0.3
filter2d.kernel_type = 'gaussian_blur'

filter2d_d.sigma = 12
filter2d_d.kernel_type = 'gaussian_blur'
video_pipeline = [video_in_in, filter2d, duplicate, [[filter2d_d], [1]], subtract, video_in_out]

cpipe.compose(video_pipeline)

cpipe.graph
```

- 第五步：停止pipeline，否则将会影响下次使用 `video.stop()`

使用 Python 调整 IP 参数

3. 用python调整IP的参数

- 用python构建可视化的组件调整IP参数

```
from ipywidgets import widgets, IntSlider, FloatSlider, interact

thr = IntSlider(min=0, max=255, step=1, value=20)
k_harris = FloatSlider(min=0, max=0.2, step=0.002, value=0.04, description='K')
```

```
algorithm = 'Fast'
def swap():
    global algorithm
    global thr
    if algorithm == 'Fast':
        cpipe.replace((fast, harr))
        algorithm = 'Harris'
        thr.max = 1024
        thr.value = 422
    else:
        cpipe.replace((harr, fast))
        algorithm = 'Fast'
        thr.max = 255
        thr.value = 20
```

```
def app(new_algorithm, threshold, k):
    global thr
    global k_harris
    if new_algorithm != algorithm:
        swap()
    elif new_algorithm == 'Fast':
        fast.threshold = threshold
        k_harris.disabled = True
    else:
        harr.threshold = threshold
        k_harris.disabled = False
        harr.k = k
```

```
interact(app, new_algorithm=['Fast','Harris'], threshold=thr, k=k_harris);
```

19

XILINX.

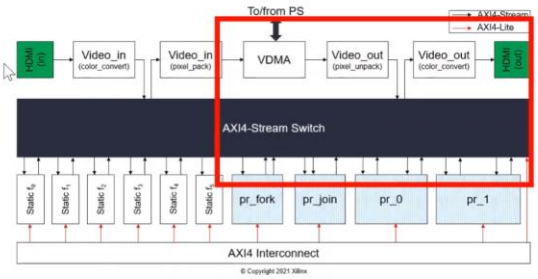
如上图所示，可以利用 Python 对算子的类型和参数阈值进行自定义，并设计一些可视化的组件。

以上是使用 HDMI 进行输入，下面是使用 Webcam 进行输入，也就是调用 VDMA 进行处理，从 video_out_in 输入，从 video_out_out 输出。

Webcam 输入（VDMA 的使用）

使用Webcam作为输入时的数据通路

- 从这个例子中可以发现，我们先把Webcam的输入加载到了板卡上的VDMA中，然后通过Video_out_in(pixel unpack)和Video_out_out(color convert)连接Composable Overlay，与前几个例子中在Video_in处连接pipeline不同。具体数据通路如下图所示：



22

XILINX.

Webcam作为输入

- 第一步：下载Overlay

```
from pynq.lib.video import *
from composable_pipeline import ComposableOverlay
from composable_pipeline.libs import *
from composable_pipeline.video import *
from ipywidgets import widgets, interact, FloatSlider, IntSlider

ol = ComposableOverlay("../overlay/cv_dfx_4_pr.bit")

cpipe = ol.composable
```


Webcam作为输入

- 第二步：搭建Webcam作为输入时的数据通路

```
video_in = Webcam()

hdmi_out = ol.video.hdmi_out
hdmi_out.configure(video_in.mode, PIXEL_BGR)

