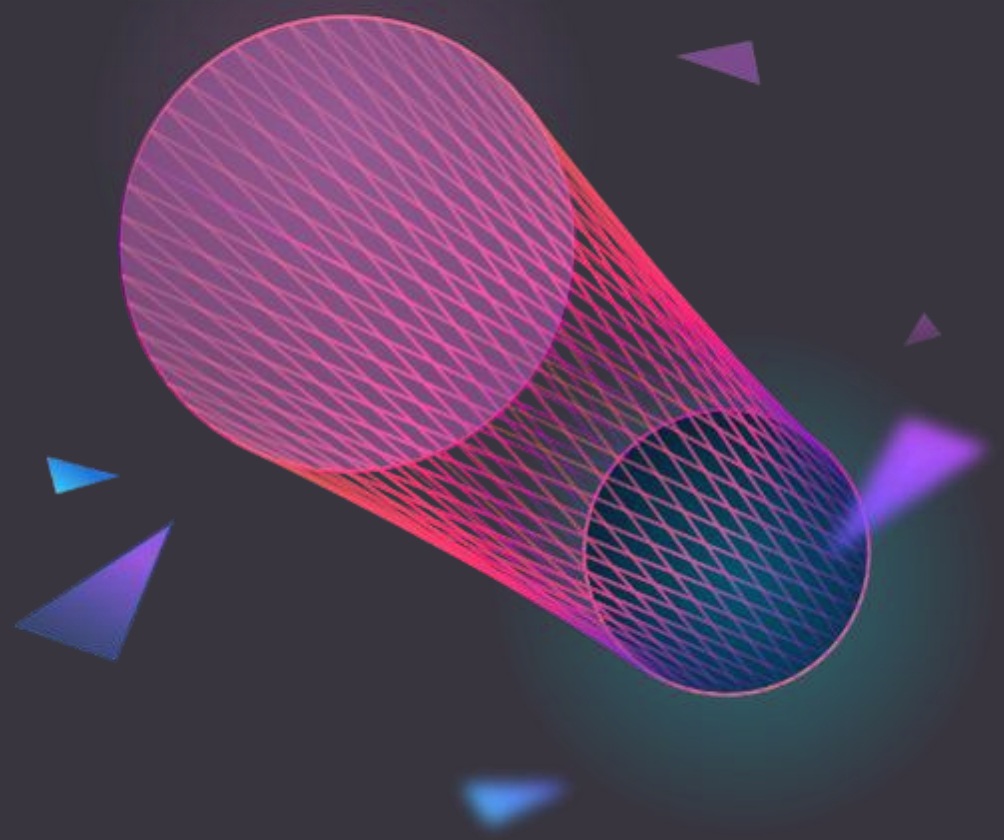
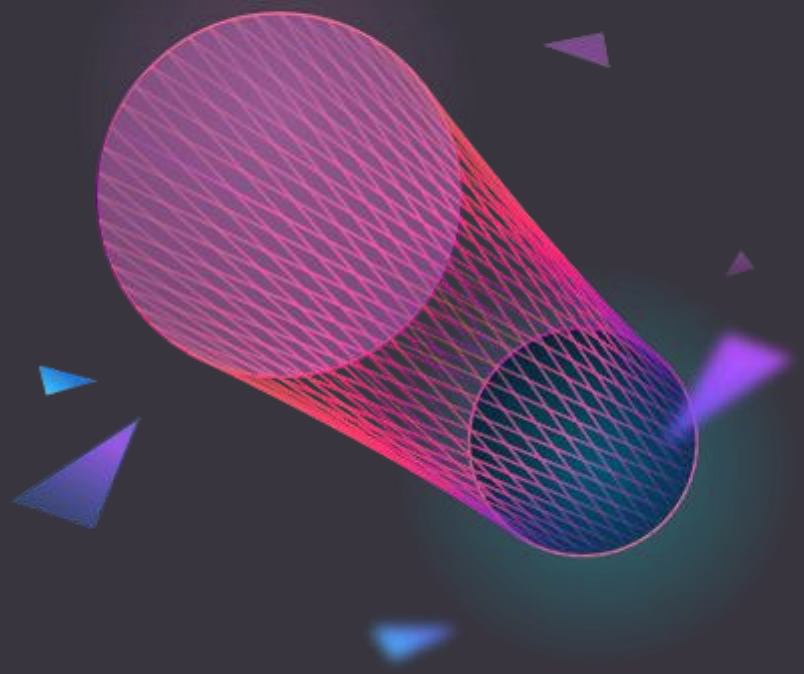


HLS for Algorithm acceleration

2019.1.14

Kang li



Outline

PART ONE

01

Introduction of HLS

xilinx Vivado HLS (High-level synthesis)

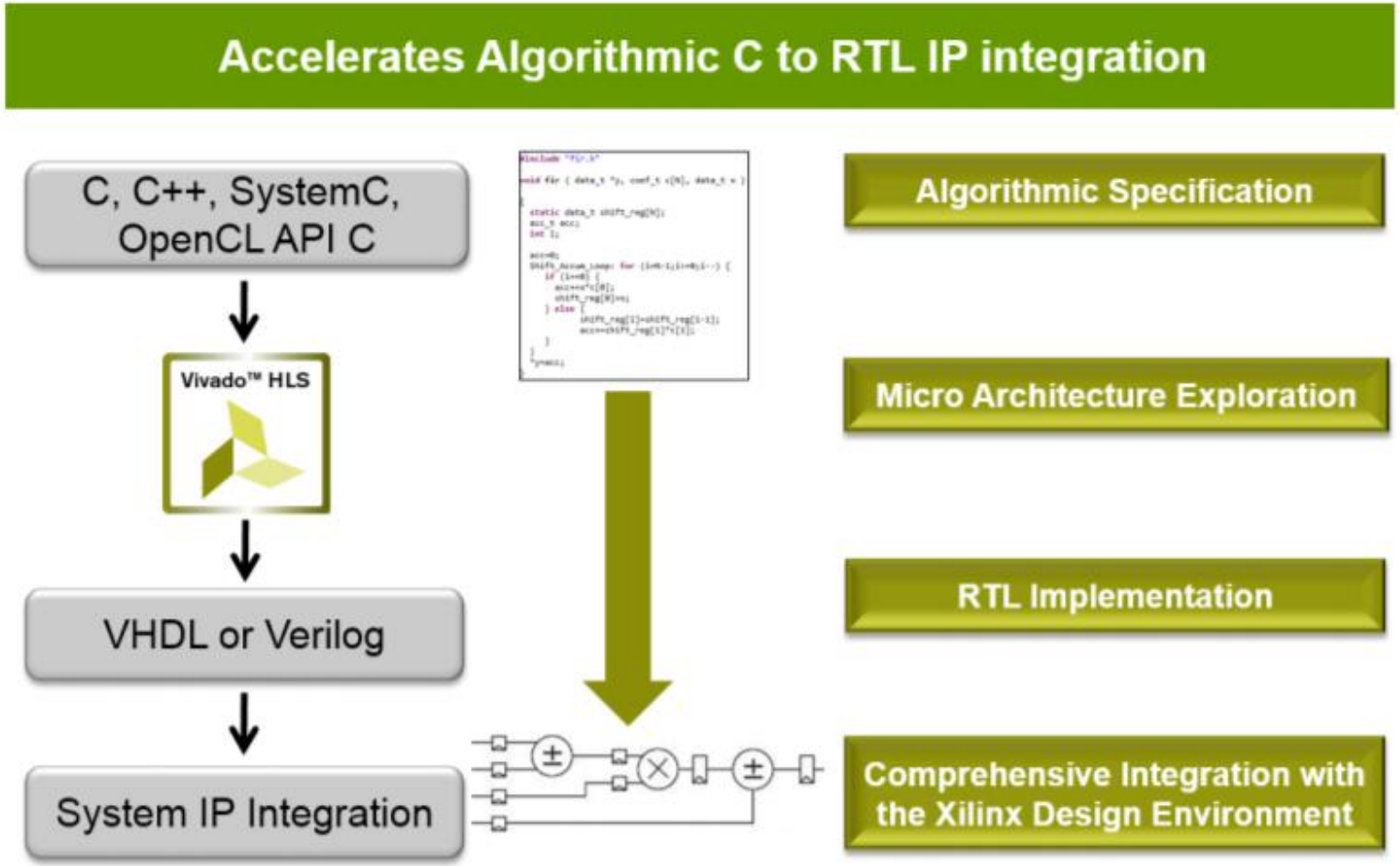
High-Level Synthesis

High-Level Synthesis

- Creates an **RTL** implementation from **C, C++, System C, OpenCL API C** kernel code
- Extracts control and dataflow from the source code
- Implements the design based on defaults and user applied directives

What should be the focus for Algorithm acceleration ?

- **Resources, speed, power consumption**
- Smaller designs, faster designs, optimal designs



Design Exploration with Directives

One body of code:
Many hardware outcomes

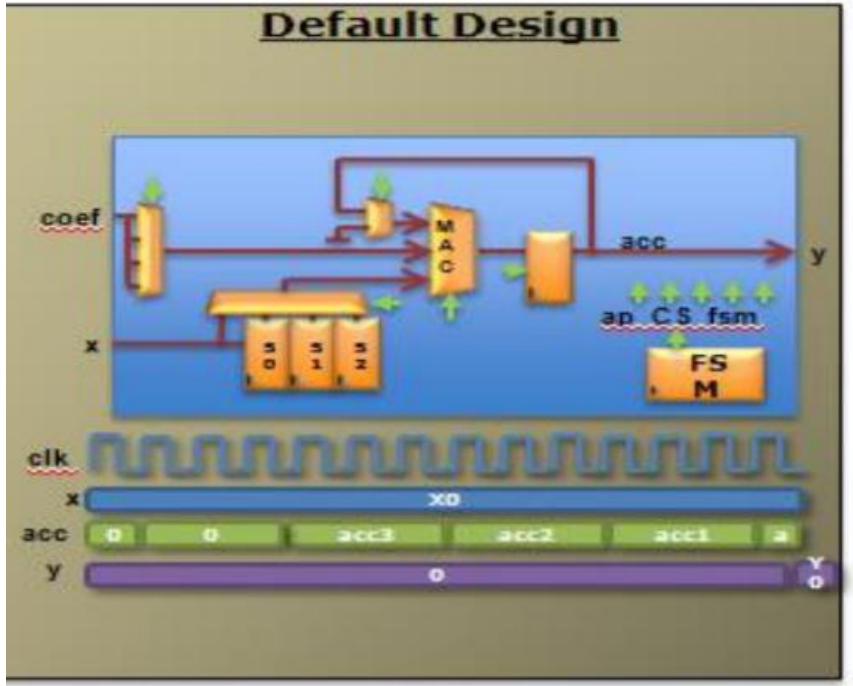
```
...
loop: for (i=3;i>=0;i--) {
  if (i==0) {
    acc+=x*c[0];
    shift_reg[0]=x;
  } else {
    shift_reg[i]=shift_reg[i-1];
    acc+=shift_reg[i]*c[i];
  }
}
....
```

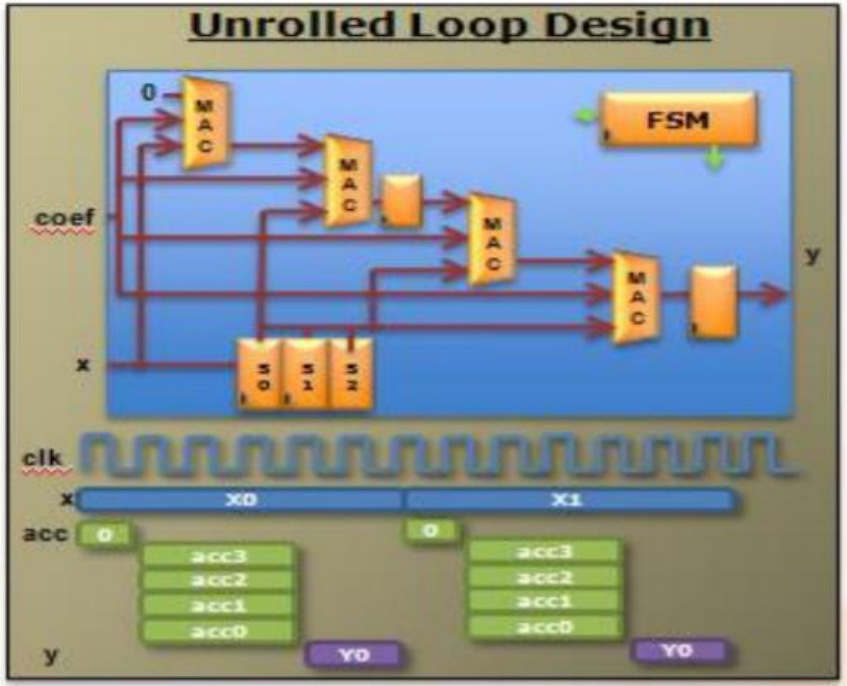
Before we get into details, let's look under the hood ....

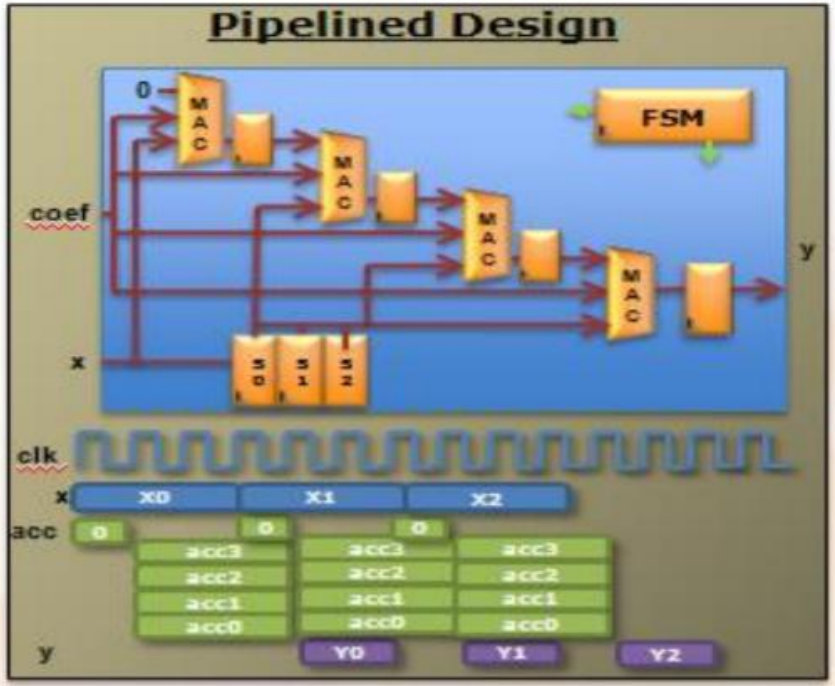
The same hardware is used for each iteration of the loop:
- **Small area**
- **Long latency**
- **Low throughput**

Different hardware is used for each iteration of the loop:
- **Higher area**
- **Short latency**
- **Better throughput**

Different iterations are executed concurrently:
- **Higher area**
- **Short latency**
- **Best throughput**







Library Support

Floating-point support

- Support for **single-precision** and **double-precision**, **floating-point** functions from math.h

Fixed-point support

- Simulate and implement fixed-point algorithms using the <**ap_int.h**> library

OpenCV video function support

- Enable migration of **OpenCV** designs into Zynq® All Programmable SoC
- Libraries target real-time full HD video processing

DSP function support

- Instantiate and parameterize FIR compiler and FFT LogiCORE™ IP as function calls from your C++ code

Unsupported Constructs: Overview

System calls and function pointers

- Dynamic memory allocation
  malloc() and free()
- Standard I/0 and file I/O operations
  fprintf() / fscanf(), etc.
- System calls
  time(), sleep(), etc.

