XILINX®

Accelerating Real-Time AI Inference

Dachang Li, Shuai Zhang
Vitis AI Technical Marketing

Agenda

- Vitis/Vitis AI Overview
- Design Overview
- Design Implementation
- Summary

Vitis/Vitis AI Overview

Vitis Unified Software Platform

Domain-specific development environments

Vitis accelerated libraries

Vitis core development kit

Vision & Image Processing	Math & Linear Algebra	Quantitative Finance	TensorFlow Caffe **Vitis AI**	FFmpeg **Vitis Video**	**Partners** Genomics, Data Analytics, And more

Compilers | Analyzers | Debuggers

Xilinx runtime library (XRT)

Vitis target platform

Zynq-7000

Zynq UltraScale+ MPSoC

Alveo

Versal ACAPs

XILINX

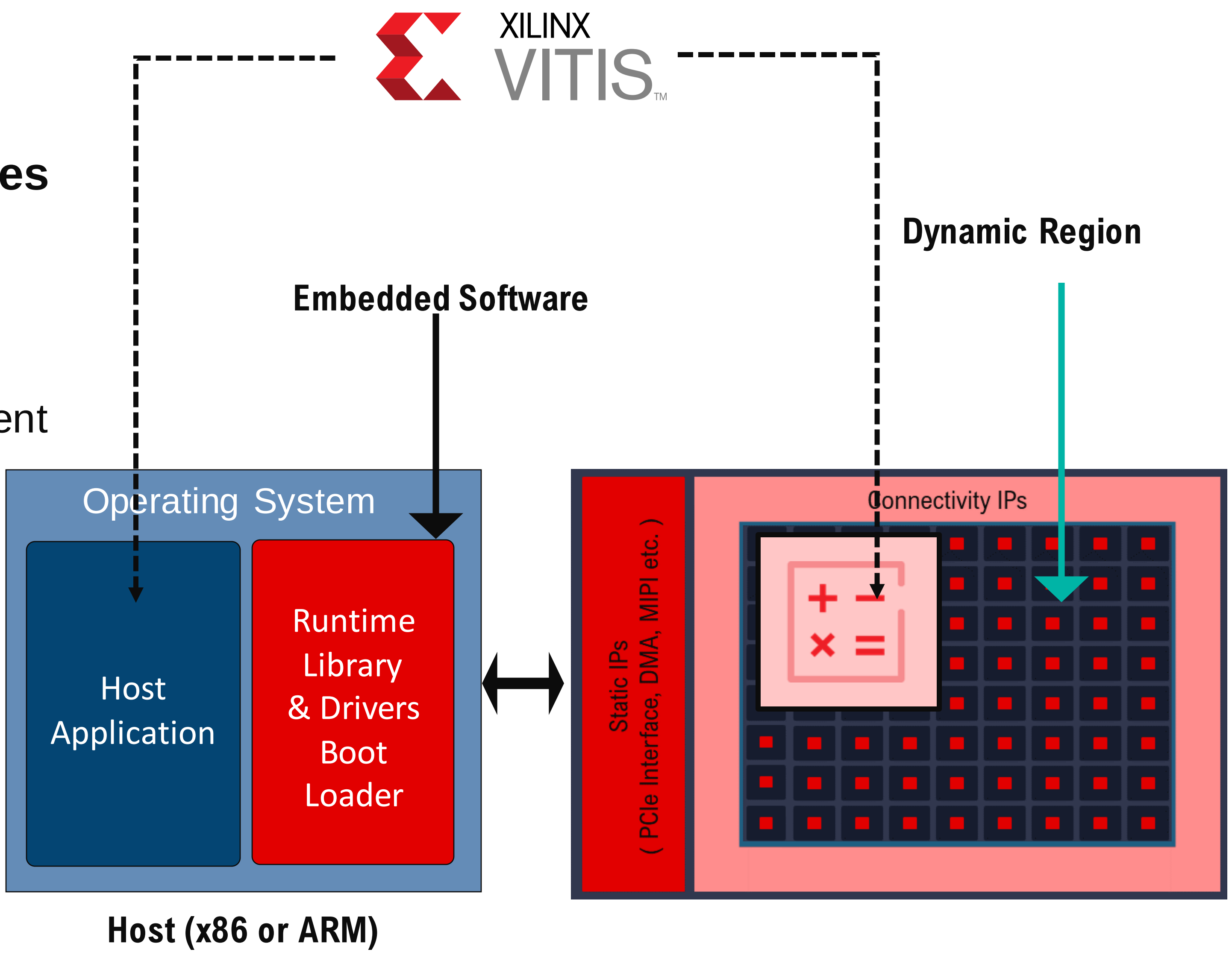
Vitis Target Platform

Base Hardware, Software Architecture

- **For PCIe Accelerator Cards, Includes**
  - PCIe® Interface Logic
  - DDR memory interface controllers
  - XDMA logic etc.
  - Hardware Config & Lifecycle Management

- **For Embedded Devices, Includes**
  - Operating System
  - Runtime library (XRT)
  - Runtime drivers (XRT)
  - Firmware & Boot loader

Ready-to-Use Vitis Target Platforms
OR
Build Your Own using Vivado Design Suite



All Developers Can Build and Deploy on All Platforms

Build

Embedded Developers

Enterprise Application Developers

Enterprise Infrastructure Developers

Data & AI Scientists

XILINX VITIS™

Deploy

Zynq-7000

Zynq UltraScale+ MPSoC

Alveo

Versal ACAPs

XILINX

Develop: Use Extensive, Open Source Vitis Libraries

GitHub

Domain-Specific Libraries

Vision & Image

Quantitative Finance

Data Analytics & Database

Data Compression

Data Security

Partner Libraries

Common Libraries

Math

Linear Algebra

Statistics

DSP

Data Management

500+ functions across multiple libraries for performance-optimized out-of-the-box acceleration

XILINX

Vitis Vision Library

- Performance-optimized kernel and primitive functions for
  - Color and bit-depth conversion, channel extractions, pixel-wise arithmetic ops.
  - Geometric transforms, image statistics, image filters
  - Feature detection and classifiers
  - 3D reconstructions
  - Motion Analysis and Tracking
- Support for color image processing and multi-channel support
- Multiple pixel/clock processing to meet through requirements
- Familiar OpenCV API interface

XILINX.

Vitis AI: ML Inference Solution

Frameworks

TensorFlow

Caffe

PyTorch

Vitis AI models

60+ pretrained, optimized reference models

Vitis AI development kit

AI Optimizer | AI Quantizer | AI Compiler | AI Profiler | AI Library

Xilinx runtime library (XRT)

Supports deploying custom AI models to Xilinx devices

Deep Learning Processing Unit (DPU)

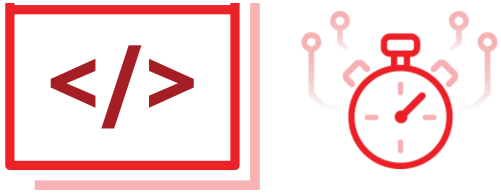
CNN DPU | LSTM DPU | MLP DPU

Optimized ”processor-like” IP for groups of AI workloads

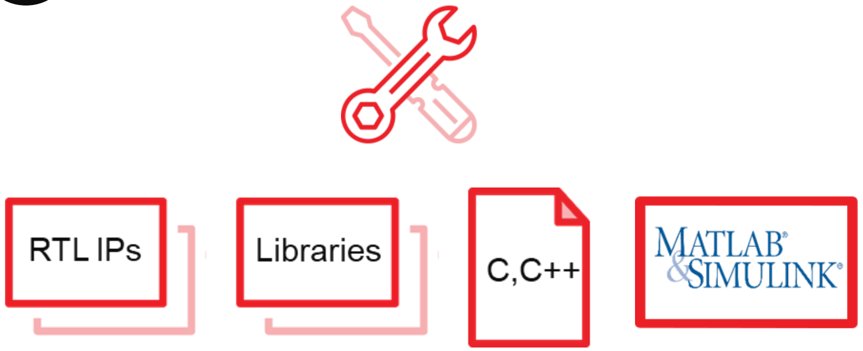
XILINX

Steps to Accelerate Applications with Vitis

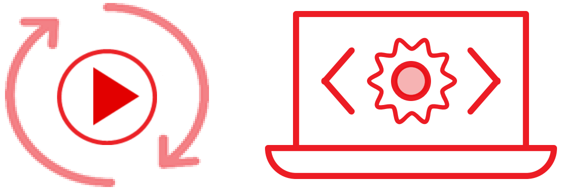
1 Profile Applications and Identify Performance-critical Functions