video_in.start()
hdmi_out.start();
video_in.tie(hdmi_out)
```


Webcam作为输入

- 第三步：搭建pipeline

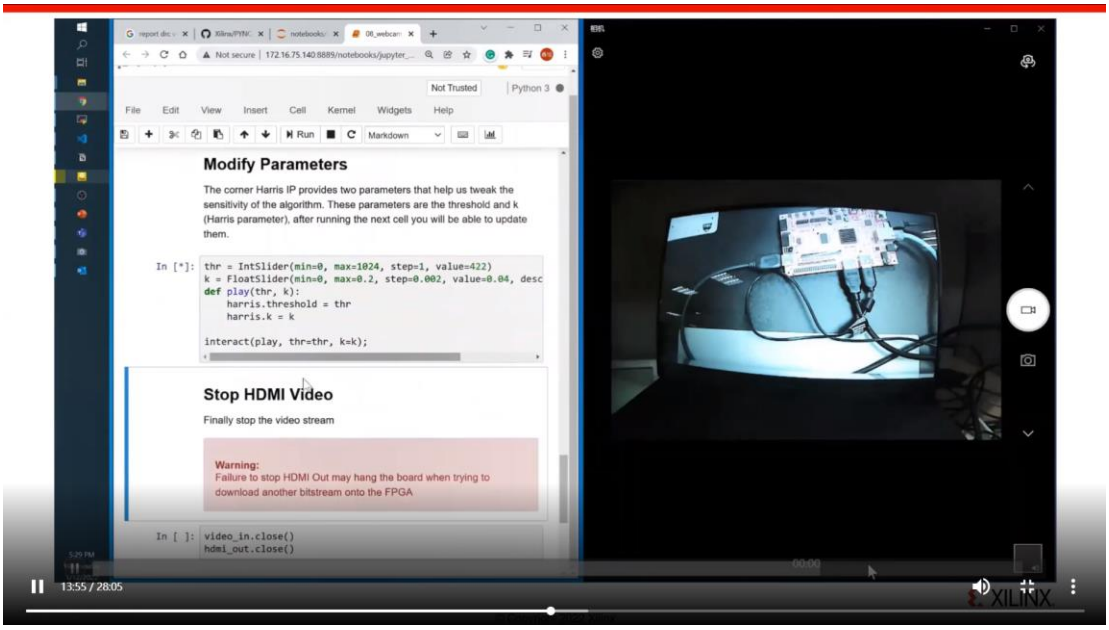
```
video_out_in = cpipe.video.hdmi_out.pixel_unpack
video_out_out =  cpipe.video.hdmi_out.color_convert
```

在载入算法将会用到的ip之前，注意要先将Webcam的输入暂停

```
video_in.pause()
cpipe.loadIP(['pr_1/cornerHarris_accel', 'pr_fork/duplicate_accel', 'pr_join/add_accel'])
```

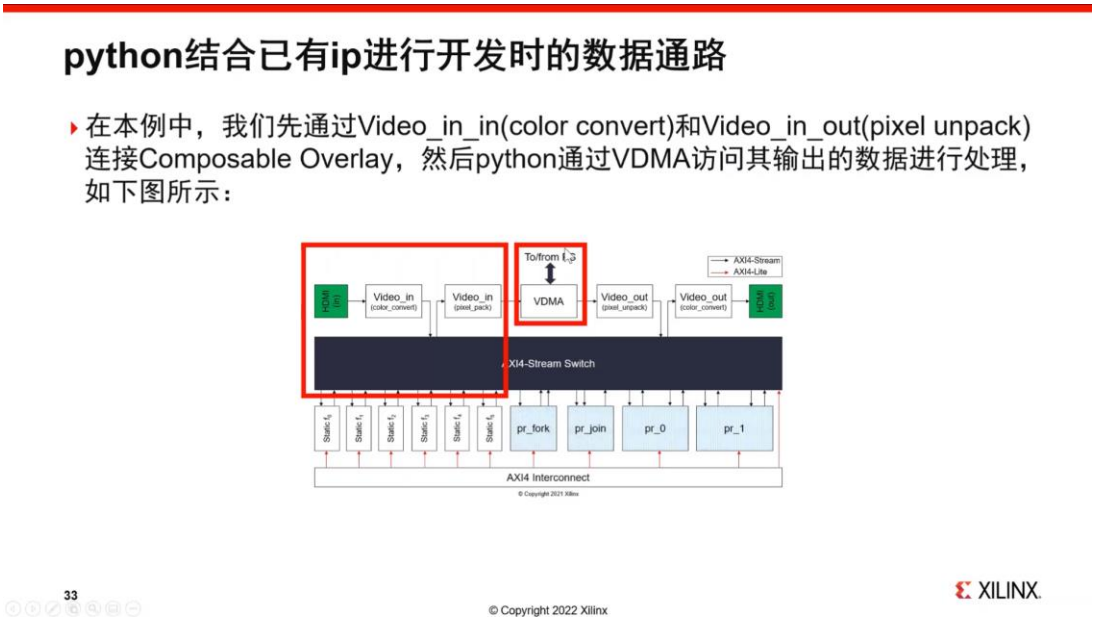
在load完成后，再重新启动