Non-standard pointers

- Pointer casting between general data types

--OK with native integers types

- If a double pointer is used in multiple functions, Vivado HLS tool will inline all the functions

--Slower synthesis, may increase area and run time

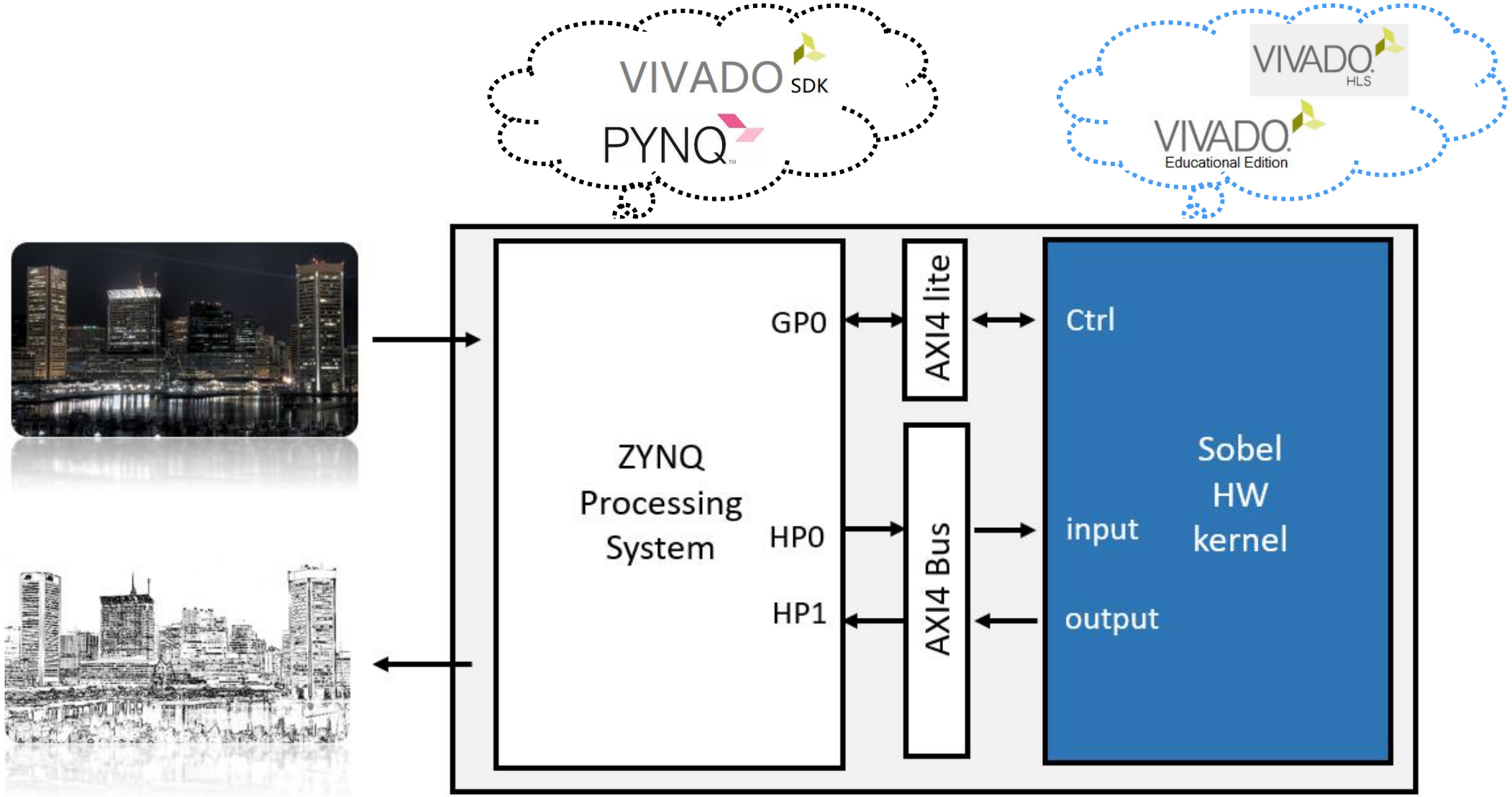
PART ONE

02

Design flow of Vivado HLS

Sobel SW/HW Design

Block diagram of the sobel filter



Design flow of HLS

C Simulation

C Synthesis

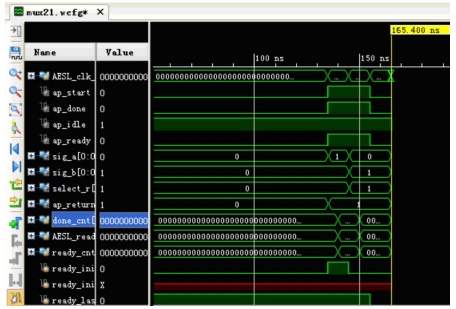
C/RTL Cosimulation

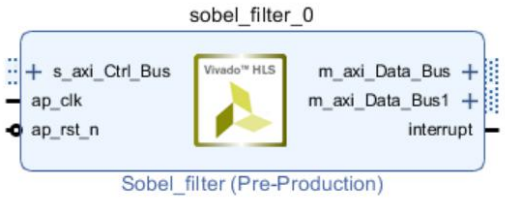
Export RTL

```
void main  ()
  {
    input data();
    hls function();
    output data();
  }
```

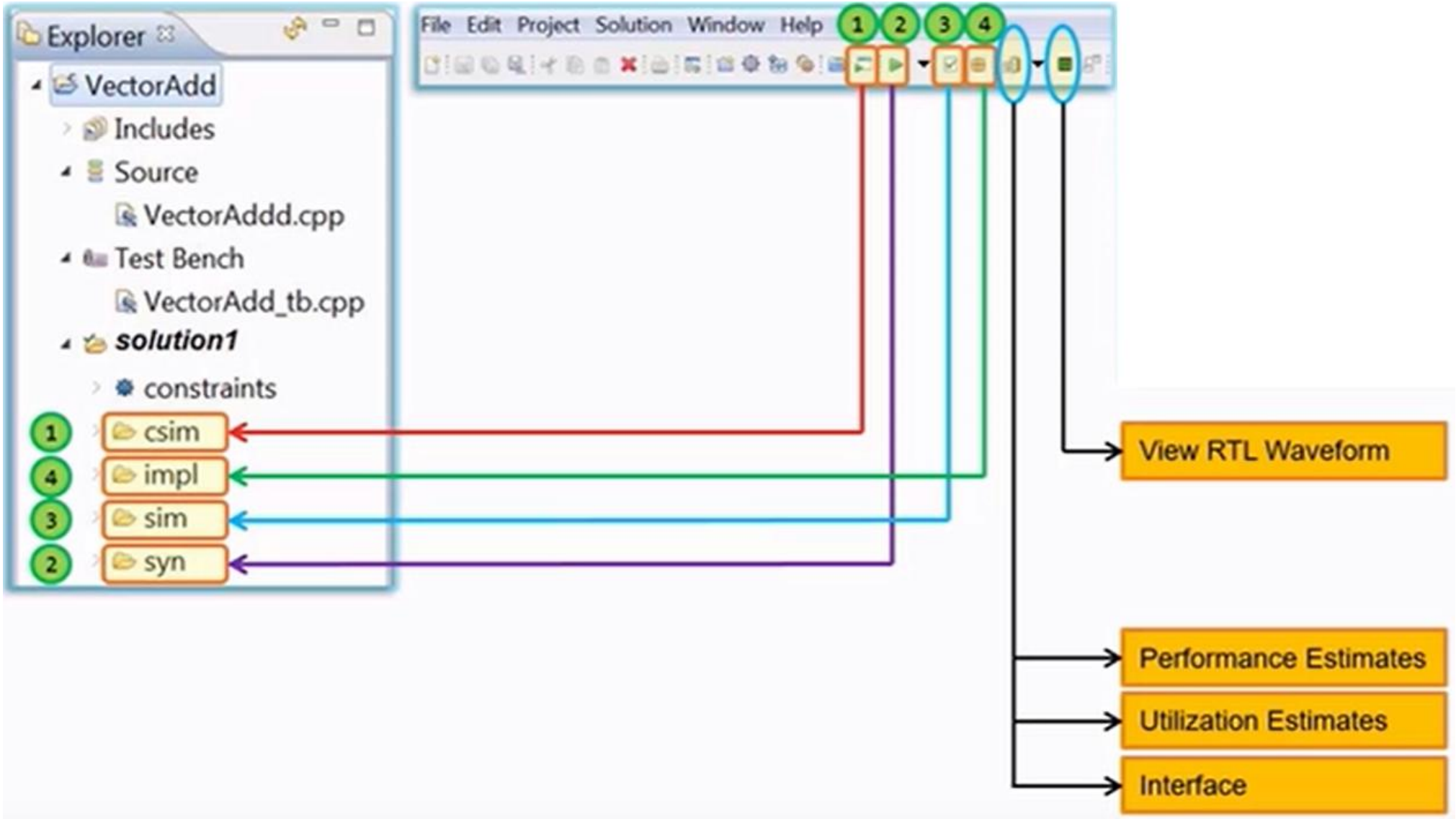
C CODE

RTL Code

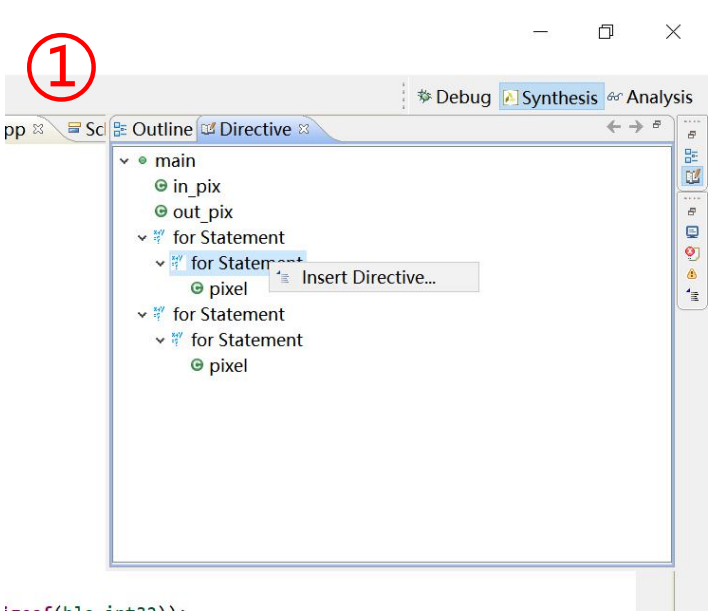


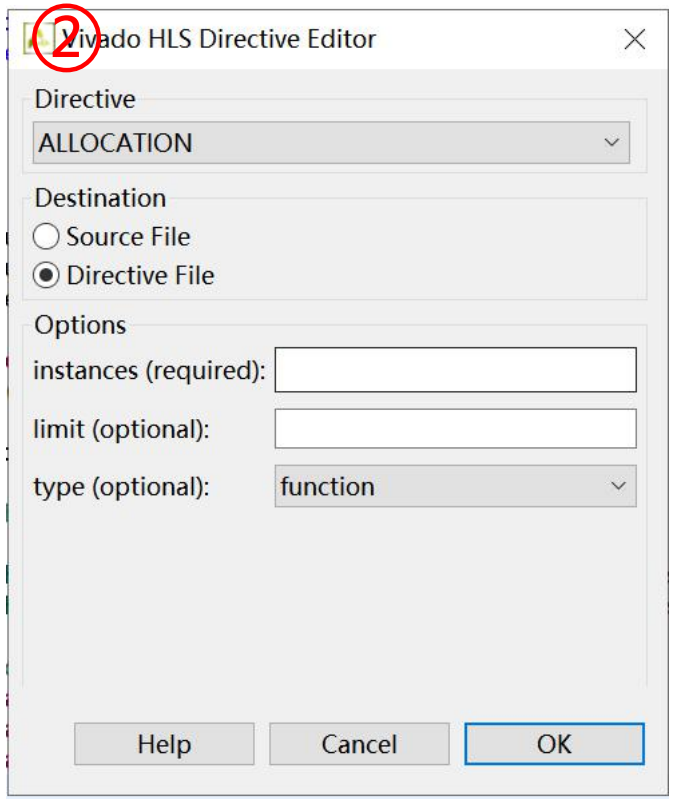


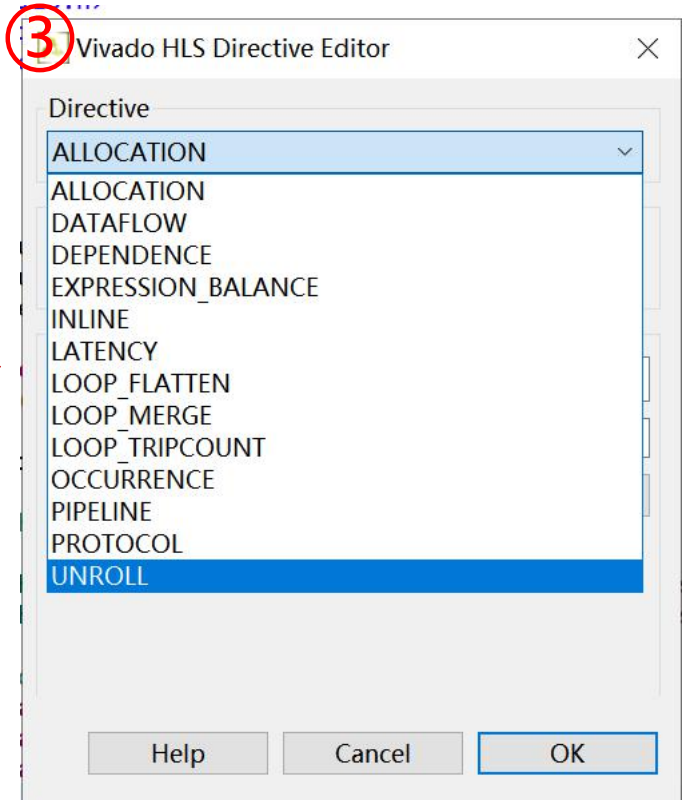
Design flow of HLS



Add HLS Directive







HLS Performance Estimates

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.283	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2084403	2319284441523	2084403	2319284441523	none