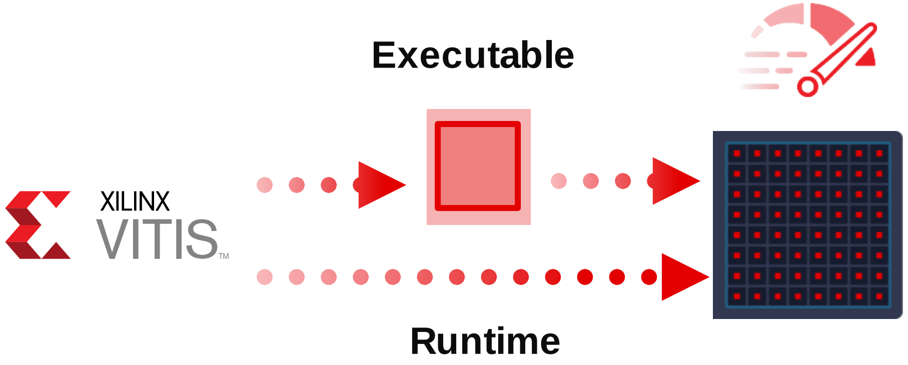
2 Design Accelerated Kernels



3 Build, Analyze & Debug : Validate Performance Goals Met

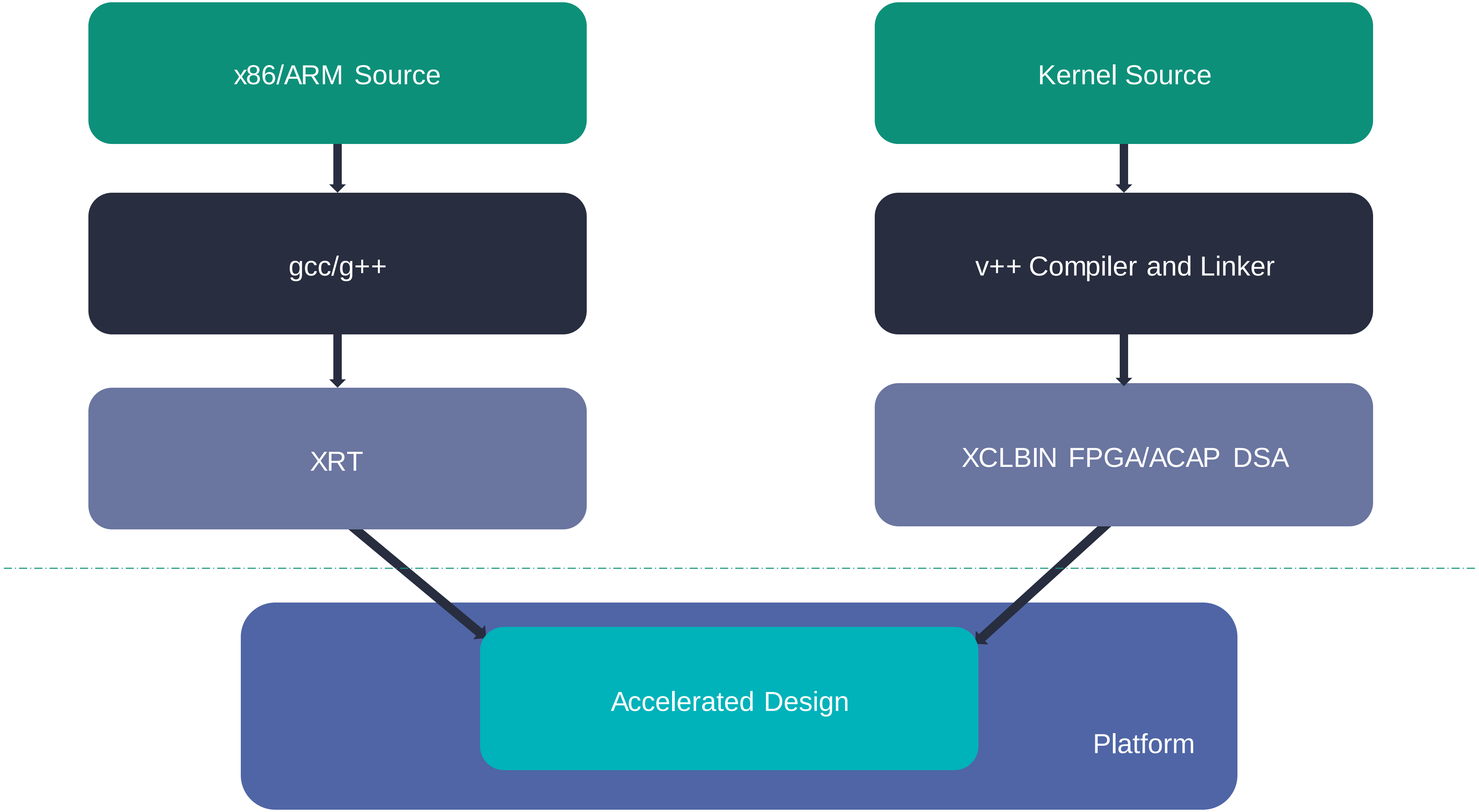


4 Deploy Accelerated Application on Xilinx Platforms



XILINX

Independent Development of SW and HW



Design Overview

Basic Idea

- Build a real-time human detection application based on zcu104, Vitis AI model zoo, Vitis AI library, DPU and Vision library.

Live Input

ML Inference (Human detection) on the ZCU104

Real time detection result

XILINX

System Configuration

- Camera
  - E-CON 3.4MP USB camera
  - Input format: UVYV, 2304x1296@30FPS
- ML Network
  - Caffe RefineDet (https://arxiv.org/abs/1711.06897)
  - Dataset: people class from COCO2014
  - Input format: BGR, 480x360
  - Computation amount: ~120GOP/frame
- DPU
  - Dual B4096@300MHz
- Target Performance
  - 30FPS end-to-end detection

RefineDet

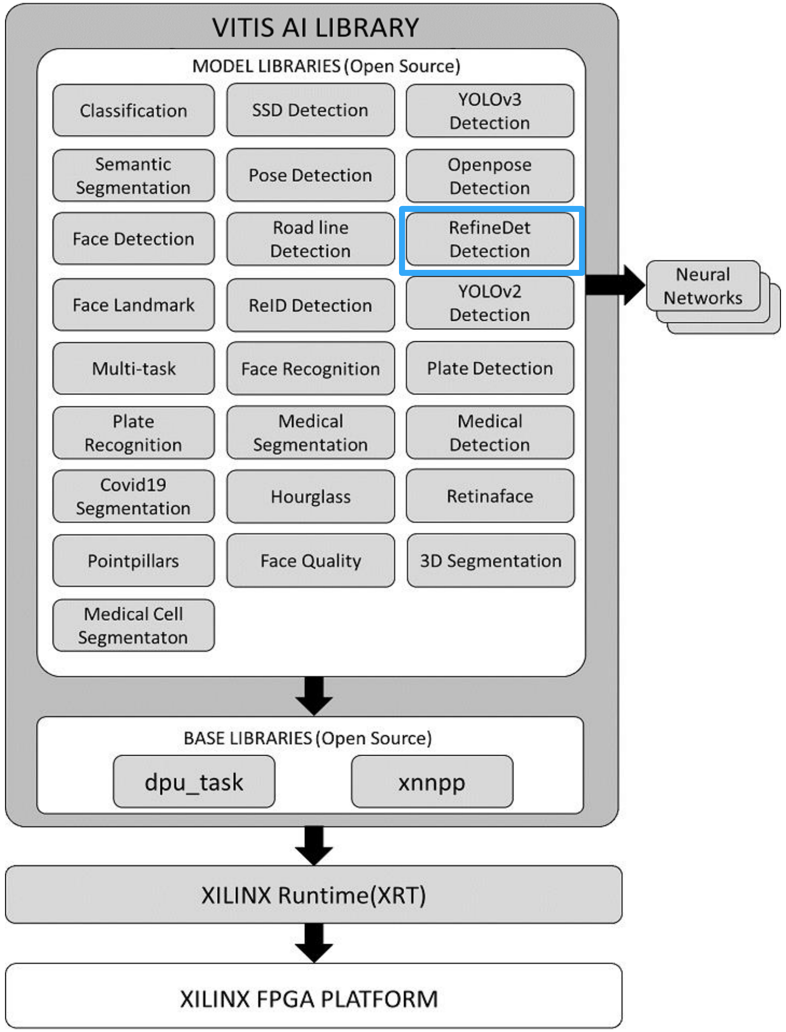
- Background
  - Improved version of SSD with addition of anchor refinement module (ARM), object detection module (ODM) and transfer connection block (TCB) for high accuracy
  - One-stage detection network level speed performance with two-stage network level accuracy