```
video_in.tie(hdmi_out)
harris = cpipe.pr_1.cornerHarris_accel
duplicate = cpipe.pr_fork.duplicate_accel
add = cpipe.pr_join.add_accel
r2g = cpipe.video.composable.rgb2gray_accel
g2r = cpipe.video.composable.gray2rgb_accel
video_pipeline = [video_out_in, duplicate, [[r2g, harris, g2r], [1]], add, video_out_out]

cpipe.compose(video_pipeline)

cpipe.graph
```



Python+Composable Pipeline 协同设计

小恐龙游戏完成自动目标检测和避障。



在 PS 侧，通过 VDMA 将数据导入，而后进行处理，前半段的 look up table 和膨胀侵蚀的 PL 操作，就是在原来的通路中完成的，而后再进入 video_in(pixel_pack)，再进入 python 中进行一些操作，例如 opencv 等，再回到原始路径中去，原始路径就是 video_out(pixel_unpack)之后的路径。

小恐龙外挂

- 第一步：下载Overlay

```
from composable_pipeline import ComposableOverlay
from time import sleep
import numpy as np
import matplotlib.pyplot as plt
import serial
```

```
ol = ComposableOverlay("./overlay/cv_dfx_4_pr.bit")
cpipe = ol.composable
```

小恐龙外挂

- 第二步：调用算法所需的ip

```
cpipe.loadIP([cpipe.pr_0.erode_accel, cpipe.pr_0.dilate_accel, cpipe.pr_1.erode_accel, cpipe.pr_1.dilate_accel])
```

- 第三步：搭建数据通路

```
hdmi_in = ol.video.hdmi_in
hdmi_out = ol.video.hdmi_out
hdmi_in.configure()
hdmi_out.configure(hdmi_in.mode)
print(hdmi_in.mode)
print(hdmi_out.mode)
hdmi_in.start()
hdmi_out.start()
hdmi_in.tie(hdmi_out)
```

```
VideoMode: width=1280 height=720 bpp=24
VideoMode: width=1280 height=720 bpp=24
```


载入 overlay，而后搭建数据通路，调用膨胀和侵蚀的 IP，而后进行一些配置，如 1280*720 的配置。

小恐龙外挂

- 第四步：搭建pipeline

```
video_in_in = cpipe.video.hdmi_in.color_convert
video_in_out = cpipe.video.hdmi_in.pixel_pack
lut = cpipe.video.composable.lut_accel
erode_0 = cpipe.pr_0.erode_accel
dilate_0 = cpipe.pr_0.dilate_accel
erode_1 = cpipe.pr_1.erode_accel
dilate_1 = cpipe.pr_1.dilate_accel
```