Detail

Instance

N/A

Loop

Loop Name	Latency min	Latency max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
- sobel_filter_label0	2084400	2319284441520	1930 ~ 2147485594	-	-	1080	no
+ memcpy.burst_buff_in.V.addr.in_pix.V	0	1073741824	3	1	1	0 ~ 1073741823	yes
+ sobel_filter_label0.2	1925	1925	7	1	1	1920	yes
+ memcpy.out_pix.V.burst_buff_out.V.addr	0	1073741824	3	1	1	0 ~ 1073741823	yes

Time = Latency×clock(ns)

Resource utilization Estimates&Driver

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	2	-	-
Expression	-	3	0	1232
FIFO	-	-	-	-
Instance	4	-	1250	1520
Memory	10	-	0	0
Multiplexer	-	-	-	336
Register	0	-	1855	224
Total	14	5	3105	3312
Available	280	220	106400	53200
Utilization (%)	5	2	2	6

Detail

- Instance
- DSP48
- Memory
- FIFO
- Expression
- Multiplexer
- Register

```
// Ctrl_Bus
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/COH)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read)
//        bit 7  - auto_restart (Read/Write)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
//        others - reserved
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
//        others - reserved
```

```
#define XSOBEL_FILTER_CTRL_BUS_ADDR_AP_CTRL            0x00
#define XSOBEL_FILTER_CTRL_BUS_ADDR_GIE                0x04
#define XSOBEL_FILTER_CTRL_BUS_ADDR_IER                0x08
#define XSOBEL_FILTER_CTRL_BUS_ADDR_ISR                0x0c
#define XSOBEL_FILTER_CTRL_BUS_ADDR_BYTE_RDOFFSET_DATA 0x14
#define XSOBEL_FILTER_CTRL_BUS_BITS_BYTE_RDOFFSET_DATA 32
#define XSOBEL_FILTER_CTRL_BUS_ADDR_BYTE_WROFFSET_DATA 0x1c
#define XSOBEL_FILTER_CTRL_BUS_BITS_BYTE_WROFFSET_DATA 32
#define XSOBEL_FILTER_CTRL_BUS_ADDR_ROWS_DATA          0x24
#define XSOBEL_FILTER_CTRL_BUS_BITS_ROWS_DATA          32
#define XSOBEL_FILTER_CTRL_BUS_ADDR_COLS_DATA          0x2c
#define XSOBEL_FILTER_CTRL_BUS_BITS_COLS_DATA          32
#define XSOBEL_FILTER_CTRL_BUS_ADDR_STRIDE_DATA        0x34
#define XSOBEL_FILTER_CTRL_BUS_BITS_STRIDE_DATA        32
```

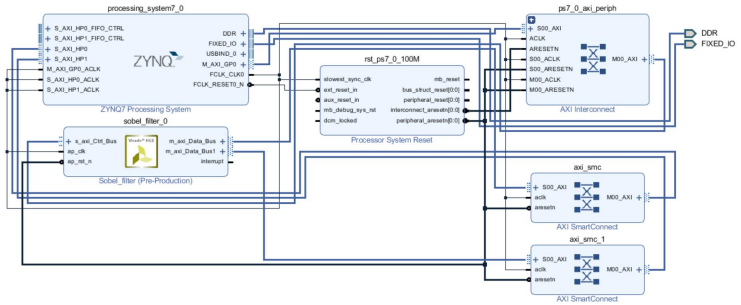
Vivado Design flow

Block design

Generate output

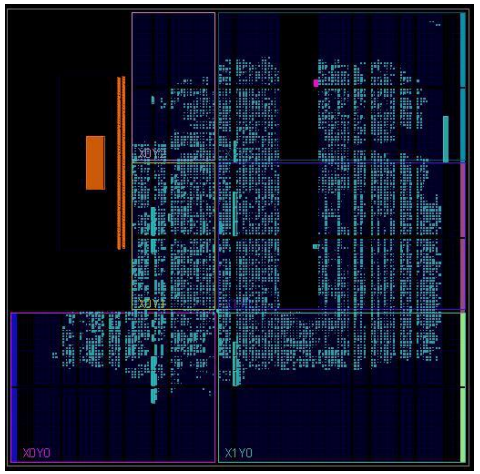
Synthesis& implement

Generate bitstream



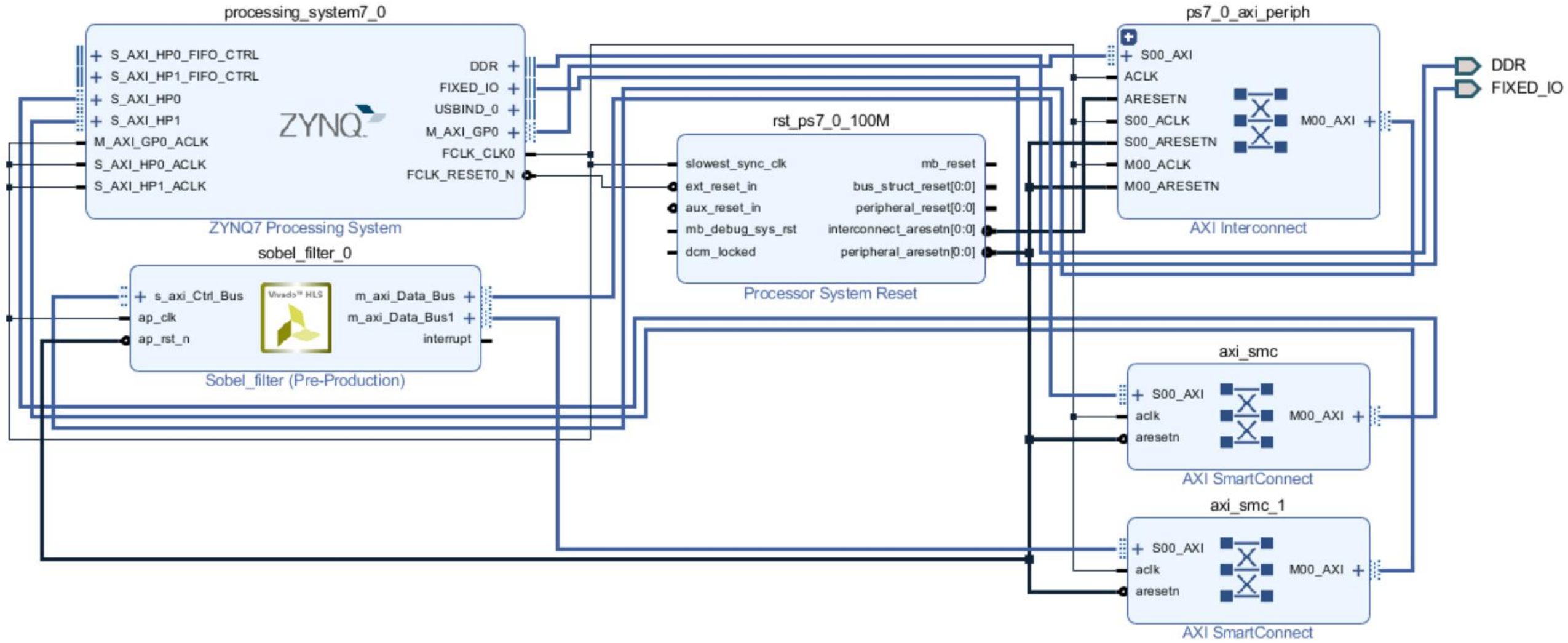
BD

RTL Code



sobel.bit
sobel.tcl

Vivado block design



FPGA part or board selection

New Project

Default Part

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

Parts | Boards

Reset All Filters

Category: All Package: clg400 Temperature: All Remaining

Family: Zynq-7000 Speed: -1

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7z007sclg400-1	400	100	14400	28800	50	0	66	0
xc7z010clg400-1	400	100	17600	35200	60	0	80	0
xc7z014sclg400-1	400	125	40600	81200	107	0	170	0
xc7z020clg400-1	400	125	53200	106400	140	0	220	0

New Project

Default Part

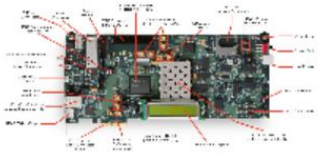
Choose a default Xilinx part or board for your project. This can be changed later.

Parts | **Boards**

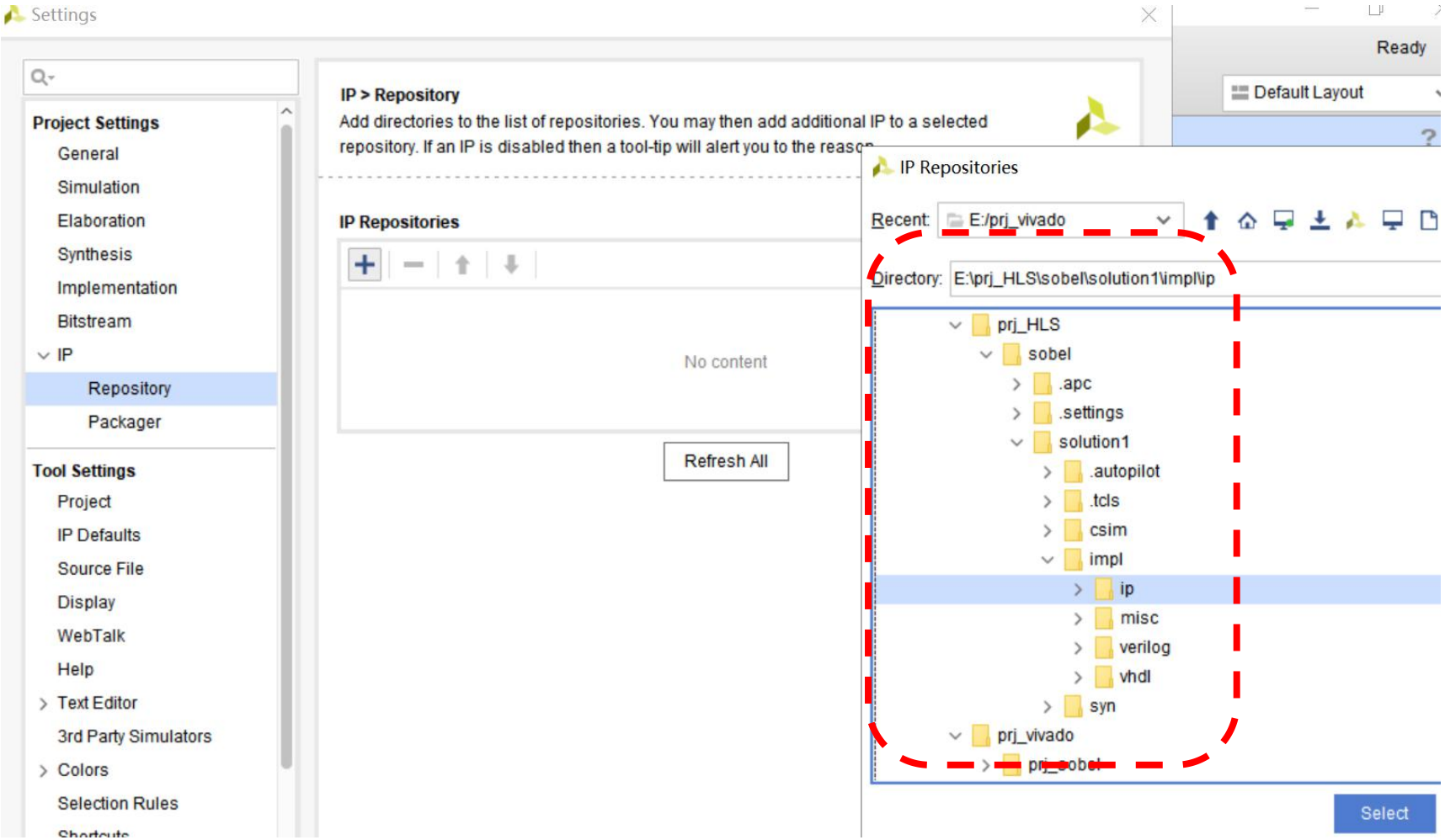
Reset All Filters

Vendor: All Name: All

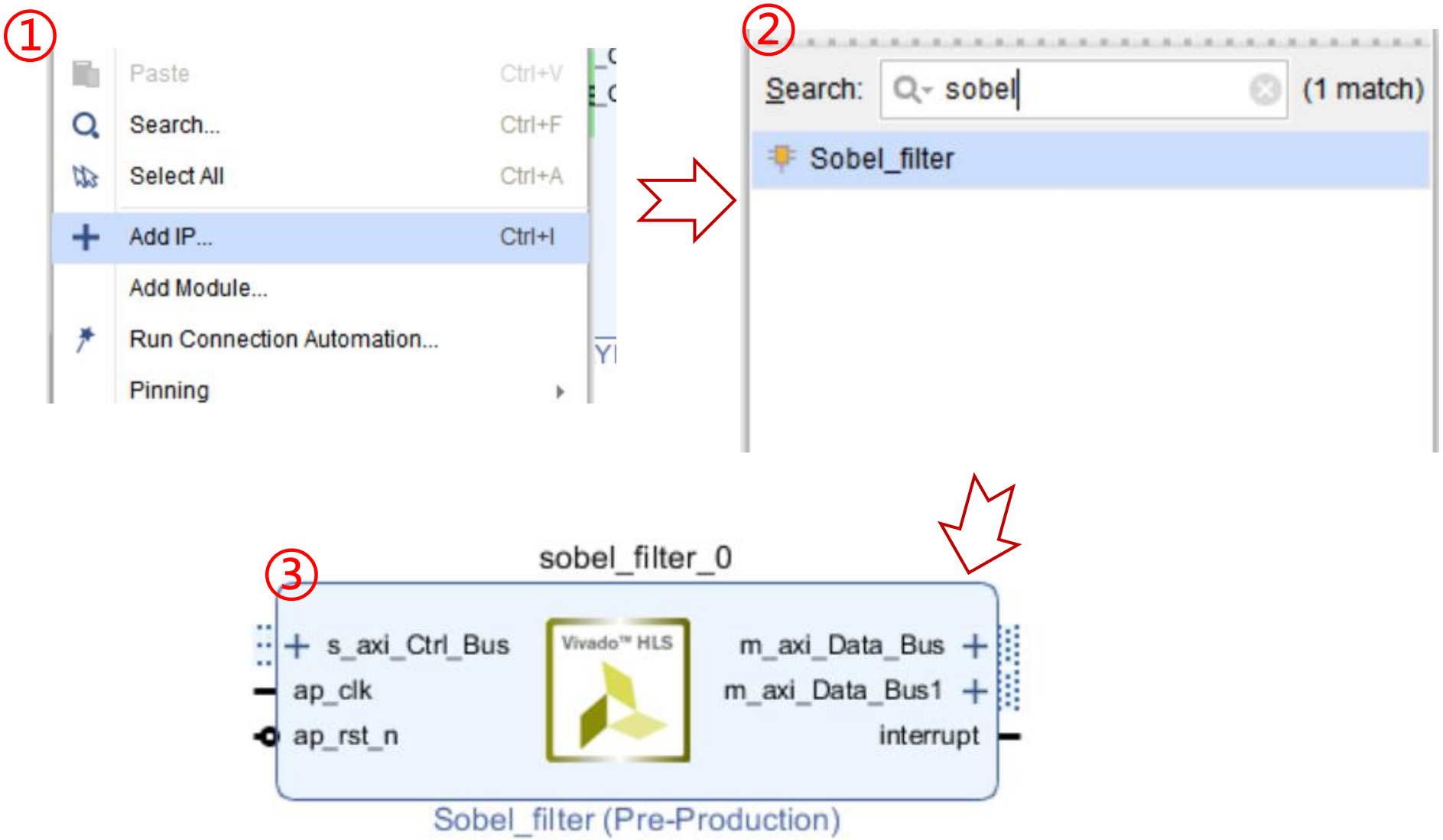
Search:

Display Name	Preview	Vendor	Fil
pynq-z2		tul.com.tw	1.0
Artix-7 AC701 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.4

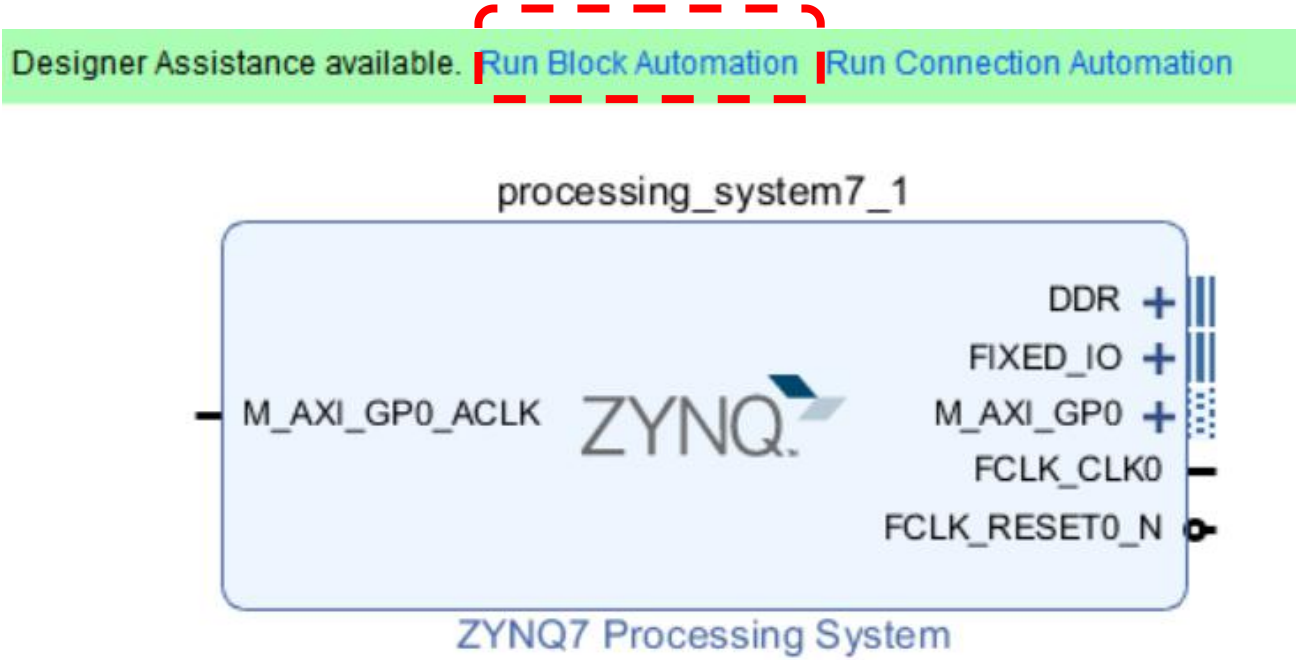
Add IP core --1


Settings
Project Settings
General
Simulation
Elaboration
Synthesis
Implementation
Bitstream
IP
Repository
Packager
Tool Settings
Project
IP Defaults
Source File
Display
WebTalk
Help
Text Editor
3rd Party Simulators
Colors
Selection Rules
IP > Repository
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason
IP Repositories
No content
Refresh All
Ready
Default Layout
IP Repositories
Recent: E:/prj_vivado
Directory: E:\prj_HLS\sobel\solution1\impl\ip
prj_HLS
sobel
.apc
.settings
solution1
.autopilot
.tcls
csim
impl
ip
misc
verilog
vhdl
syn
prj_vivado
prj_sobel
Select

Add IP core --2

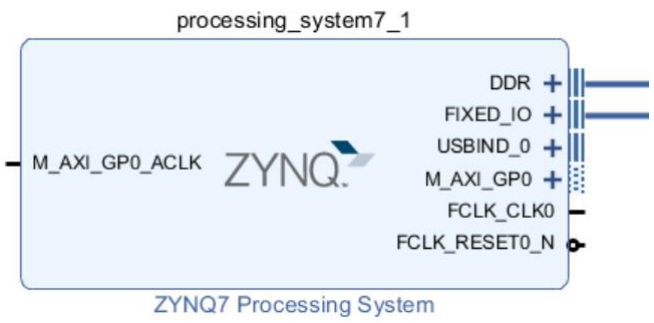

①
Paste Ctrl+V
Search... Ctrl+F
Select All Ctrl+A
Add IP... Ctrl+I
Add Module...
Run Connection Automation...
Pinning
②
Search: sobel
(1 match)
Sobel_filter
③
sobel_filter_0
s_axi_Ctrl_Bus
ap_clk
ap_rst_n
Vivado™ HLS
m_axi_Data_Bus
m_axi_Data_Bus1
interrupt
Sobel_filter (Pre-Production)