- Vitis AI Modification
  - The version provided by Xilinx model zoo has been modified based on the use case demand and Vitis AI solution constrain
  - Details can be found in Vitis In-Depth Tutorial Machine Learning Introduction Module 5

Vitis AI Deployment

- Vitis AI library enables fast deployment for common networks
  - Model specific libraries for released networks
  - Optimized common post-process function (xnnpp)
  - Low-level API for custom model deployment (dpu_task)

- Model library “RefinetDet Detection” will be used in this design to handle ML inference



Pre-processing for ML Inference

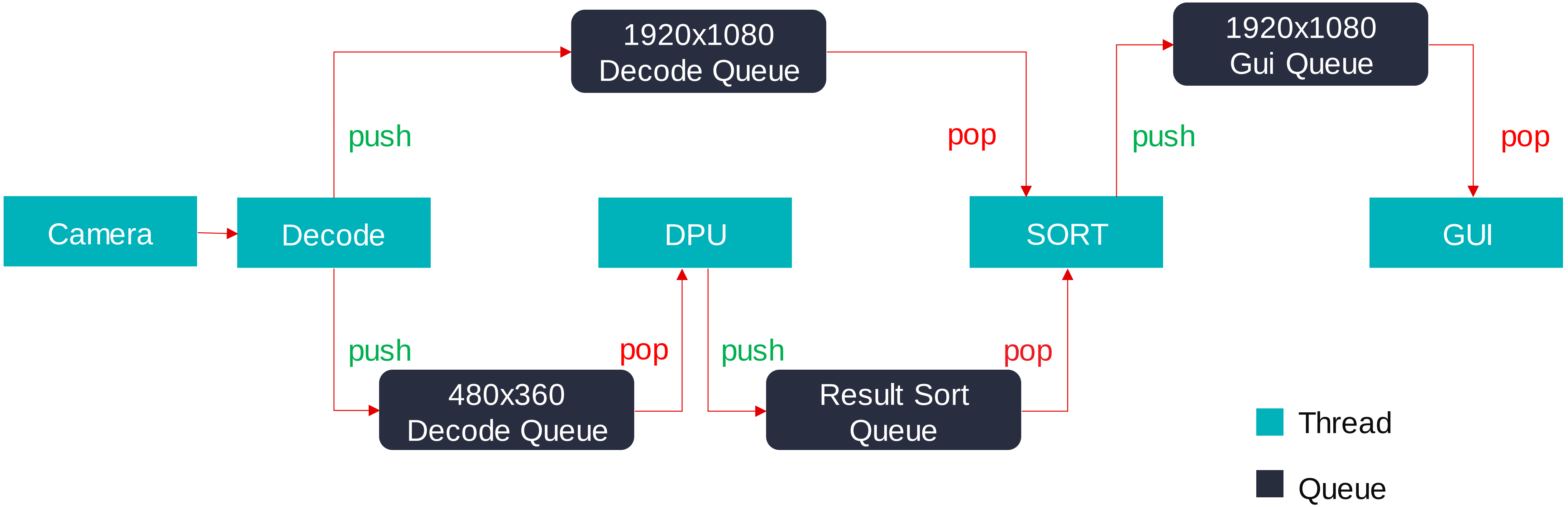
- RefineDet requires following image pre-process to be correctly performed
  - Channel order: BGR
  - Image resize: 480x360
  - Mean value subtraction: 104, 117, 123 (B,G,R)
  - Scale: 1

- Vitis AI will take care of mean value and scale value when using model specific library

- Color space conversion and resize need to be implemented by users

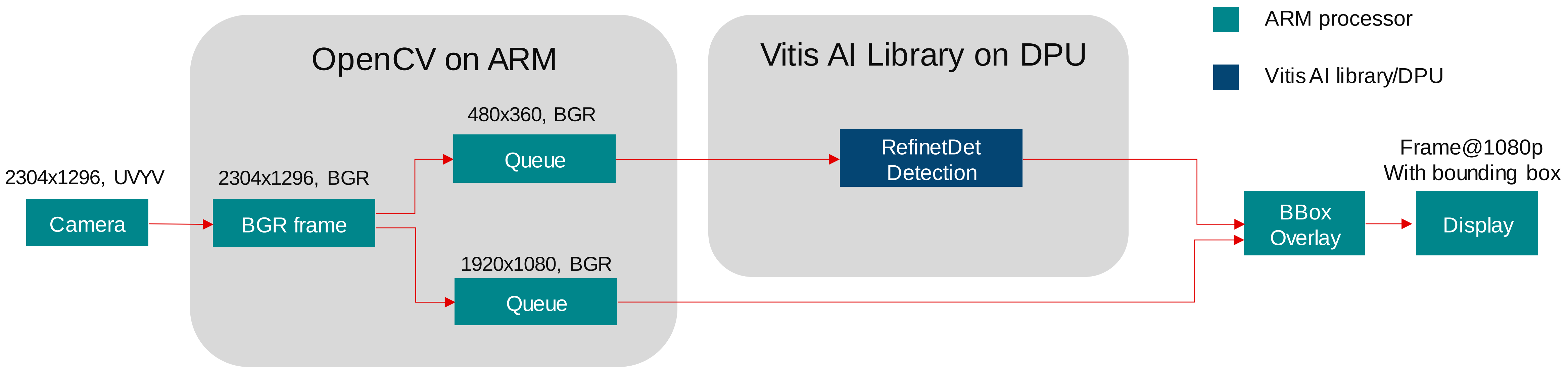
Data Flow

- Different threads are designed in pipelined style and will run in parallel to maximize throughput



Detailed Design Implementation

Start with Baseline Application



- In the baseline implementation, OpenCV will be used for image processing for the simplicity and will run on the ARM processor
- The Decode thread will call resize function two times for 1920x1080 and 480x360 frames respectively
- ML inference will run mainly on DPU in PL with very small portion of Vitis AI library process on the ARM (mean value subtraction in this case)

Easy Implementation of Core Function

- Both image processing and DPU inferencing functions can be implemented with few lines of code

```
if (m_device->getFormat() == V4L2_PIX_FMT_UYVY)
{
    cv::Mat v4l2Mat = cv::Mat(m_device->getHeight(), m_device->getWidth(), CV_8UC2, (void *)buffer);
    cv::Mat src, dst;
    cv::cvtColor(v4l2Mat, src, cv::COLOR_YUV2BGR_UYVY);
    readImage.reserve(2);
    auto size_show = cv::Size(1920, 1080);
    auto size_dpu = cv::Size(480, 360);
    cv::resize(src, src, size_show);
    readImage.emplace_back(src);
    cv::resize(src, dst, size_dpu);
    readImage.emplace_back(dst);
}
```

Function "V4l2Capture::read_images" is implemented by
- "cv::cvtColor"
- "cv::resize"

```
const auto model_name = argv[1];
/**
 * @brief push back the g_num_of_threads of DPU Filters into DPU THREAD
 *
 */
for (int i = 0; i < g_num_of_threads[0]; ++i) {
  dpu_thread.emplace_back(new DpuThread(
      create_dpu_filter( [model_name]() {return vitis::ai::RefineDet::create(model_name, 0);}, process_result),
      decode_queue.get(),sorting_queue.get(), std::to_string(i)));
}
```

DPU inferencing is implemented by
- create_dpu_filter
- vitis::ai::RefineDet

Hardware Integration in Software Way

- All the hardware blocks are integrated into the system using the v++ compiler, which looks and feels like a standard SW compiler
- To build a system with kernels, or ".xo"s, we can link with a Makefile

```
ZCU104_XOS = dpu_b4096_zcu104.xo
```

- To integrate the DPU, just copy/paste the source from the Vitis AI repository and add it to your project – no need to open Vivado!
  - Need two DPUs for your system? Three? Easily configure system topology with version-controllable parameters

```
# This project has a lot of ML – add a second DPU!
nk=DPUCZDX8G:2
```

- Numerous examples available online and in our Git repositories

XILINX

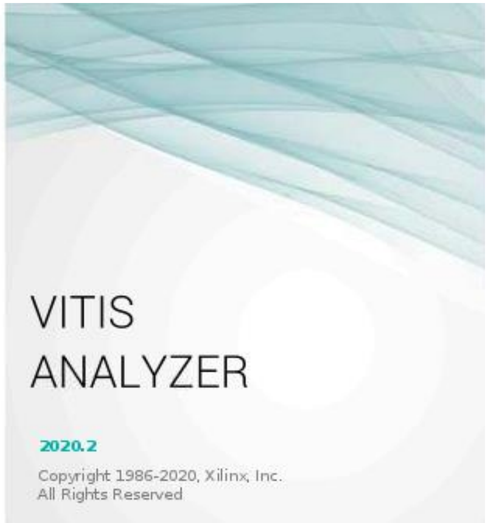
Performance Estimation

- We could implement baseline application very easily but how about the performance?
- Rough ML inference estimation
  - Single B4096 DPU core provides around 1200GOP peak performance at 300MHz
  - RefineDet consumes around 120GOP to process one frame
  - In best case (100% efficiency), 10FPS for single core or 20FPS for dual cores which **cannot** meet the target performance
- Don’t forget the image process has to be taken into consideration too!
  - The overall end-to-end performance will be far from our target
- What if we want to see actual application profiling information?