```
lut.kernel_type = 'binary_threshold'
```

```
cpipe.compose([video_in_in, lut, dilate_0, dilate_1, erode_0, erode_1, video_in_out])
```


先把需要用到的算子进行实例化，而后设置 look up table 的类型，而后用 cpipe.compose 搭建一个通路，里面是膨胀侵蚀等操作和获得一个 LUT 查找表。以上就完成了 PL 部分的搭建，也就是把 composable pipeline 通路搭建完毕了。

下面是 PS 侧功能，也就是判断黑色障碍物

小恐龙外挂

- 第五步：使用python对composable pipeline的输出图像进行进一步处理，根据处理后的图像判断小恐龙是否应该起跳

```
class DinoBot:
    def __init__(self, hdmi_in):
        self.hdmi_in = hdmi_in
        self.serial = serial.Serial('/dev/ttyUSB0', 115200)
        if self.serial.isOpen():
            print("serial open success")
        else:
            print("serial open failed")
    def start(self):
        self.serial.write(b'press')
        sleep(0.1)
        self.serial.write(b'release')
        while(True):
            try:
                frame = self.hdmi_in.readframe()
                img_slice = frame[330:450, 320:350]
                if((img_slice==255).all()):
                    self.serial.write(b'release')
                else:
                    self.serial.write(b'press')
            except KeyboardInterrupt:
                break
```

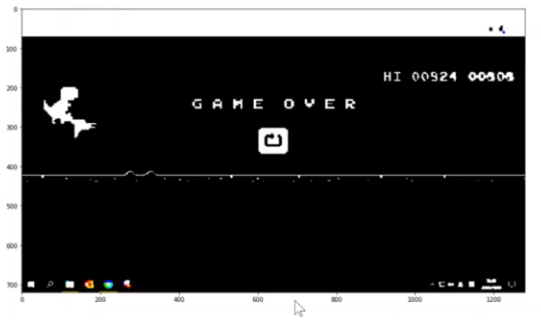

首先用到 hdmi.readframe 封装来读取一帧画面，在 Python 程序的 frame 中，就是选出一个框范围，截图出一个 slice 出来，通过串口输出一些空格键的指令。判断框内是否为黑色，如果是，则跳跃，否则是正常行走。

下面就是一个最终的结果了：

小恐龙外挂

- 第六步：启动pipeline

```
In [9]: dino_bot = DinoBot(hdmi_in)
        serial open success

In [ ]: dino_bot.start()

In [11]: frame = hdmi_in.readframe()
         plt.figure(figsize=(15, 15))
         plt.imshow(frame)