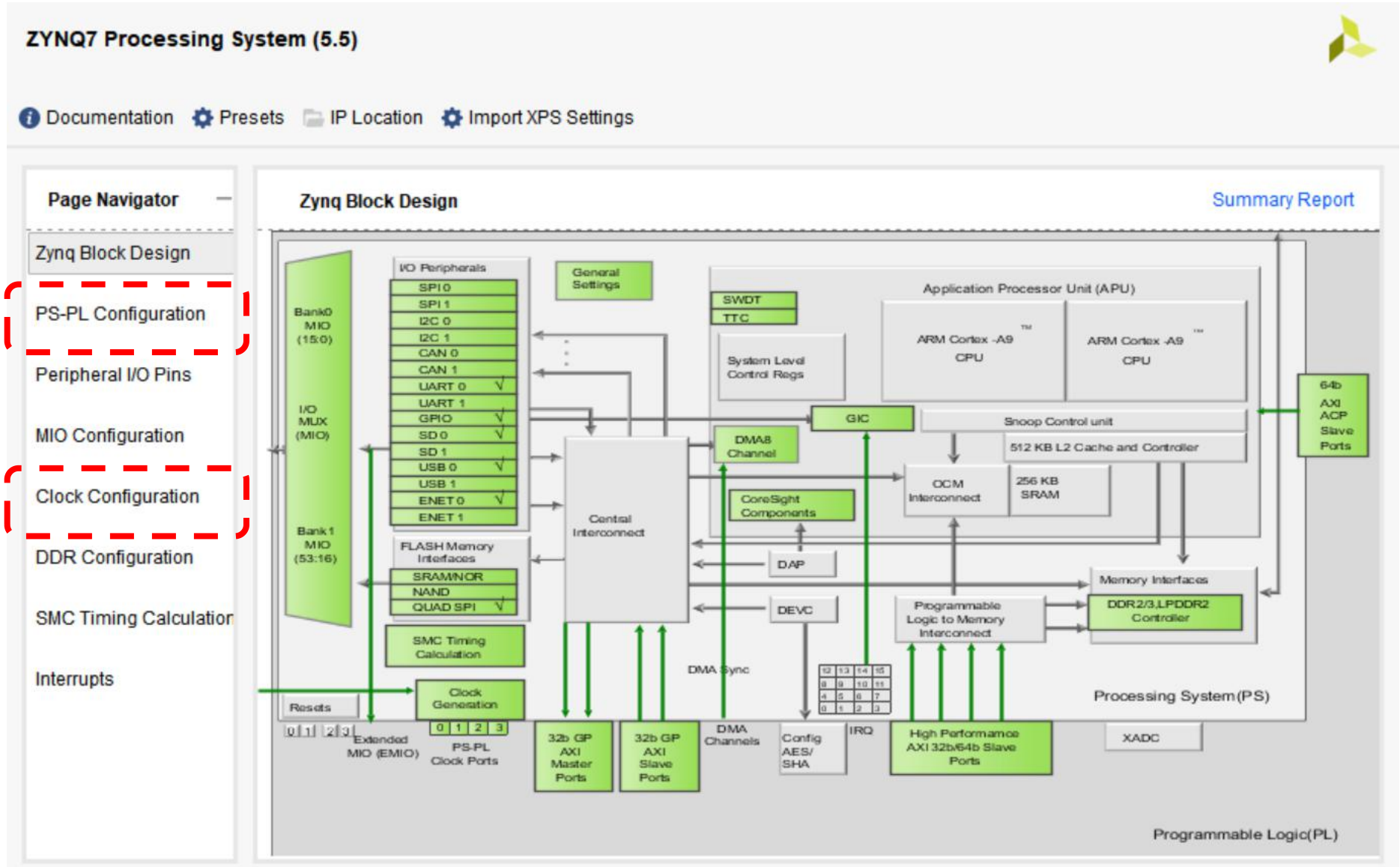
Vivado block design



① Run Block Automation

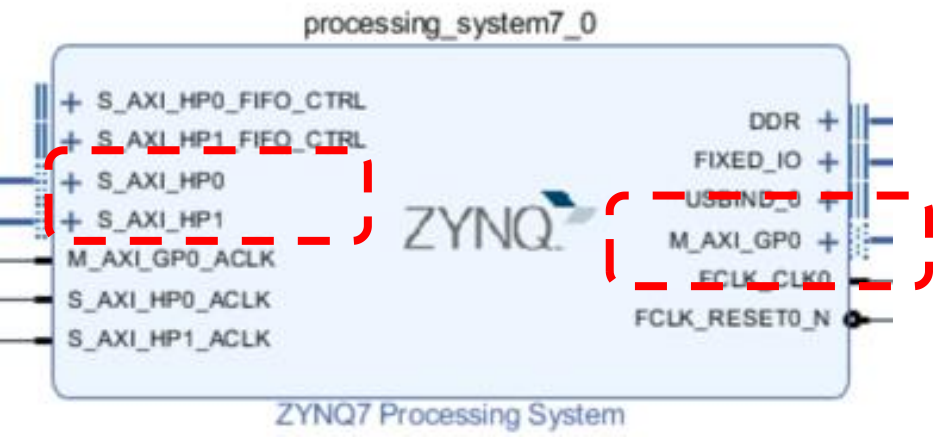
Vivado PS Configuration --1



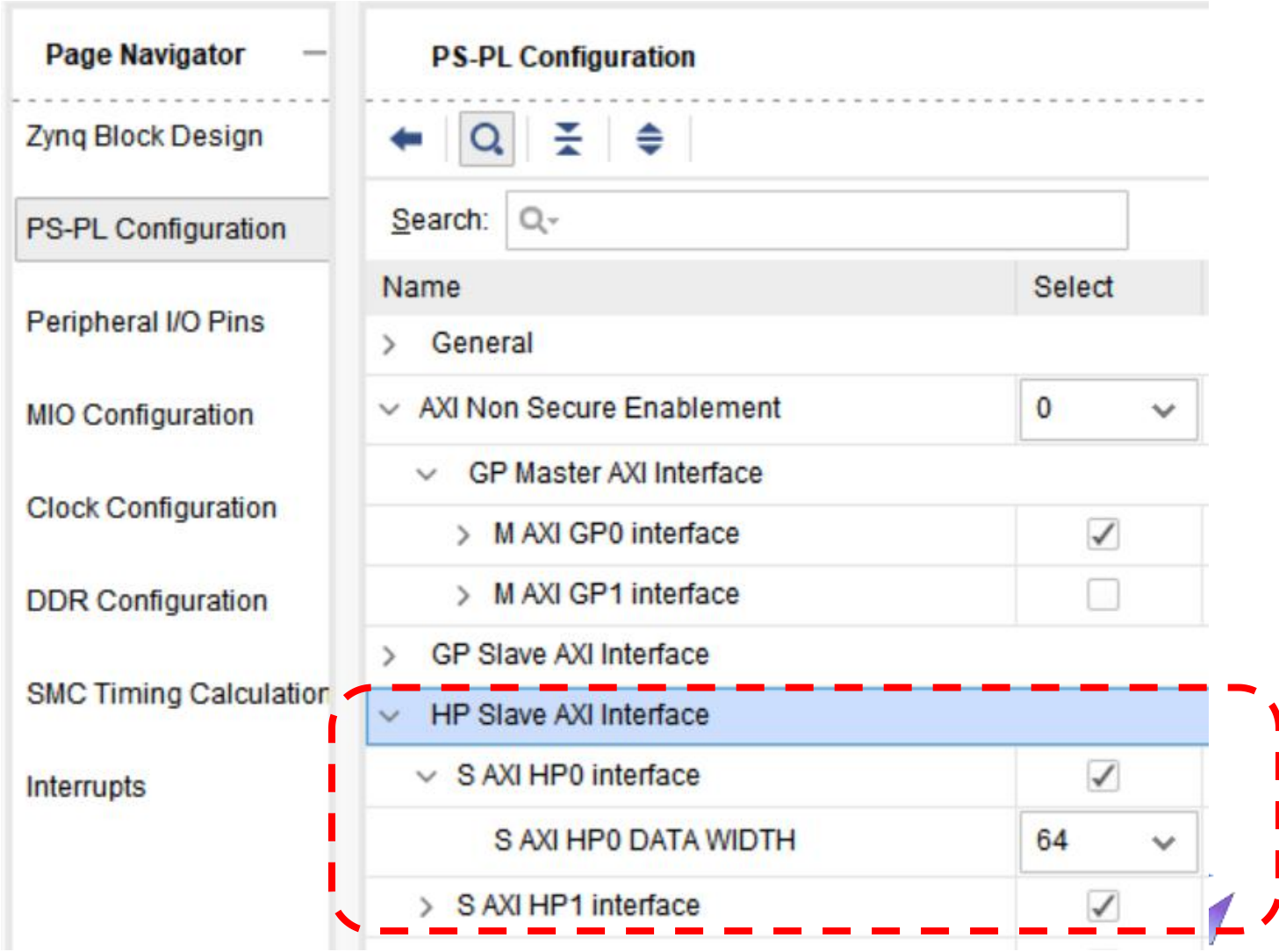


Auto configuration: DDR、I/O peripherals

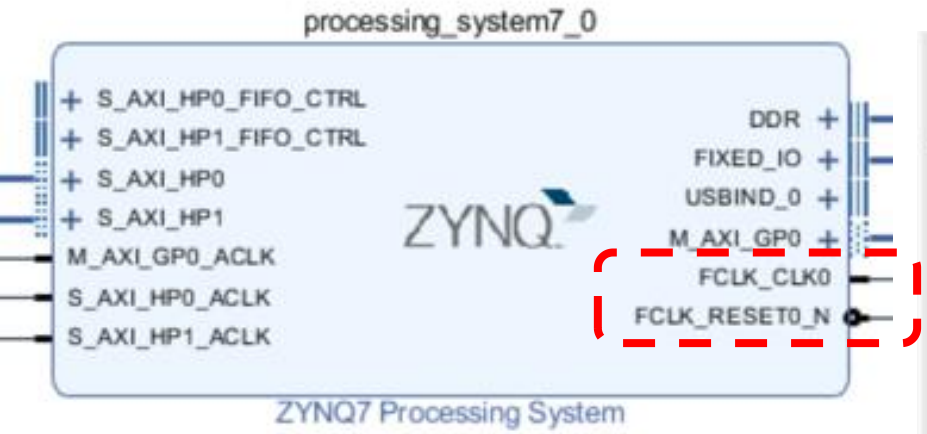
Vivado PS Configuration --2

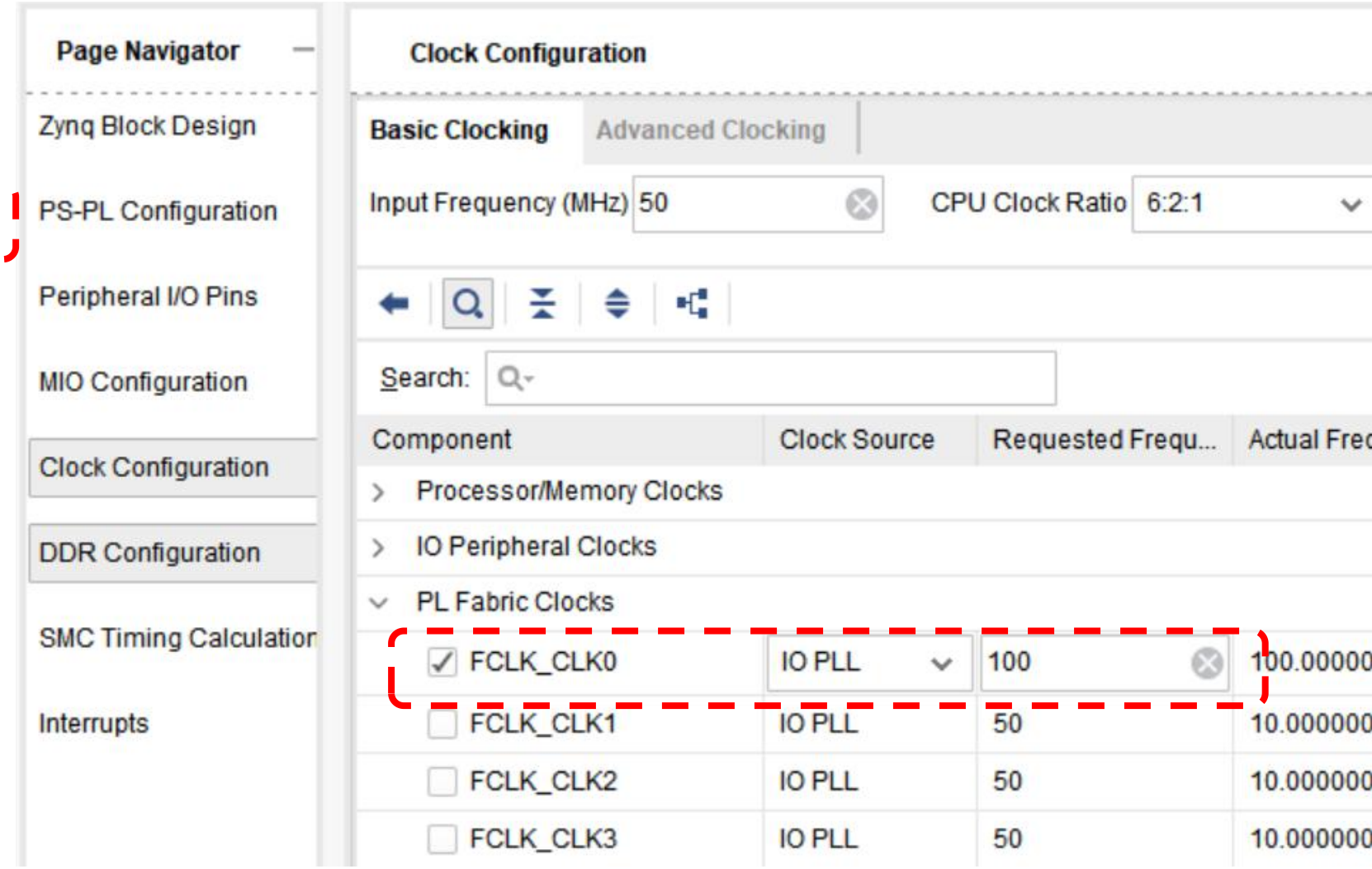


②PS-PL Configuration



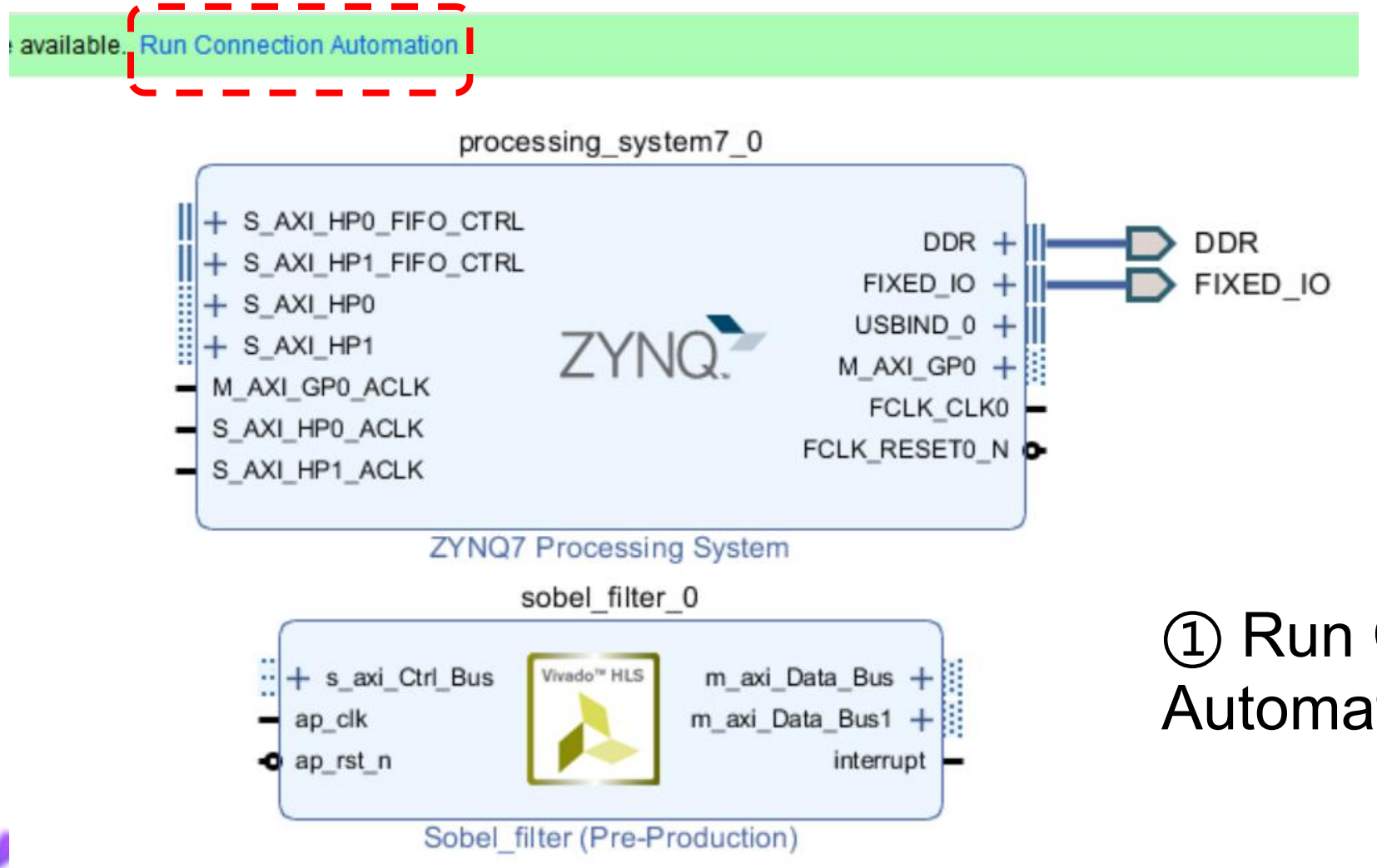
Vivado PS Configuration --3





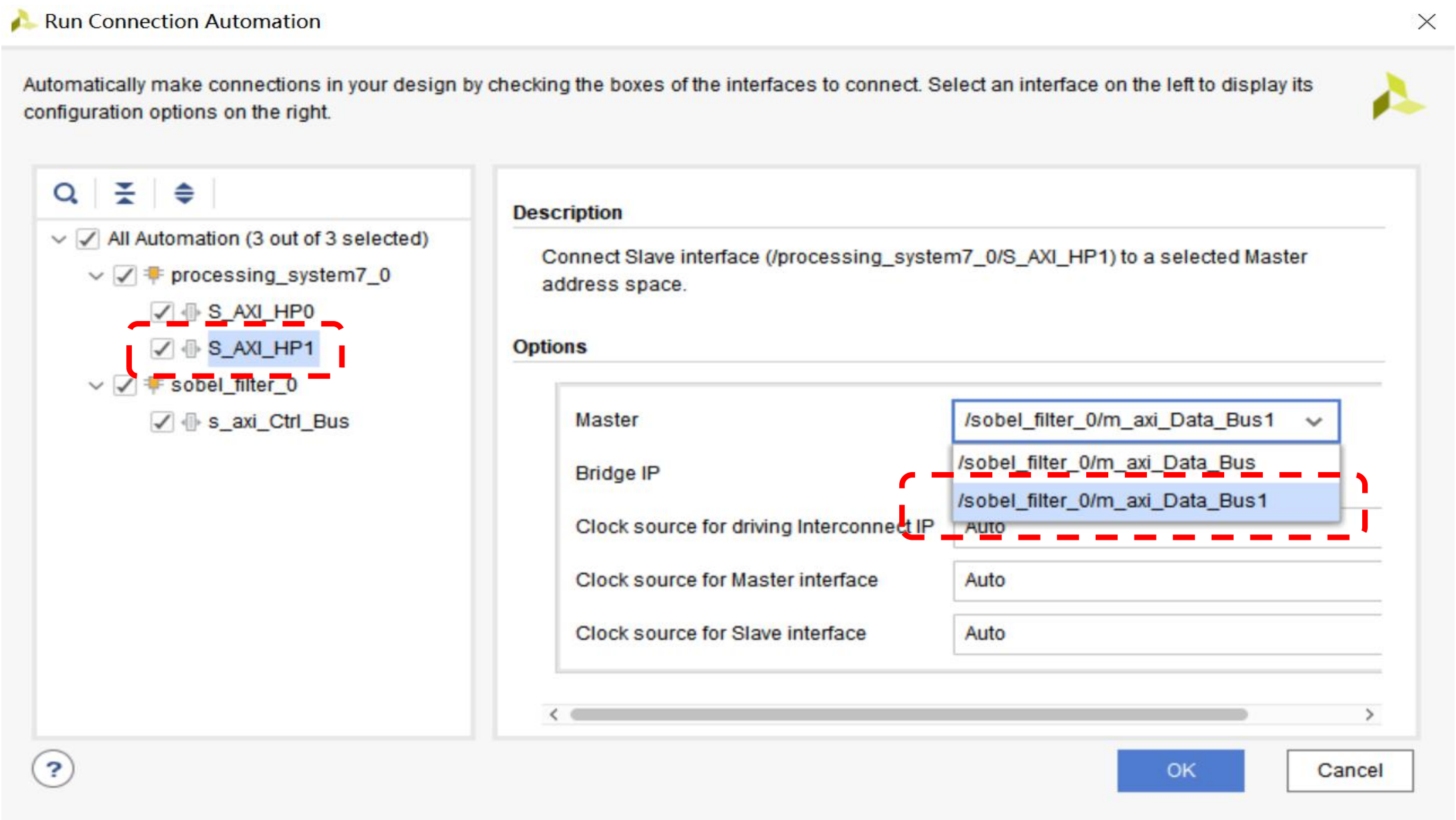
③Clock&interrupts Configuration

Vivado PS-PL Connection --1

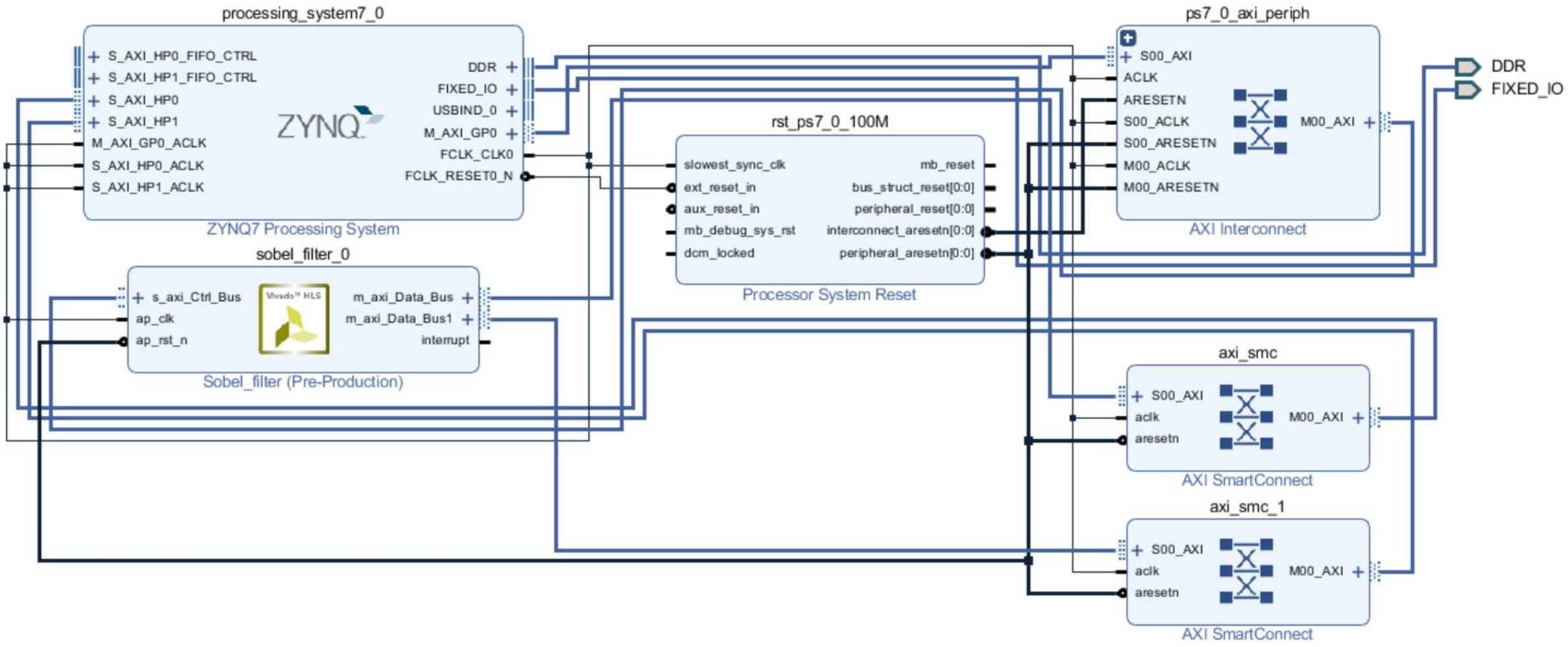


① Run Connection Automation

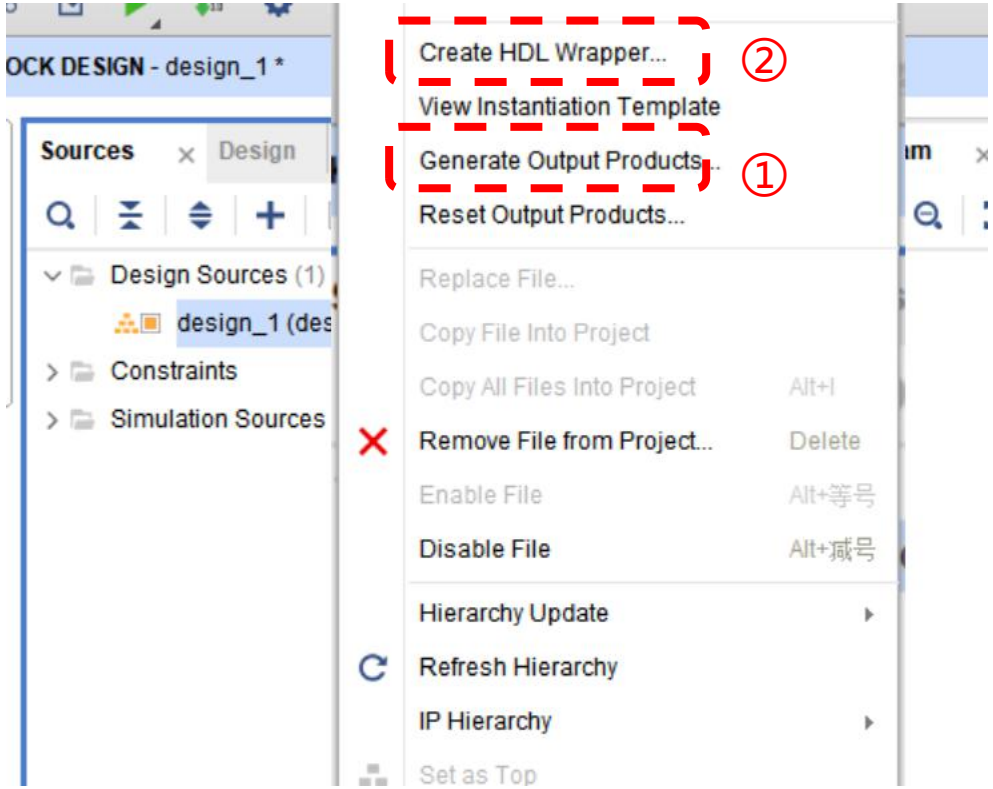
Vivado PS-PL Connection --2



Vivado PS-PL Connection --3



Generate output



①Generate output Products

②Create HDL Wrapper

Synthesis & implementation & bitstream

①

SYNTHESIS

Run Synthesis

Open Synthesized Design

IMPLEMENTATION

Run Implementation

Open Implemented Design

PROGRAM AND DEBUG

Generate Bitstream

Open Hardware Manager

②

prj_sobel - [E:/prj_vivado/prj_sobel/prj_sobel.xpr] - Vivado 2018.2

File Edit Flow Tools Reports Window Layout View