XILINX

Vitis Analyzer

- Vitis analyzer is the powerful tool to visualize application profiling information, including SW code running time, kernel compute time, data movement and etc.
- Vitis AI profiler has been integrated into Vitis analyzer latest version to better profile applications based on DPU and Vitis AI library



Vitis Analyzer Usage – Step 1

- Create cfg.json used to profile DPU, common libraries and custom functions
  - Common libraries: vitis-ai-library, opencv, vart and xnnpp_post_process
  - Custom function: DecodeThread::run, DpuThread::run and etc

```
{
"options": {
     "runmode": "normal"
  },
  "trace": {
     "enable_trace_list": ["vitis-ai-library", "opencv", "vart","xnnpp_post_process", "custom"]
  },
  "trace_custom": ["read_images_with_kernel",
                   "DecodeThread::run",
                   "DpuThread::run",
                   "GuiThread::run",
                   "SortThread::run"]
}
```

Add custom function name for profiling

Vitis Analyzer Usage – Step2

- Create xrt.ini
  - Profiling of HLS kernel needs the “xrt.ini” file to specified mode parameters
  - Place it in the same directory as the application
- The config file format is shown as below:

```
[Debug]
Profile=true
xrt_profile=true
vitis_ai_profile=true
lop_trace=true
data_transfer_trace=coarse
```

Vitis Analyzer Usage – Step 3

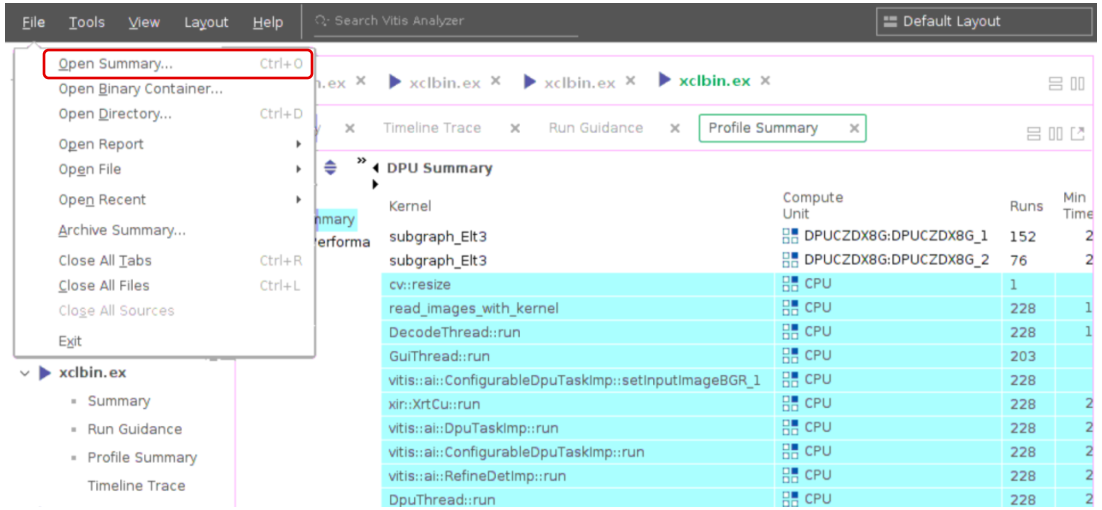
- Use “vaitrace” to run the application with config file
  - *vaitrace -c cfg.json ./<application_name> <model_name> 0 -t <thread_num>*

- The meta data will be generated after the application is stopped.

```
├── hal_host_trace.csv
├── profile_summary.csv
├── vart_trace.csv
├── vitis_ai_profile.csv
└── xclbin.ex.run_summary
```

Vitis Analyzer Usage – Step 4

- Inspect summary files on the host machine with Vitis Analyzer
  - Transfer meta files from step 3 to host machine
  - On the host, run the command ***vitis_analyzer***
  - Click ***File -> Open Summary***
  - Select and open the the summary file
  - Check execution time of each components
  - Find performance bottleneck for improvement



Custom Functions Hierarchy

- Vitis Analyzer will give time information based on function names
  - In this design, the hierarchy of custom functions is as below
  - DecodeThread and DpuThread are two main components

```
DecodeThread::run
        cv::resize
        cv::cvtcolor
DpuThread::run
        vitis::ai::RefineDetImp::run
                vitis::ai::ConfigurableDpuTaskImp::setInputImageBGR
                vitis::ai::ConfigurableDpuTaskImp::run
                        vitis::ai::DpuTaskImp::run
                                xir::XrtCu::run
                                        subgraph_Elt3
                                        subgraph_Elt3
SortingThread::run
GuiThread::run
```

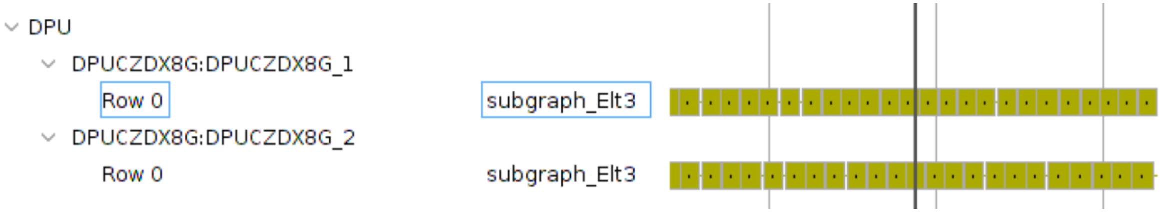
Baseline Application Profiling

Pre-processing on ARM | Baseline model on DPU

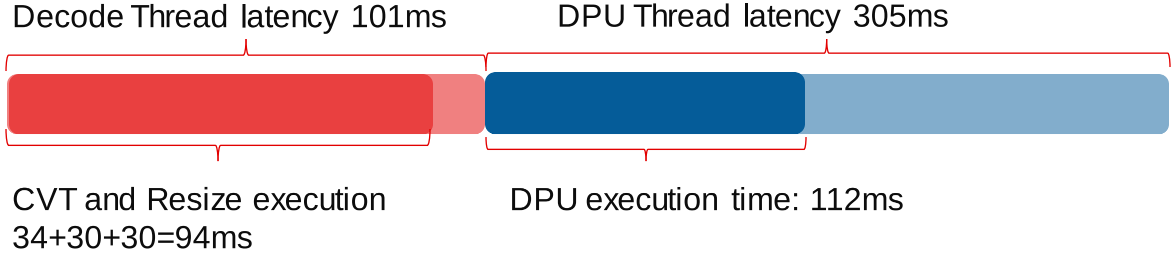
- Baseline application runs at 10.2FPS
  - DecodeThread takes 101ms/frame, i.e., 10FPS
  - DPU core takes 110ms/frame, i.e., 9FPS for single core and 18FPS for dual cores

```
root@xilinx-zcu104-2020_1:~# vaitrace -c cfg.json
./usb_input_multi_threads_refinedet_drm refinedet_baseline 0 -t 3
Setting env
INFO:root:VART will run xmodel in [NORMAL] mode
……
INFO:root:Generating VTF
INFO:root:Overall FPS 10.20
```