Out[11]: <matplotlib.image.AxesImage at 0xad8d2a50>
```

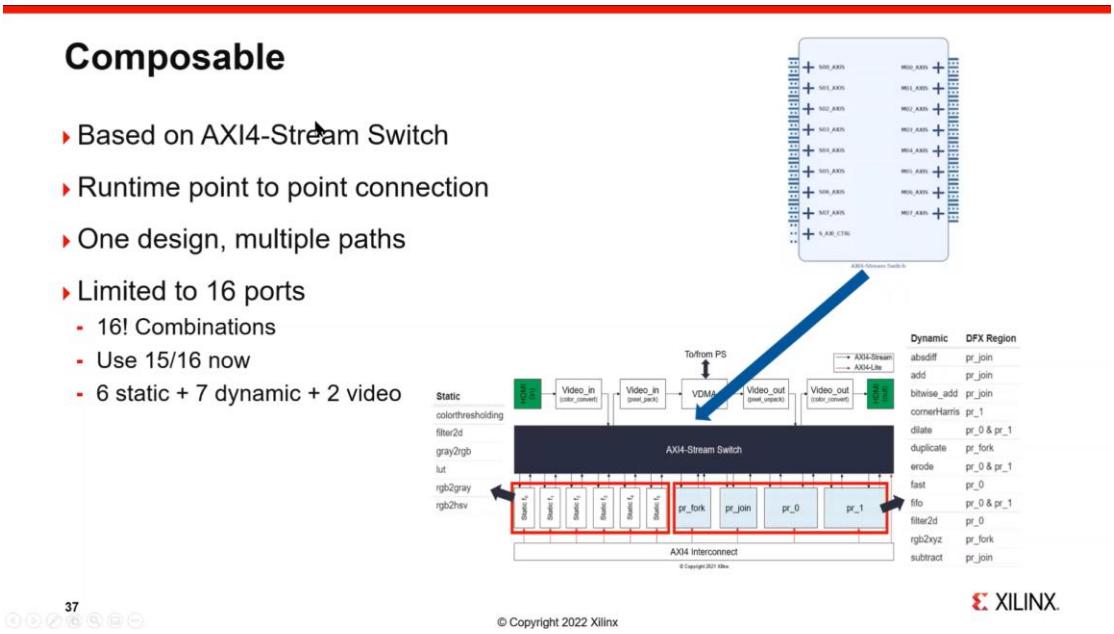


- 第七步：停止pipeline，否则将会影响下次使用

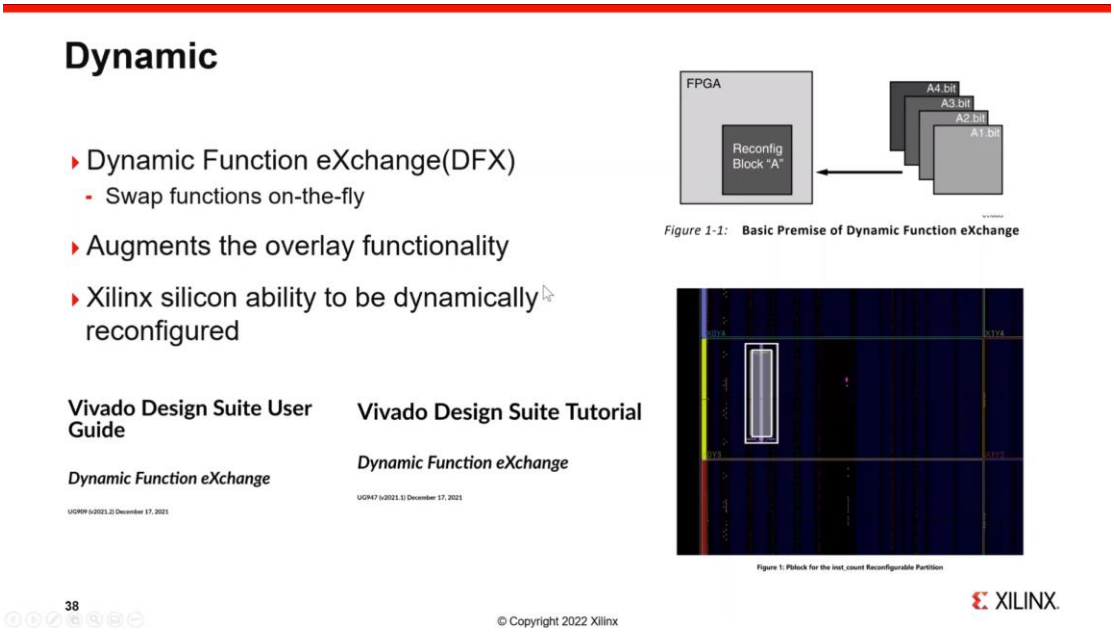
```
hdmi_in.close()
hdmi_out.close()
```


还有一些有趣的应用，比如机械臂识别抓取物品。

Composable Pipeline 结构



Composable 就是可组合的意思，核心就是 AXI4-stream，支持 16 输入 16 输出的接口的配置，所以就有 16 中配置的方式，可以从任何一个端口连接到对面的端口去，并且是可以在运行时进行配置。因此我们可以将这些 IP 进行动态的调整。

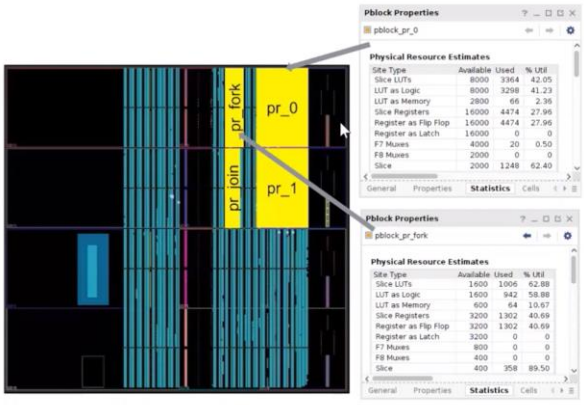


另外一个重要的功能就是 DFX（动态函数交换），可以动态配置 IP，节省 FPGA 资源和空间，在结构图中的右下角部分就是 DFX 部分，里面有很多的 IP，在不用的时候配置在里面会占用 FPGA 资源，需要的时候再下载入 DFX 中即可。

优点是，当 FPGA 片上资源不够的时候，先把一些小的比特流文件下到设计中去，做一个运行时的硬件级别的重构。

The composable Pipeline

Static	Dynamic	pr_0	pr_1	pr_fork	pr_join