Project

Add Sources... Alt+A

Close Project

Save Block Design Ctrl+S

Save Block Design As...

Close Block Design

Constraints

Simulation Waveform

Checkpoint

IP

Text Editor

Import

Export

Launch SDK

Print... Ctrl+P

Exit

Export Hardware...

Export Block Design...

Export Bitstream File...

Export Simulation...

LOCK DESIGN - design_1 *

Sources Design

Design Sources (1)

design_1 (design

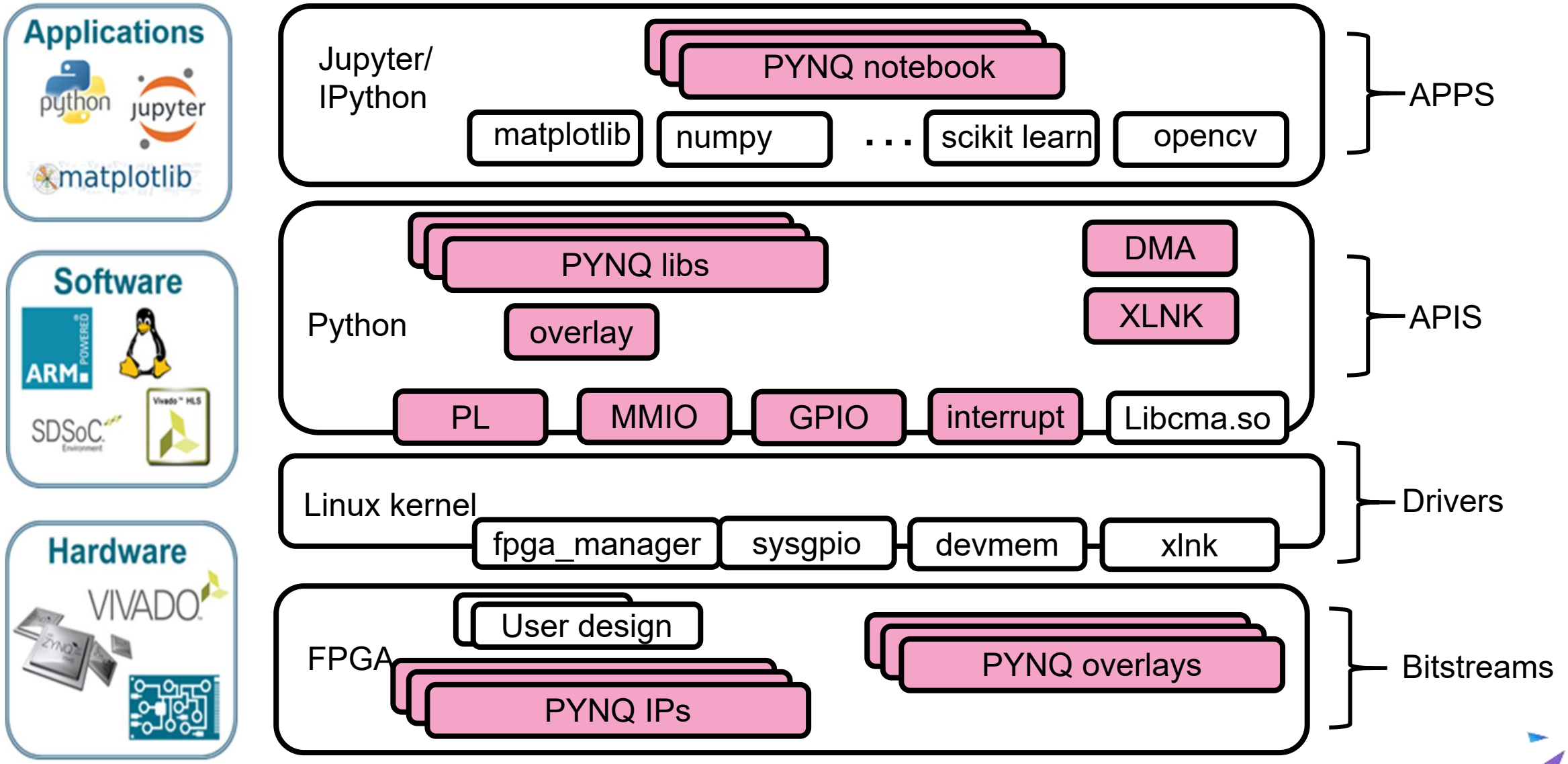
Constraints

Simulation Sources (1)

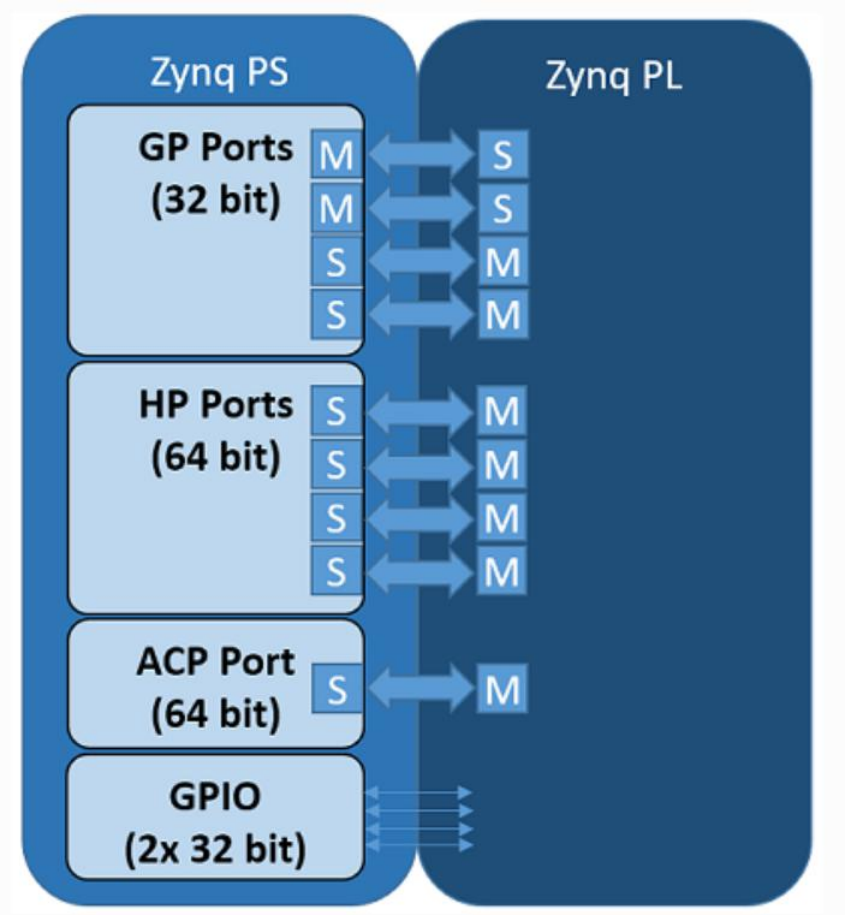
③

design_1_wrapper.bit
design_1_wrapper.tcl

PYNQ introduction



PYNQ PS/PL Interfaces



① Load Bitstream

```
Overlay  = Overlay("sobel.bit")
```

② MMIO（ Memory Mapped IO）

```
mmio = MMIO(0x43C00000, 0x1000)
mmio.write(0x10, 0x021)
```

③ Xlnk（Memory allocation）

```
xlnk = Xlnk()
input_buffer=xlnk.cma_array(shape=(5,),
            dtype=np.uint32)
```

④ DMA（Direct Memory Access）

⑤ GPIO（ General Purpose Input/Output）

```
output = GPIO(GPIO.get_gpio_pin(0), 'out’  )
output.write(0)
```

Sobel filter test on PYNQ

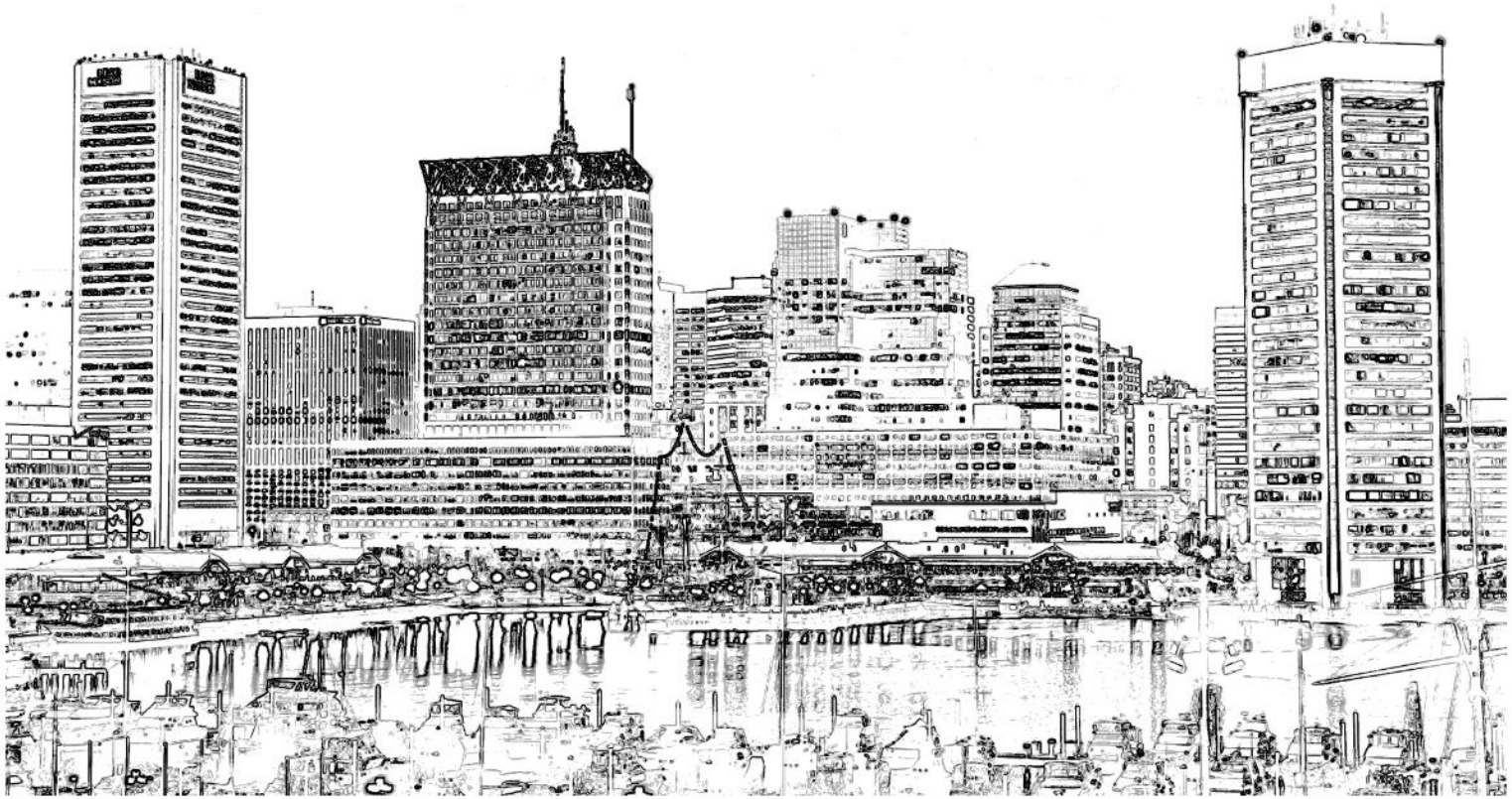
jupyter Untitled Last Checkpoint: 2018年5月4日 (autosaved)