Kernel	Compute Unit	Runs	Min Time (ms)	Avg Time (ms)	Max Time (ms)
subgraph_Elt3	DPUCZDX8G:DPUCZDX8G_1	115	108.517	111.877	139.702
subgraph_Elt3	DPUCZDX8G:DPUCZDX8G_2	57	110.451	111.412	122.918
cv::resize	CPU	345	0.073	30.073	102.425
cv::cvtColor	CPU	172	17.369	34.938	65.576
V4l2Capture::read_images	CPU	172	67.043	100.034	163.977
DecodeThread::run	CPU	172	67.210	101.712	299.480
vitis::ai::ConfigurableDpuTaskImp::setInputImageBGR_1	CPU	172	0.341	0.501	0.851
xir::XrtCu::run	CPU	172	108.595	111.889	139.821
vitis::ai::DpuTaskImp::run	CPU	172	109.408	115.918	152.617
vitis::ai::ConfigurableDpuTaskImp::run	CPU	172	109.423	115.942	152.644
vitis::ai::RefineDetImp::run	CPU	172	110.441	126.188	165.560
DpuThread::run	CPU	174	197.807	305.321	500.296
GuiThread::run	CPU	172	33.176	102.537	318.932
SortingThread::run_1	CPU	2,236	0.004	0.011	4.565
SortingThread::run_2	CPU	344	0.004	0.037	6.866
SortingThread::run	CPU	172	45.373	102.397	307.602



Thread name	Decode	DPU	SORT	GUI
Parallel Thread Num	1	3	1	1
Thread Latency(ms)	101	305	102	102



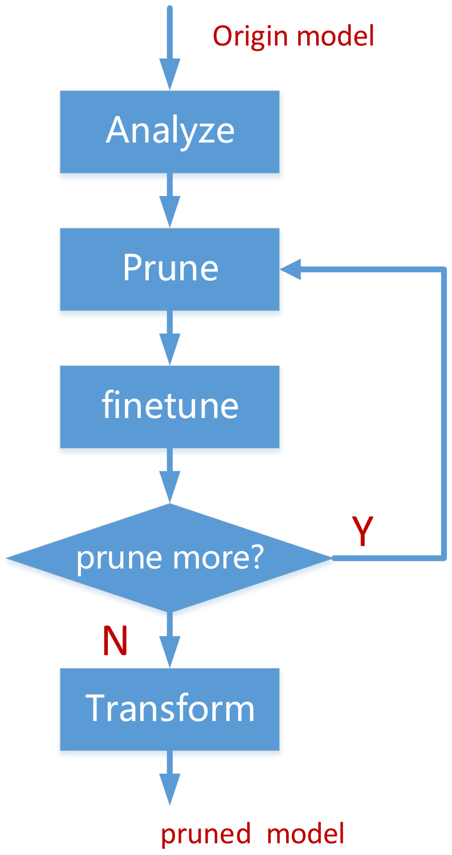
XILINX

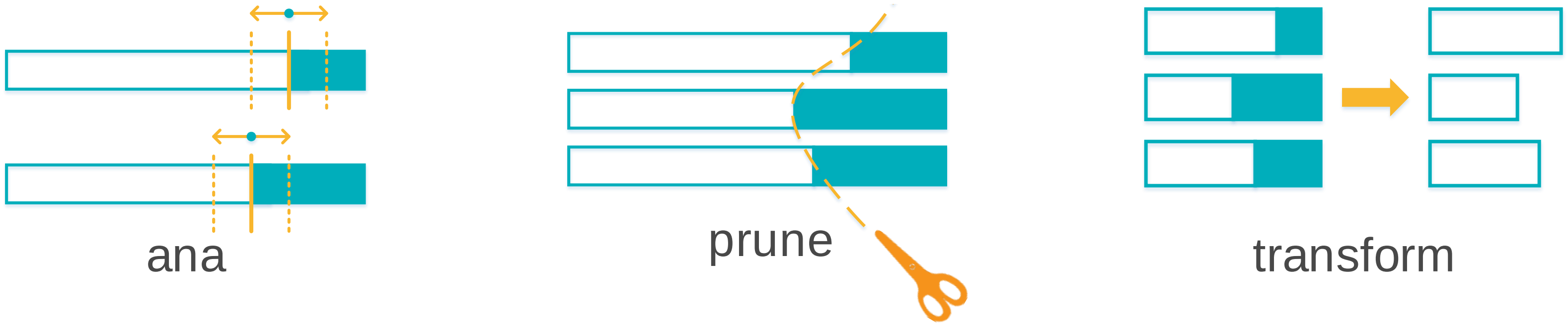
Baseline Application Bottleneck

- Profile result
  - Camera is capable of generating data frames at 30FPS
  - DecodeThread is only capable of processing frames at 10FPS
  - DPUs are only capable of inferencing baseline model at 20FPS
- Bottlenecks
  - DecodeThread has to be faster than 30FPS
    - Optimize software code efficiency – cannot get 3x performance boost
    - Accelerate by programming logic – promising
  - DPU inference performance has to be faster than 30FPS too
    - Increase DPU core computation ablity – not feasible, already using largest core
    - Reduce network computation amount – feasible way with Vitis AI

Vitis AI Optimizer

- Vitis AI optimizer (vai_p) is capable of reducing redundant connections and the overall operations of networks in iterative way
  - Automatically analysis and prune the network models to desired sparsity
  - Significantly reduce the OPs and parameters of networks without losing much accuracy
- Five functions to optimize model
  - ana – run sensitivity analysis
  - prune – prune the network according to config
  - finetune – finetune the network to recovery accuracy
  - transform – transform the pruned model to regular model
  - stat – get flops and the number of parameters of a model





ML Acceleration by Vitis AI Optimizer

- The computation amount of RefineDet could be efficiently reduced by optimizer
  - Latency is reduced and maximum throughput is increased
  - Use 80% pruning ratio model could meet our target with big margin (76FPS vs 30FPS)

Model	Pruning Ratio	Operation (GOP)	Latency (ms)*	Throughput (FPS)**
RefineDet	-	123	115	18
	80%	25	31	76
	92%	10	16	154
	96%	5	12	228

* Latency is measured with single thread
** Throughput is measured on ZCU104 dual B4096 cores
*** Optimized models are also release in model zoo

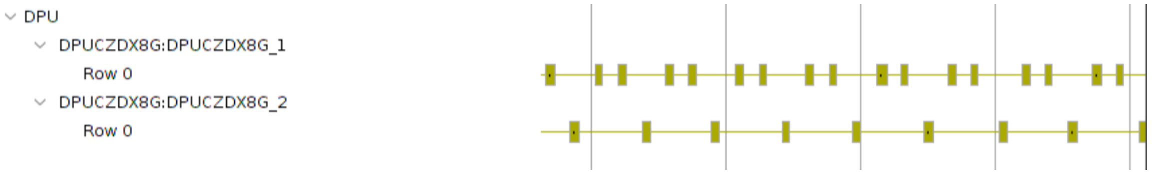
ML Enhanced Application Profiling

Pre-processing on ARM | Pruned model on DPU

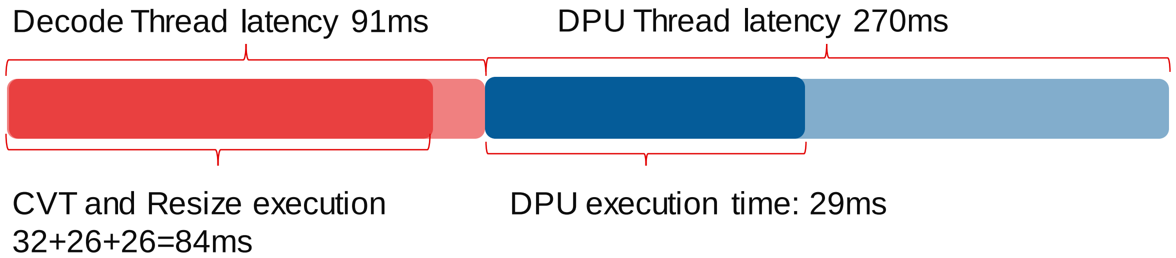
- ML enhanced application runs at 11.23FPS
  - DPU takes 30ms/frame and is mostly idle according to profiling result – not bottleneck anymore!
  - DecodeThread takes 91ms/frame, i.e., 11FPS – let’s boost it!

```
root@xilinx-zcu104-2020_1:~# vaitrace -c cfg.json
./usb_input_multi_threads_refinedet_drm refinedet_pruned_0_8 0 -t 3
Setting env
INFO:root:VART will run xmodel in [NORMAL] mode
……
INFO:root:Generating VTF
INFO:root:Overall FPS 11.23
```