colorThresh	absdiff				✓
filter2d	add				✓
gray2rgb	bitwise_and				✓
lut	cornerHarris		✓		
rgb2gray	dilate	✓	✓		
rgb2hsv	duplicate			✓	
	erode	✓	✓		
	fast	✓			
	fifo	✓	✓		
	filter2d	✓			
	rgb2xyz			✓	
	subtract				✓



函数的PR占用情况　　所指定的4个DFX区域

1x full bitstream and 11x partial bitstreams

XILINX.

上图是 composable pipeline 在 vivado device 的一个界面，图中可以看出他分出来 4 个动态可重构区域，就是图中黄色的 4 个 pr 区域，这 4 个区域具有 DFX 的性质

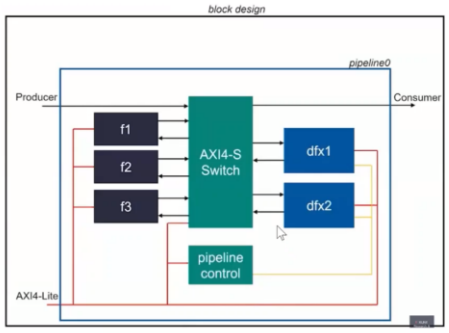
在上图中同样可以看出，在 pr_0 中已经配置好了资源，比如大的设计分配了 8000 个 look up table，小的设计就分配 1600 个 LUT。而我们常用的这些函数就可以占据这些区域进行动态配置。

上图中的左侧函数部分，当我们需要使用这些函数的时候，将这个函数编成一个 partial 的比特流下载到对应的 pr 区去，而不会对外面的这些设计产生影响。