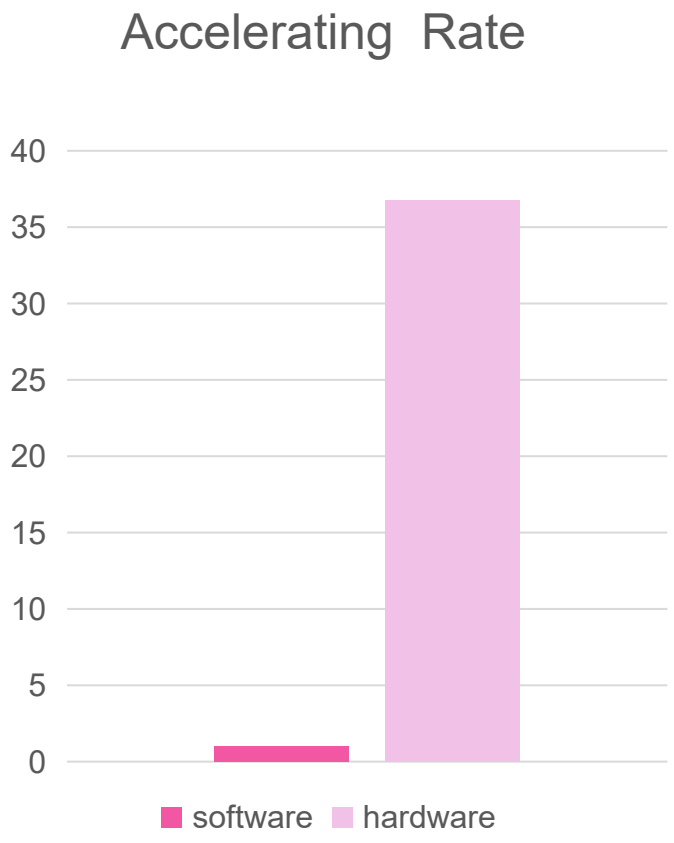
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
Sobel software time: 2.4683401584625244
```

Out[11]:





PART ONE

03

Optimization Methods

Loop; Array; Interface

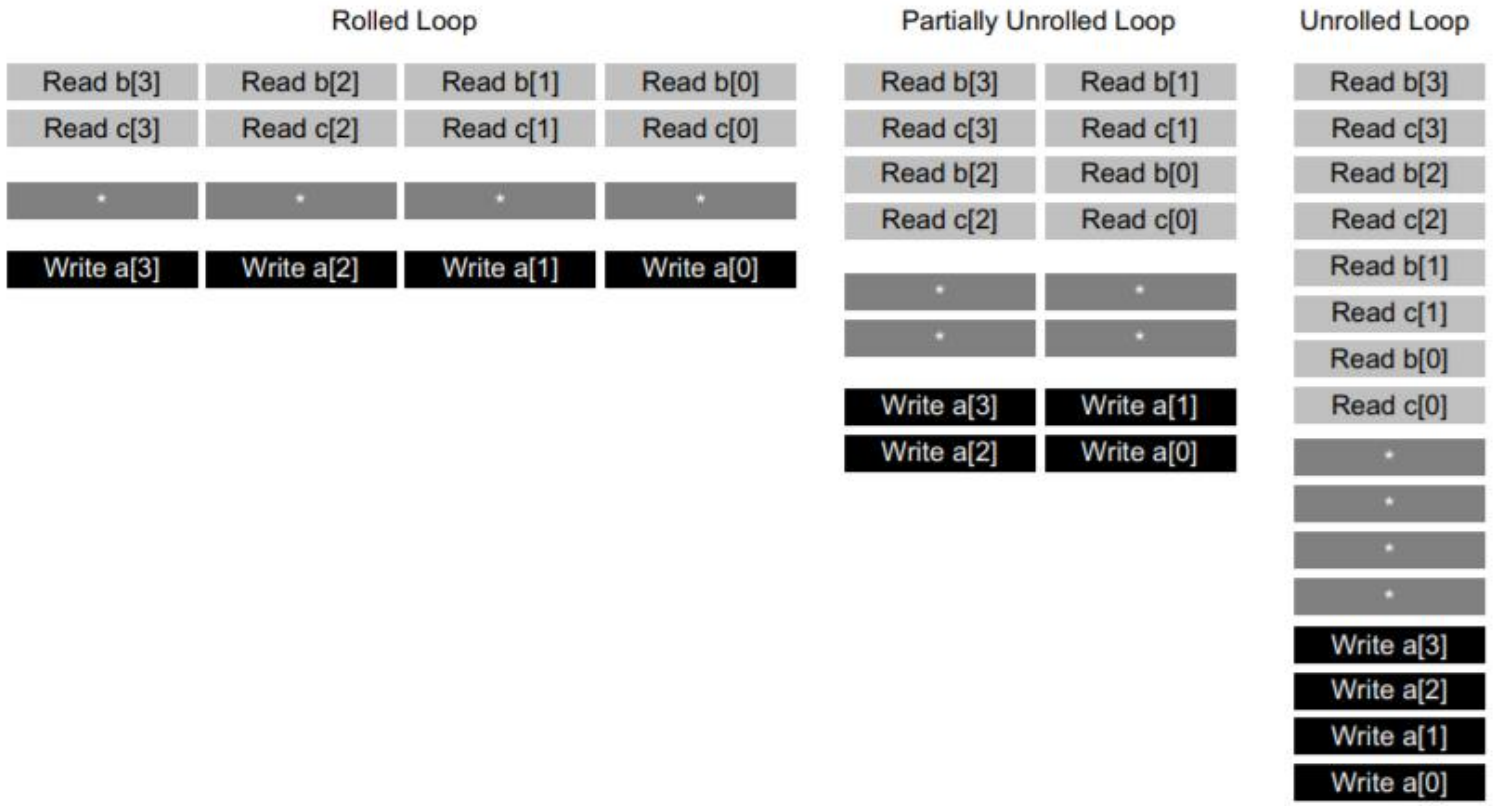
Optimization Methods

#pragma HLS UNROLL

- 功能：将**循环展开**，循环所有操作并行计算
- 性能比较：速度提升效果明显
- 资源比较：消耗大量逻辑计算资源，速度提升明显
- 指令细节：partially unrolled loop，unrolled loop

```
void  top(...) {

 ...
 for_mult:for (i=3;i>0;i--)  {
       a[i] = b[i] * c[i];
  }
 ...
}
```



Optimization Methods

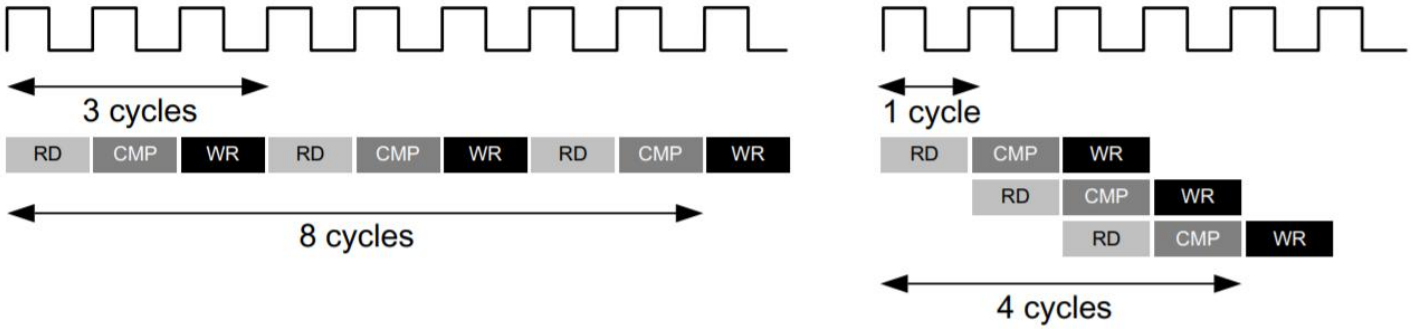
#pragma HLS PIPELINE

- 功能：将循环内操作进行流水线布局，采用时间并行方式加速
- 性能比较：加速效果好坏取决于加速的对象
- 资源比较：相比较循环并行展开资源消耗少。

```
void func(m,n,o) {

  for (i=2;i>=0;i--) {
       op_Read;
       op_Compute;
       op_Write;

  }
}
```



(A) Without Loop Pipelining

(B) With Loop Pipelining

Optimization Methods

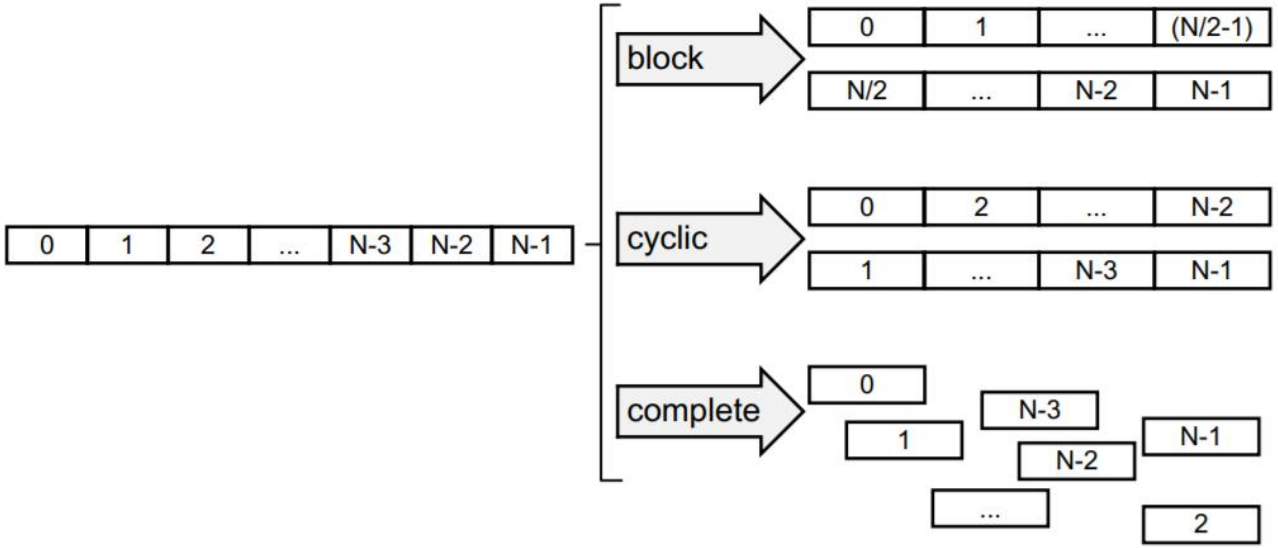
#pragma HLS ARRAY_PARTITION

- 功能：将存储位置在RAM中的数组修改并存储在寄存器或者多块RAM中，**消除**各数组内值的**连续依赖**，方便进行流水与并行等加速操作。

- 资源比较：占用大量逻辑资源

- 分割细节：block，cyclic，complete
dimension

- 数组自动Partitioning Automatic Array Partitioning：Solution > Solution Settings > General > Add > config_array_partition

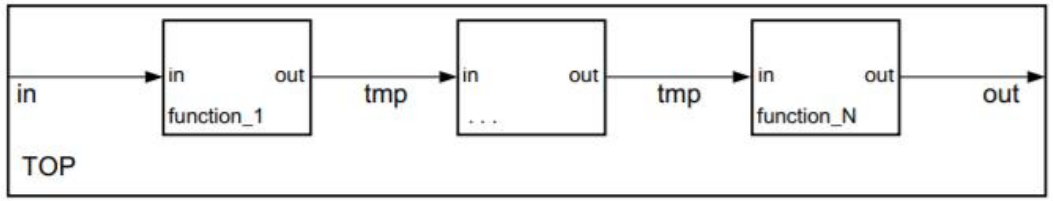


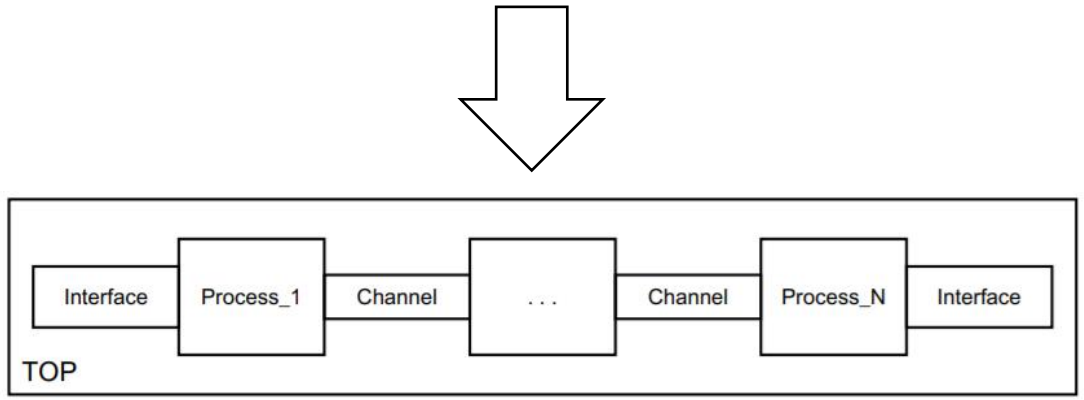
Optimization Methods

#pragma HLS Dataflow

- 功能：函数Fun()之间数据流水。

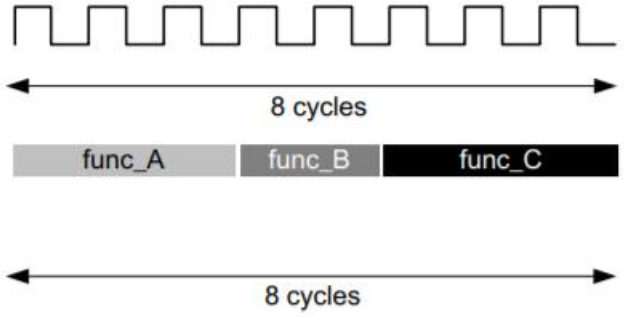
- Dataflow适用于函数之间
- Pipeline 适用于函数内



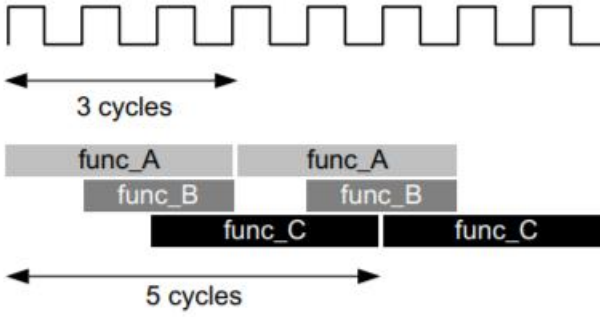