Kernel	Compute Unit	Runs	Min Time (ms)	Avg Time (ms)	Max Time (ms)
subgraph_Elt3	DPUCZDX8G:DPUCZDX8G_1	218	27.016	29.143	49.077
subgraph_Elt3	DPUCZDX8G:DPUCZDX8G_2	109	27.351	29.527	50.886
cv::resize	CPU	655	0.073	26.475	74.147
cv::cvtColor	CPU	327	17.877	32.114	58.603
V4l2Capture::read_images	CPU	327	64.897	89.323	141.064
DecodeThread::run	CPU	327	65.036	91.455	635.478
GuiThread::run	CPU	327	50.010	91.338	500.399
DpuThread::run	CPU	331	173.527	272.118	500.090
vitis::ai::ConfigurableDpuTaskImp::setInputImageBGR_1	CPU	327	0.322	0.441	0.639
xir::XrtCu::run	CPU	327	27.064	29.540	51.002
vitis::ai::DpuTaskImp::run	CPU	327	27.720	30.410	51.680
vitis::ai::ConfigurableDpuTaskImp::run	CPU	327	27.737	30.436	51.697
vitis::ai::RefineDetImp::run	CPU	327	28.824	38.484	66.609
SortingThread::run_1	CPU	4,239	0.004	0.011	8.458
SortingThread::run	CPU	327	54.938	91.304	500.092



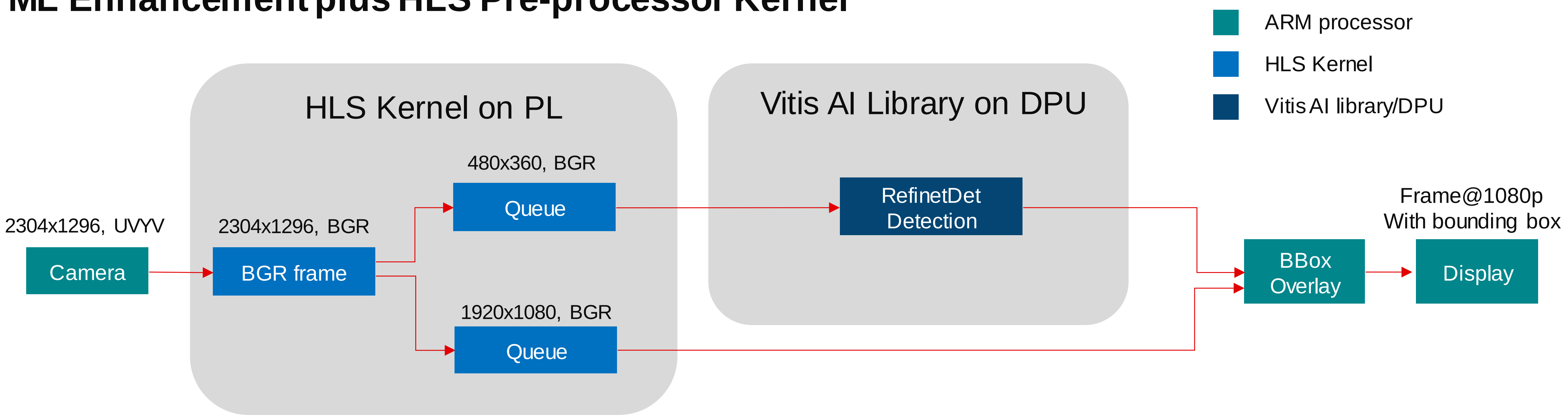
Thread name	Decode	DPU	SORT	GUI
Parallel Thread Num	1	3	1	1
Thread Latency(ms)	89	270	91	91



XILINX

Final Application

ML Enhancement plus HLS Pre-processor Kernel



- In the final implementation, HLS kernel will be used for image processing and will run on the programming logic
- ML inference will still run on DPU with very small portion of Vitis AI library process on the ARM (mean value subtraction in this case)

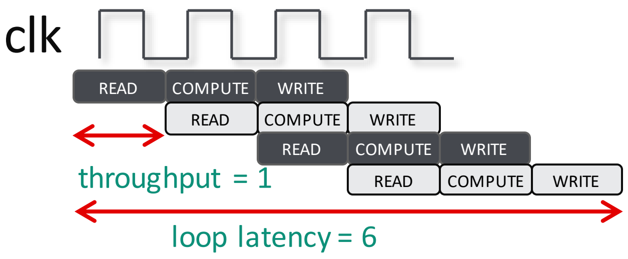
Pre-processor by HLS

- The pre-processor kernel is implemented based on Vitis Vision library pre-built functions
  - https://github.com/Xilinx/Vitis_Libraries/tree/master/vision
  - array2xfMat
  - uyvy2bgr
  - resize
  - xfMat2array
- It's straightforward to convert OpenCV function into PL accelerated xfOpenCV function

Kernel Design Optimization

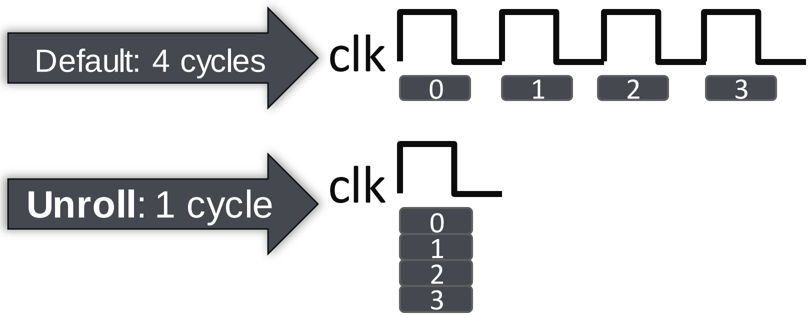
- Each FPGA kernel represents a single thread, so we leverage parallelism within that thread

Pipelining

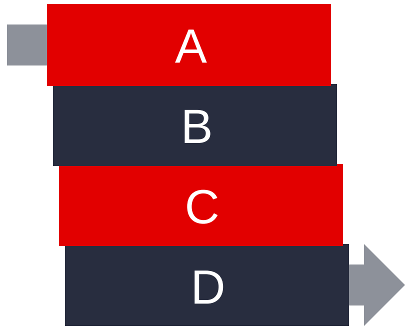


Loop Unrolling

```
void F (...)  {
...
add: for (i=0;i<=3;i++) {
  b = a[i] + b;
...
```



Dataflow Streaming



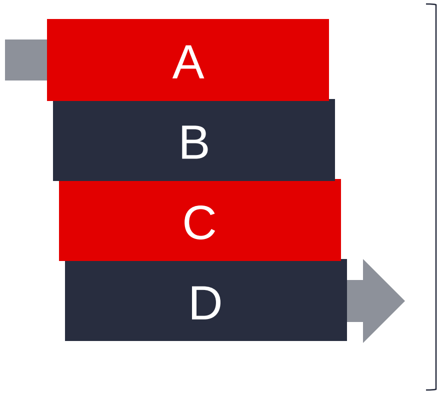
XILINX

Kernel Design Optimization

- Without Streaming

Latency = A + B + C + D + 8 * DDR Latency

- With Streaming



Latency = ~A + 2 * DDR Latency

And lower resource utilization, too!

XILINX.

HLS Kernel Definition

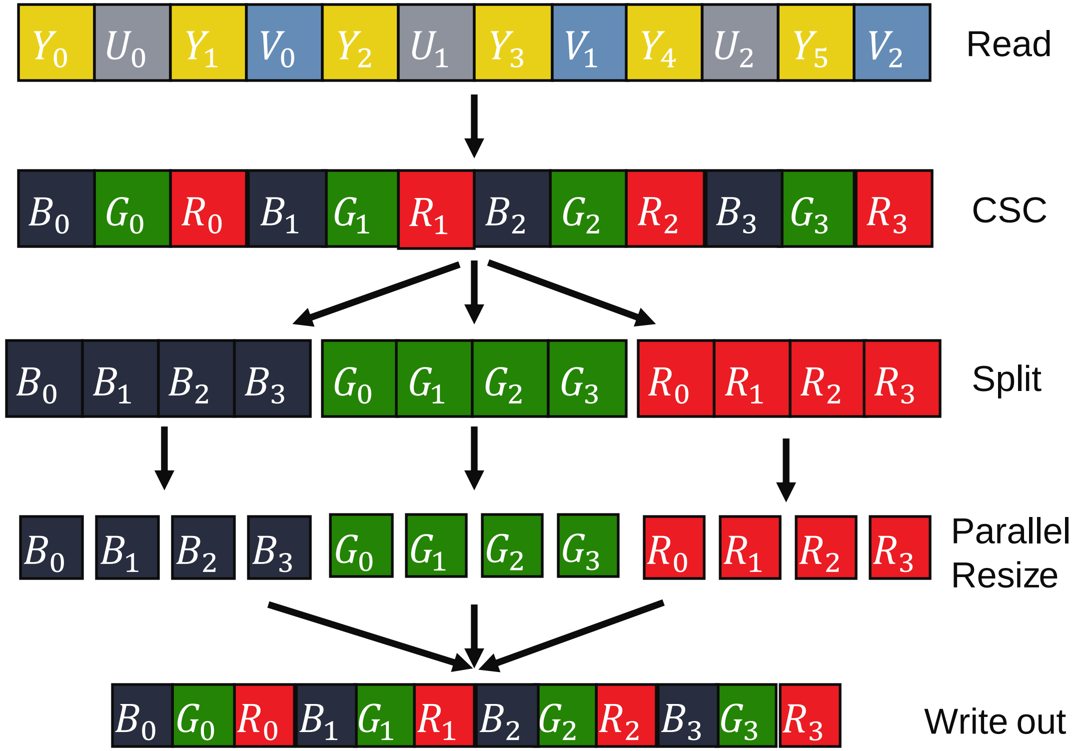
```
extern "C" {
void PREPROCESSOR(ap_uint<AXI_WIDTH> *image_in, ap_uint<AXI_WIDTH> *image_out,
        ap_uint<AXI_WIDTH> *image_out_full, int width_in, int height_in,
        int width_out, int height_out);
}
```