最终完整的 composable pipeline 设计就是 1 个完整的比特流文件，并含有 11 个 partial 比特流文件（图中左侧的函数的比特流文件）。

Block Design

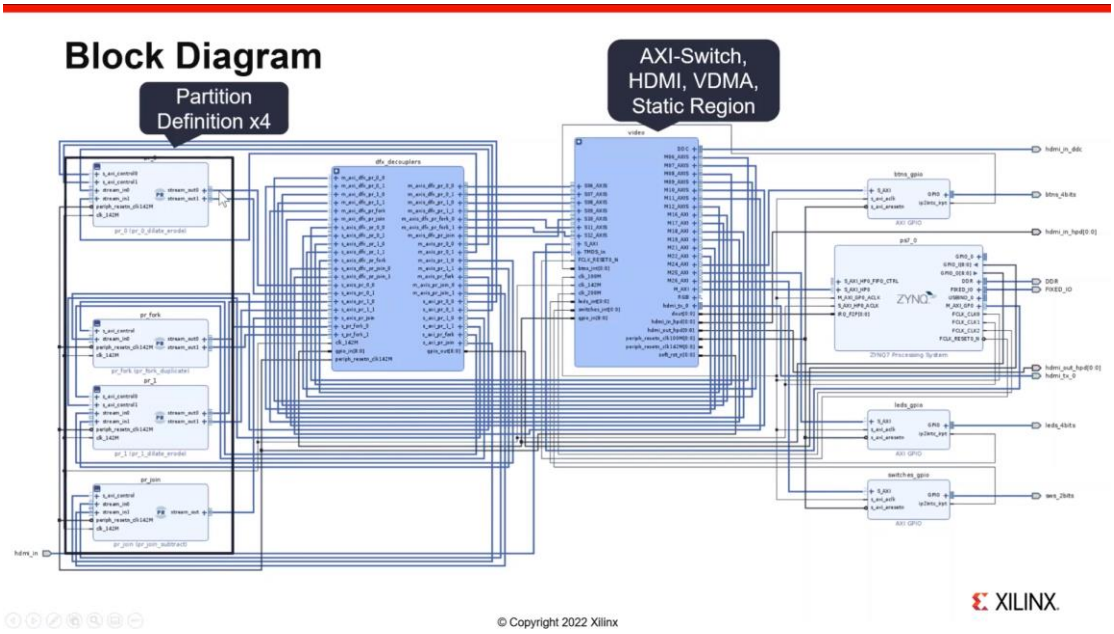
- AXI4-Stream Switch
- Pipeline Control
  - Soft reset
  - DFX control
- Static Functions
- DFX Regions
  - Add reconfigurable modules



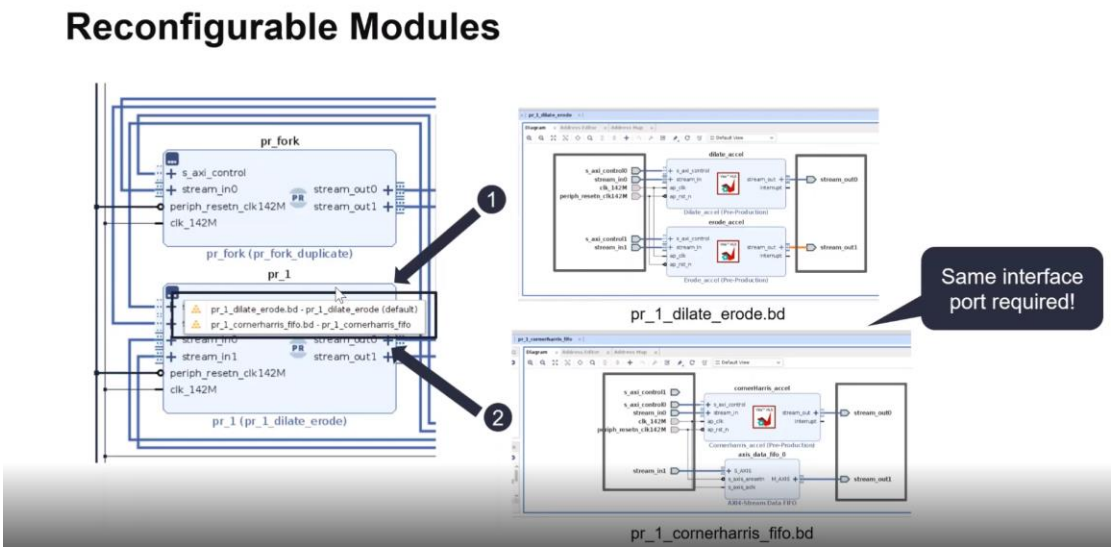
XILINX.

上图就是 composable pipeline 的一个整体设计情况，有静态区域和动态区域，也有 pipeline control 的函数，用胶水逻辑来粘合整个工程。

从 block design 的角度来看的话，如下图：

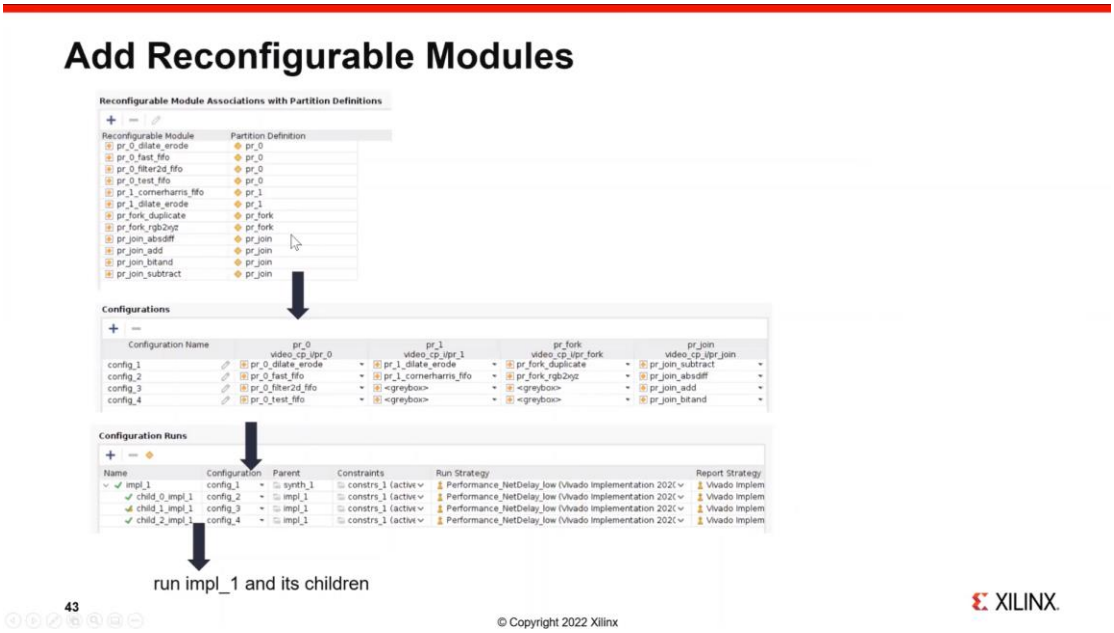


最左边就是 DFX 区域 4 个，是动态可重构的。其余部分，都是静态的区域。下面来具体讲一下 DFX 的配置，如下图，拿 pr_fork 和 pr_1 进行举例。



比如希望 pr_1 支持两个算子，但是可能放不下，此时可以设置 DFX 属性，在 Block Diagram 中进行设置。点开可以看见这两个例子一个是膨胀侵蚀和 harris，看起来差不多，接口也完全相同，都是 AXI_Control 和 stream 的接口，可以看 Harris 中，由于有一个 FIFO 所以 AXI_Control 没有用到被悬空。

所以我们在进行动态设计的时候，只需要保证接口区域完全一致，在 Diagram 内部怎样改动都是可以的，只要资源可以承载的下。



如上图所示，在加载这些函数的时候，我们就可以去定义一下 pr_0、pr_1、pr_fork、pr_join 上的每一个函数。配置在 configuration 里面。可以分成几个 implement 去实现，然后生成比特流。

参考资料及链接

Composable Pipeline 简介

https://mp.weixin.qq.com/mp/homepage?__biz=MzU0Mzk1MzMyNQ==&hid=9&sn=5d2ac6e24b255385b9062628824f5383&scene=18

官方参考资料

1. PYNQ 官方社区：http://www.pynq.io/community.html
2. PYNQ 官方文档：https://pynq.readthedocs.io/en/latest/index.html
3. Xilinx 中文学习资料与开源设计：https://github.com/xupsh
4. PYNQ 开源社区微信公众号的菜单栏索引，包含丰富的项目案例与分享。

Composable Pipeline 的 GitHub

https://github.com/Xilinx/PYNQ_Composable_Pipeline