```
void top (a,b,c,d) {
  ...
  func_A(a,b,i1);
  func_B(c,i1,i2);
  func_C(i2,d)

  return d;
}
```



(A) Without Dataflow Pipelining



(B) With Dataflow Pipelining

PART ONE

04

Computing architecture

linbuffer;ping pong;dataflow

Data quantization

操作名称	DSP(个)	LUT(个)	FF(个)
浮点 32 位乘法	3	135	128
浮点 32 位加法	2	214	227
定点 32 位乘法	2	0	0
定点 32 位加法	2	0	0
定点 16 位乘法	1	0	0
定点 16 位加法	1	0	0

- 减少计算资源与存储资源
- 减少数据传输

- 量化方法：

$$V_q = (int)V \times 2^Q$$

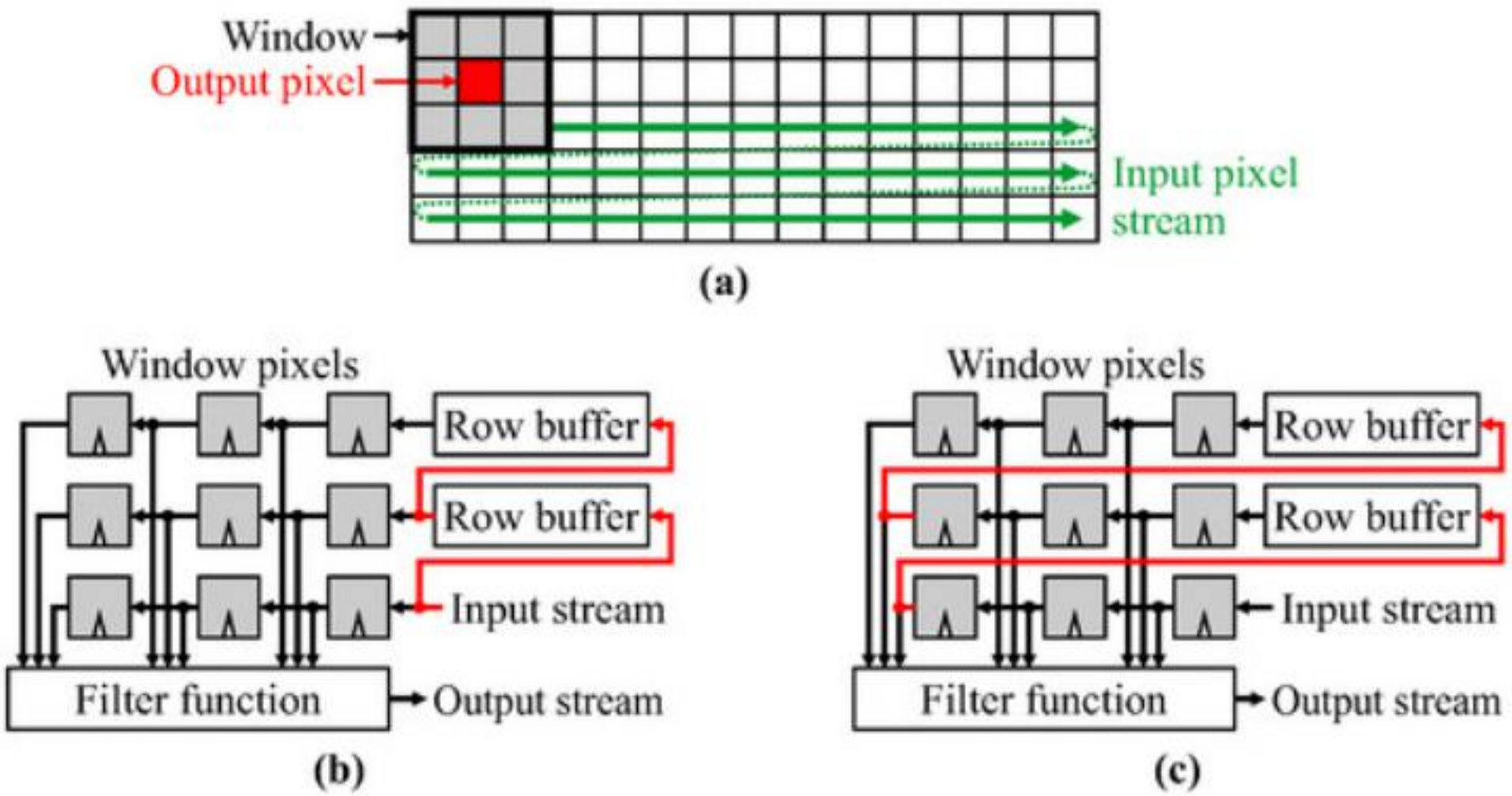
$$V = (float)V_q \times 2^{-Q}$$

- 例如：浮点数 Fx = 0.5，定标 Q = 15；

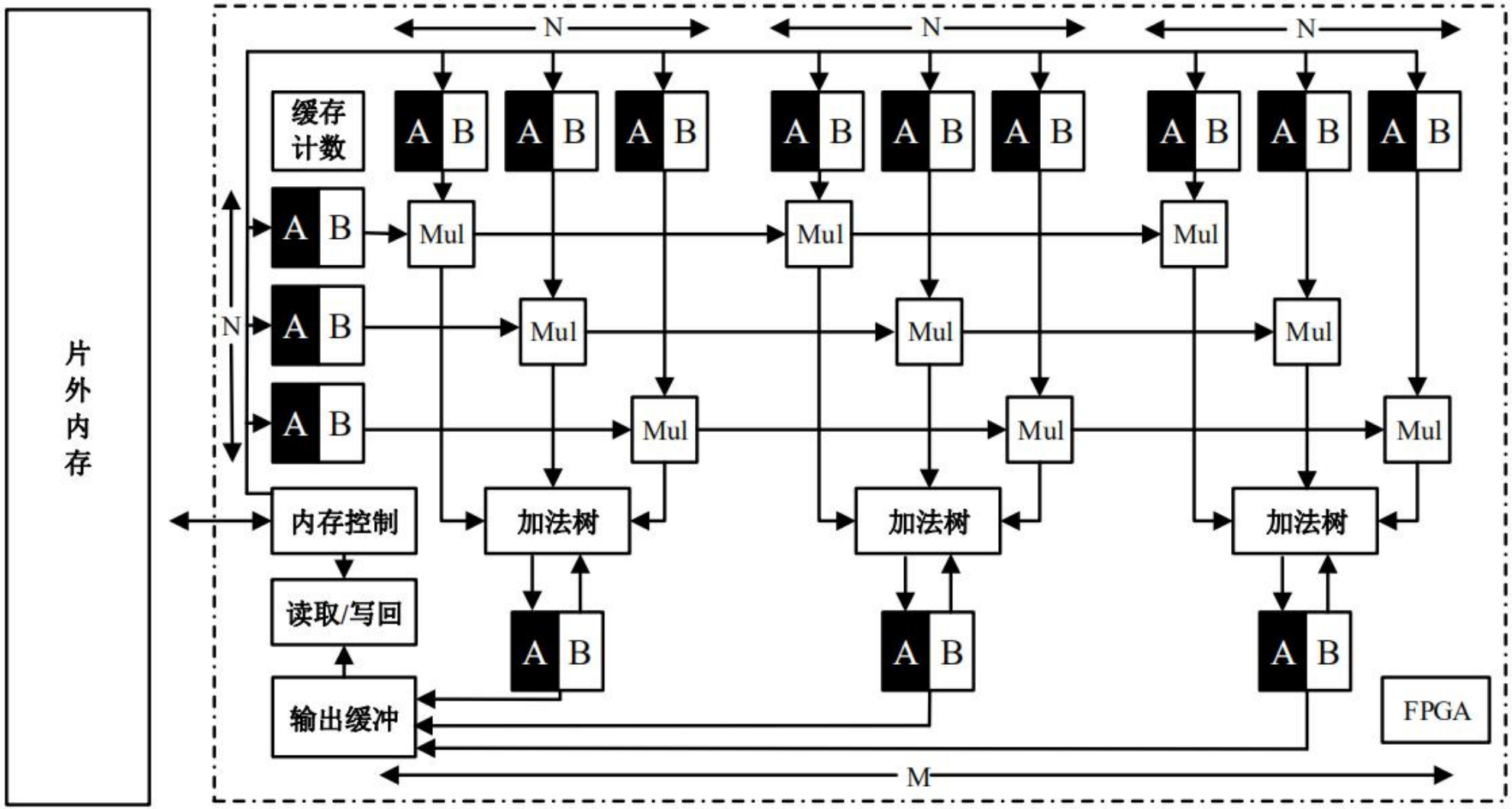
Ix = floor(0.5*32768) = 16384

Fx = （float）16384 * 2^(-15) = 16384 / 32768 = 0.5

Windows/Linebuffer

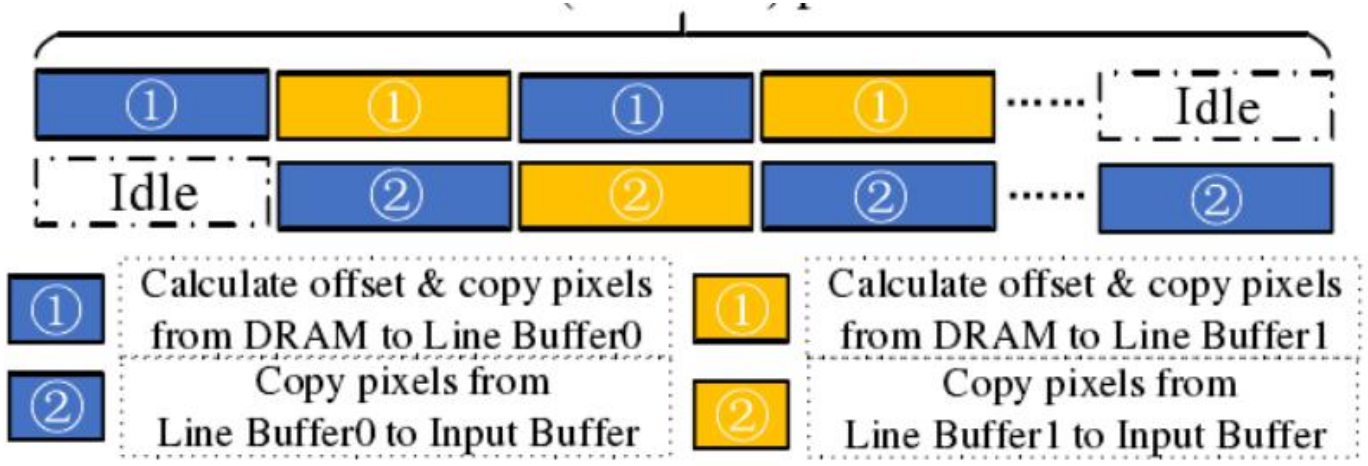


双缓冲流水计算架构



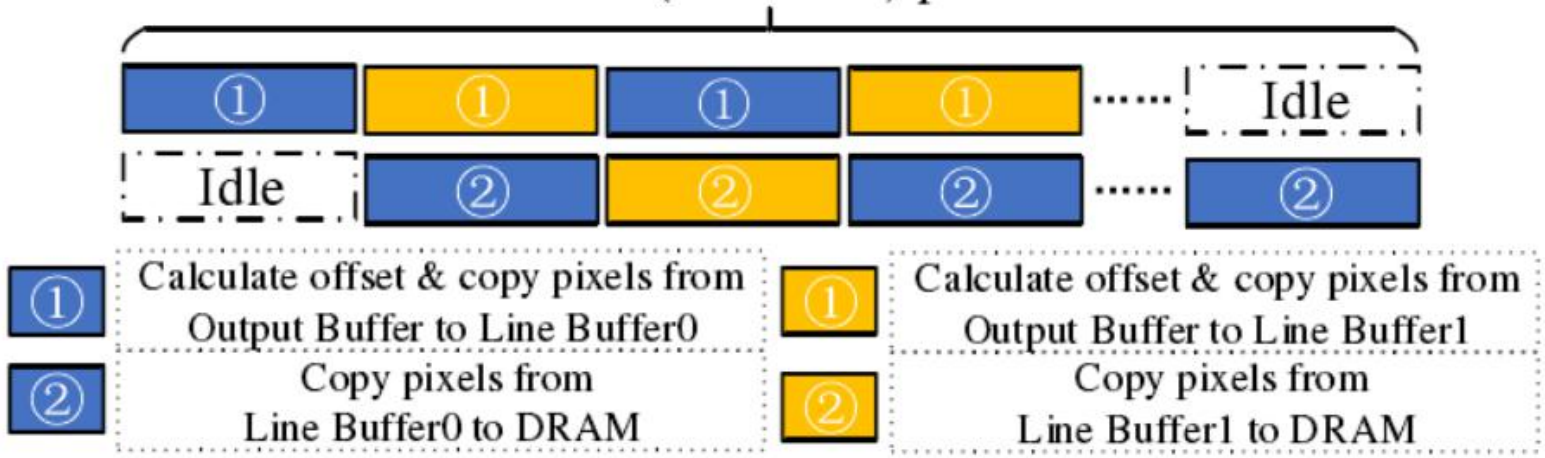
- 并行加法树
- 双缓冲
- 流水线

双缓冲流水计算架构



(a)

Tm*(OPPL +1) phases



(b)

➢ 双缓冲时序图

```
for(row = 0; row < rows : row++)
{
if (pingpongm == 0)
  {
    coupute(input0, output0);
   write_back_out(output1);
   pingpongm = 1;
  }
  else
  {
    coupute(input1, output1);
    write_back(output0);
    pingpongm = 0;
  }
}
```

Github

- Sobel_PYNQ： https://github.com/clancylea/pynq-sobel
- Yolov2_FPGA: https://github.com/dhm2013724/yolov2_xilinx_fpga
- NEST_FPGA:https://github.com/OpenHEC/SNN-simulator-on-PYNQcluster

2020

THANKS