- image_in: The data from USB camera
- image_out: The resized image output for ML
- Image_out_full: The 1920x1080 image output for display
- Width_in: The input width of the usb camera
- Height_in: The input height of the usb camera
- Width_out: the width of the resized image
- Height_out: the height of the resized image

Kernel Implementation

- Implement preprocess kernel based on Vision library building blocks

```
// Read in data
xf::cv::Array2xfMat<AXI_WIDTH, XF_16UC1, MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC>(
            image_in, in_mat);
// Color Space Converter
xf::cv::uyvy2bgr<XF_16UC1, XF_8UC3, MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC>(
            in_mat, in_rgb);
// Duplicate the input image to get two images of different sizes
xf::cv::DuplicateMat_rpp<XF_8UC3, MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC>(in_rgb,
            in_rgb_copy0, in_rgb_copy1);
// Resize the first image
xf::cv::resize<XF_INTERPOLATION_AREA, XF_8UC3,MAX_IN_HEIGHT, MAX_IN_WIDTH,
    MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC, MAX_DOWN_SCALE>(in_rgb_copy0, out_rgb0);
// Resize the second image
xf::cv::resize<XF_INTERPOLATION_AREA, XF_8UC3, MAX_IN_HEIGHT, MAX_IN_WIDTH,
    MAX_IN_HEIGHT,  MAX_IN_WIDTH, NPC, MAX_DOWN_SCALE>(in_rgb_copy1, out_rgb1);
// Output the first image
xf::cv::xfMat2Array<AXI_WIDTH, XF_8UC3, MAX_OUT_HEIGHT, MAX_OUT_WIDTH, NPC>(
            out_rgb0, image_out);
// Output the second image
xf::cv::xfMat2Array<AXI_WIDTH, XF_8UC3, MAX_OUT_HEIGHT, MAX_OUT_WIDTH, NPC>(
            out_rgb1, image_out_full);
```



For complete design, please refer to [Vitis In-Depth Tutorial Machine Learning Introduction Module 7]

SW Function Migration

- Software migration from OpenCV to HLS kernel is not difficult with OpenCL API
  - Initialize
  - Allocate buffer
  - Load kernel
  - Set parameter
  - Move data to kernel
  - Execution
  - Get data from kennel

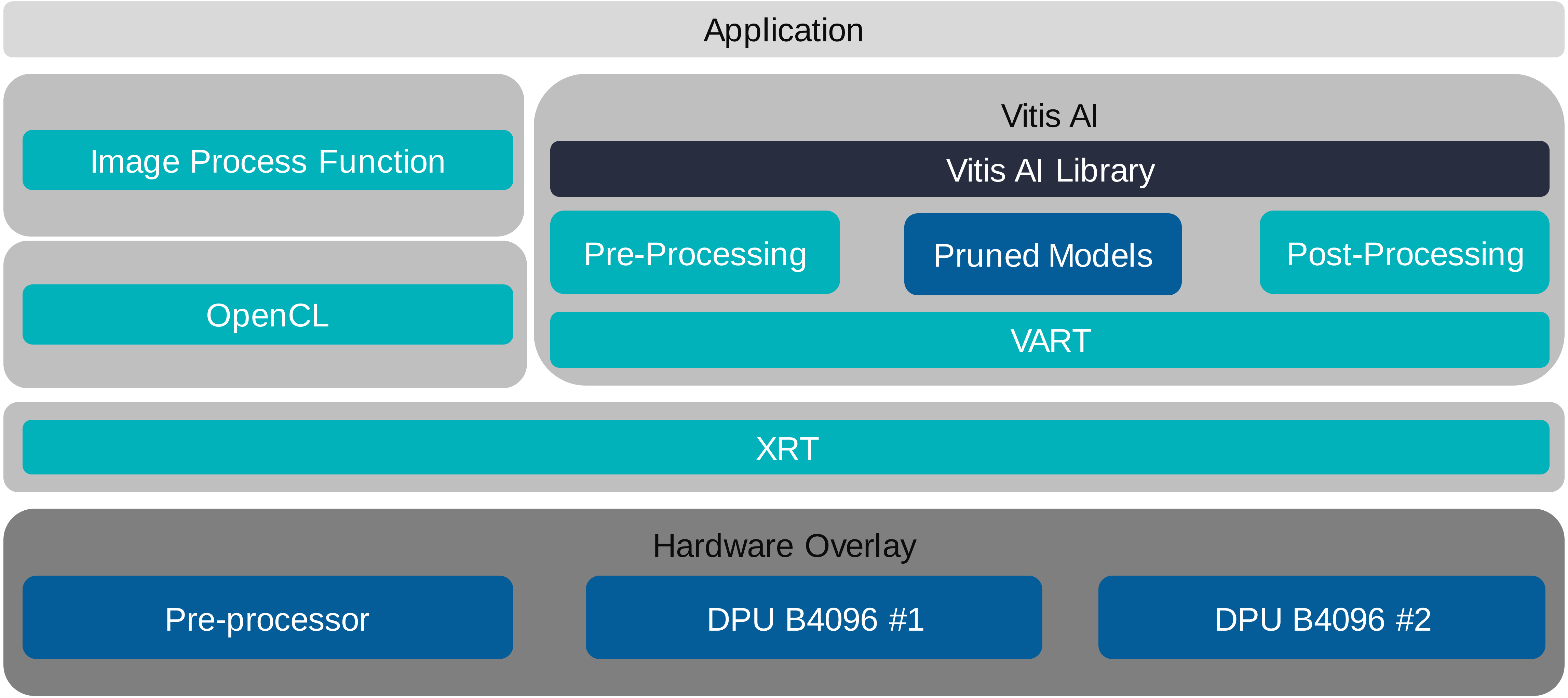
```
if (m_device->getFormat() == V4L2_PIX_FMT_UYVY)
{

    if (!xocl_initialized)
    {
        q = xocl.get_command_queue();
        imgToDevice = xocl.create_buffer(rsize, CL_MEM_READ_ONLY);
        resizeFromDevice = xocl.create_buffer(resize_size, CL_MEM_WRITE_ONLY);
        fullFromDevice = xocl.create_buffer(full_size, CL_MEM_WRITE_ONLY);
        krnl = xocl.get_kernel("pre_processor");
        krnl.setArg(0, imgToDevice);
        krnl.setArg(1, resizeFromDevice);
        krnl.setArg(2, fullFromDevice);
        krnl.setArg(3, IN_WIDTH);
        krnl.setArg(4, IN_HEIGHT);
        krnl.setArg(5, OUT_RESIZE_WIDTH);
        krnl.setArg(6, OUT_RESIZE_HEIGHT);
        xocl_initialized = true;
    }

    q.enqueueWriteBuffer(imgToDevice, CL_TRUE, 0, rsize, (void *)buffer);
    q.enqueueTask(krnl, NULL, &event_sp);
    clWaitForEvents(1, (const cl_event *)&event_sp);
    q.enqueueReadBuffer(resizeFromDevice, CL_TRUE, 0, resize_size, out_buf_0);
    q.enqueueReadBuffer(fullFromDevice, CL_TRUE, 0, full_size, out_buf_1);

    cv::Mat roi_mat0(OUT_RESIZE_HEIGHT, OUT_RESIZE_WIDTH, CV_8UC3, out_buf_0);
    cv::Mat roi_mat1(OUT_HEIGHT, OUT_WIDTH, CV_8UC3, out_buf_1);
    readImage.emplace_back(roi_mat1);
    readImage.emplace_back(roi_mat0);
    printf("DONE\n");
}
```

Final Design Architecture



HLS Kernel Performance

- HLS kernel execution information and time can be inspected in Vitis Analyzer
  - Average execution time per frame is about 17ms, i.e., 58FPS
  - One preprocess kernel is capable of handling camera input in real time
    - Part of time will even be idle because of waiting for camera data

Top Kernel Execution

Kernel	Kernel Instance Address	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)
pre_processor	0xffff90001a80	0	0	edge-0	3244.520	22.799
pre_processor	0xffff90001a80	0	0	edge-0	1741.750	20.827
pre_processor	0xffff90001a80	0	0	edge-0	3074.300	19.727
pre_processor	0xffff90001a80	0	0	edge-0	2615.810	19.370
pre_processor	0xffff90001a80	0	0	edge-0	1007.410	19.329
pre_processor	0xffff90001a80	0	0	edge-0	1873.760	17.732
pre_processor	0xffff90001a80	0	0	edge-0	1452.770	17.246
pre_processor	0xffff90001a80	0	0	edge-0	1503.730	14.916
pre_processor	0xffff90001a80	0	0	edge-0	1654.430	13.739
pre_processor	0xffff90001a80	0	0	edge-0	1085.610	11.387

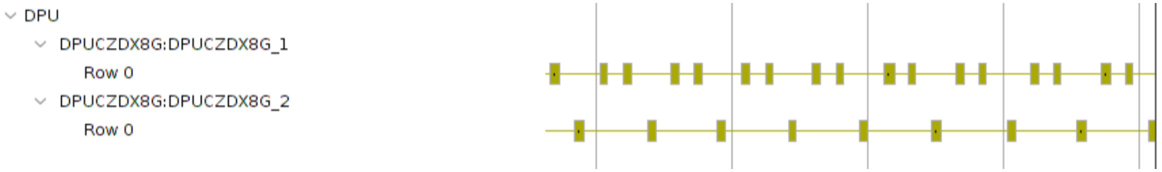
Final Application Profiling

Pre-processing on PL | Pruned model on DPU

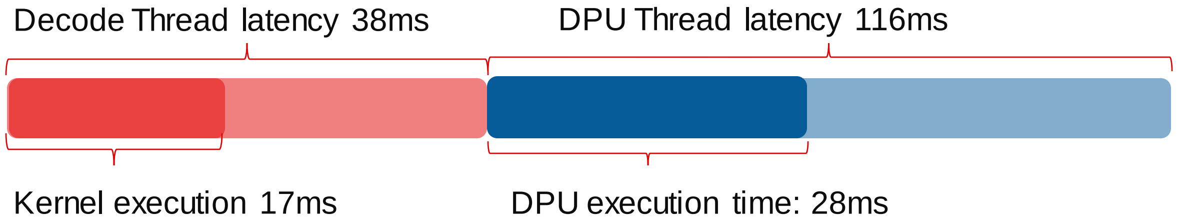
- Final application runs at 26FPS in profiling mode
  - DPU core takes 28ms/frame, from profiling result cores are idle in most time – not bottleneck!
  - DecodeThread shrinks to 38ms/frame, i.e., 26.3FPS
  - Profiling mode slightly affects overall performance

```
root@xilinx-zcu104-2020_1:~# vaitrace -c cfg.json
./usb_input_multi_threads_refinedet_hls_drm refinedet_pruned_0_8 0 -t 3
Setting env
INFO:root:VART will run xmodel in [NORMAL] mode
……
INFO:root:Generating VTF
INFO:root:Overall FPS 26.02
```

Kernel	Compute Unit	Runs	Min Time (ms)	Avg Time (ms)	Max Time (ms)
subgraph_Elt3	DPUCZDX8G:DPUCZDX8G_1	151	27.088	28.380	49.254
subgraph_Elt3	DPUCZDX8G:DPUCZDX8G_2	76	27.163	28.539	34.531
cv::resize	CPU	1	0.392	0.392	0.392
read_images_with_kernel	CPU	227	17.602	37.631	62.979
DecodeThread::run	CPU	227	17.736	38.159	64.056
GuiThread::run	CPU	205	6.248	42.316	114.259
vitis::ai::ConfigurableDpuTaskImp::setInputImageBGR_1	CPU	227	0.316	0.660	10.839
xir::XrtCu::run	CPU	227	27.148	28.670	49.378
vitis::ai::DpuTaskImp::run	CPU	227	27.821	30.337	57.506
vitis::ai::ConfigurableDpuTaskImp::run	CPU	227	27.837	30.366	57.529
vitis::ai::RefineDetImp::run	CPU	227	28.879	32.425	58.849
DpuThread::run	CPU	228	29.494	116.298	500.090
SortingThread::run_1	CPU	2,938	0.004	0.007	0.749
SortingThread::run_2	CPU	452	0.004	0.008	0.840
SortingThread::run	CPU	226	2.639	38.379	98.270



Thread name	Decode	DPU	SORT	GUI
Parallel Thread Num	1	3	1	1
Thread Latency(ms)	38	116	38	42



XILINX.

Final Application Performance

Pre-processing on PL

Pruned model on DPU

- Final application runs at 30FPS and achieve the target

```
root@xilinx-zcu104-2020_1: ./usb_input_multi_threads_refinedet_hls_drm refinedet_pruned_0_8 0 -t 3
Setting env
INFO:root:VART will run xmodel in [NORMAL] mode
……
I1111 05:07:30.497097 3812 guithread.cpp:101] screen [1920 x 1080]; r = [1920 x 1080 from (0, 0)]
I1111 05:07:30.497117 3812 dpdrm.hpp:563] fb_size [1920 x 1080] fb_roi [1920 x 1080 from (0, 0)] image_size [1920 x 1080] image_roi [1920 x 1080 from (0, 0)]
I1111 05:07:30.497143 3812 dpdrm.hpp:569] from = [1920 x 1080]
I1111 05:07:30.497645 3812 dpdrm.hpp:571] to = [1920 x 1080]
I1111 05:07:30.512097 3811 hlsV4l2Capture.cpp:224] OpenCL duration:27
I1111 05:07:30.512209 3811 decodethread.cpp:61] Decode and Resize :31ms
I1111 05:07:30.521747 3808 mythread.cpp:90] thread [DedodeThread-0] is stopped.
I1111 05:07:30.521888 3808 mythread.cpp:90] thread [GUIThread] is stopped.
I1111 05:07:30.521935 3808 mythread.cpp:90] thread [DPU-0] is stopped.
I1111 05:07:30.521953 3808 mythread.cpp:90] thread [DPU-1] is stopped.
I1111 05:07:30.521970 3808 mythread.cpp:90] thread [DPU-2] is stopped.
I1111 05:07:30.521984 3808 mythread.cpp:90] thread [SORT-0] is stopped.
I1111 05:07:30.522648 3812 guithread.cpp:133] Gui duration :27ms
I1111 05:07:30.523404 3815 dputhread.cpp:48] dpu queue size 0
I1111 05:07:30.523440 3815 dputhread.cpp:56] DPU in single thread duration :96ms
I1111 05:07:30.523458 3815 mythread.cpp:68] thread [DPU-2] is ended
I1111 05:07:30.523782 3816 sortthread.cpp:58] Sort thread duration : 2025 ms
I1111 05:07:30.524152 3816 sortthread.cpp:73] thread [SORT-0] frame id 243 sorting queue size 0 FPS: 30.1235
I1111 05:07:30.524201 3816 mythread.cpp:68] thread [SORT-0] is ended
I1111 05:07:30.545967 3811 hlsV4l2Capture.cpp:224] OpenCL duration:28
DONE
```

Reference

- https://github.com/Xilinx/Vitis-AI
- https://github.com/Xilinx/Vitis_Libraries
- https://github.com/Xilinx/Vitis-In-Depth-Tutorial
  - Introduction 03-Basic contains full design used in this presentation
- https://github.com/Xilinx/Vitis_Embedded_Platform_Source

Summary

- Vitis provides unified development environment across all platforms and enables hardware development in a software way
- Vitis AI provides whole stack AI inference acceleration solution, including model optimization, toolchain and high-efficiency DPU processor
- Vitis library could help to accelerate pre/post-process components in the system and boost whole application performance.

Thank You