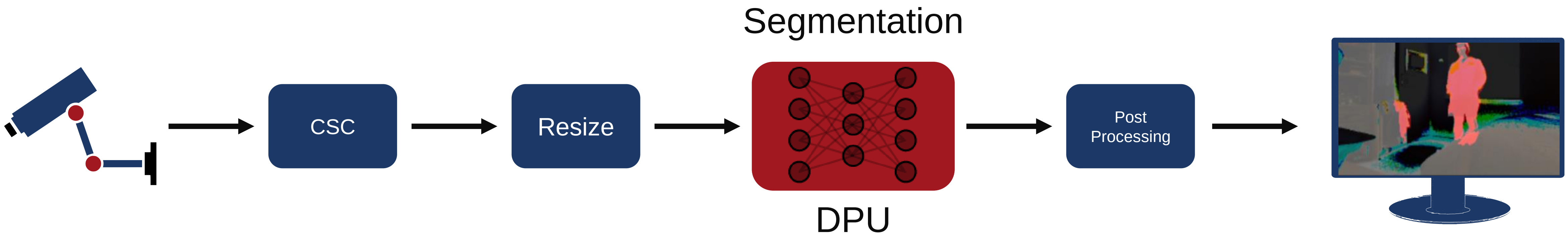
Whole Application Acceleration
Designing an AI-enabled System

Alvin Clark
Sr. Technical Marketing Engineer

Design Overview

› Design captures 640x480 camera data @ 60FPS, runs neural network inference, and displays the results on a monitor.



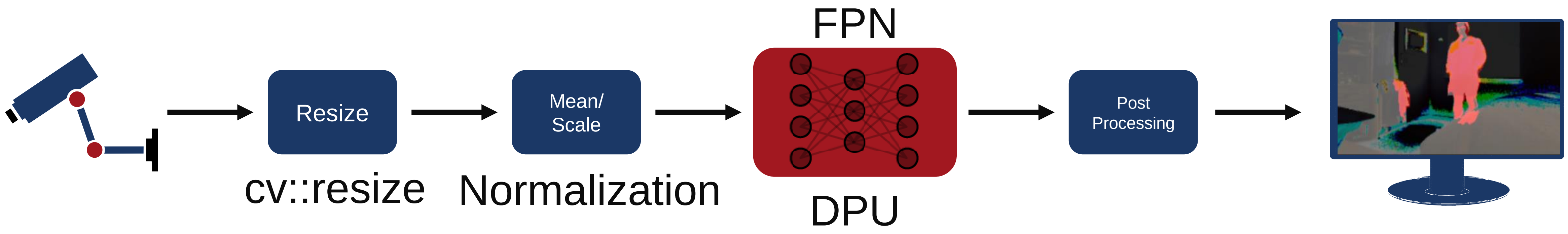
› The neural network has been trained to accept RGB images at 512x256

› The camera outputs UYVY video format, but this is converted to RGB in the platform

› The design must resize the RGB video

› Mean values must be subtracted from each pixel and a scale factor of 0.5 must be applied for input normalization

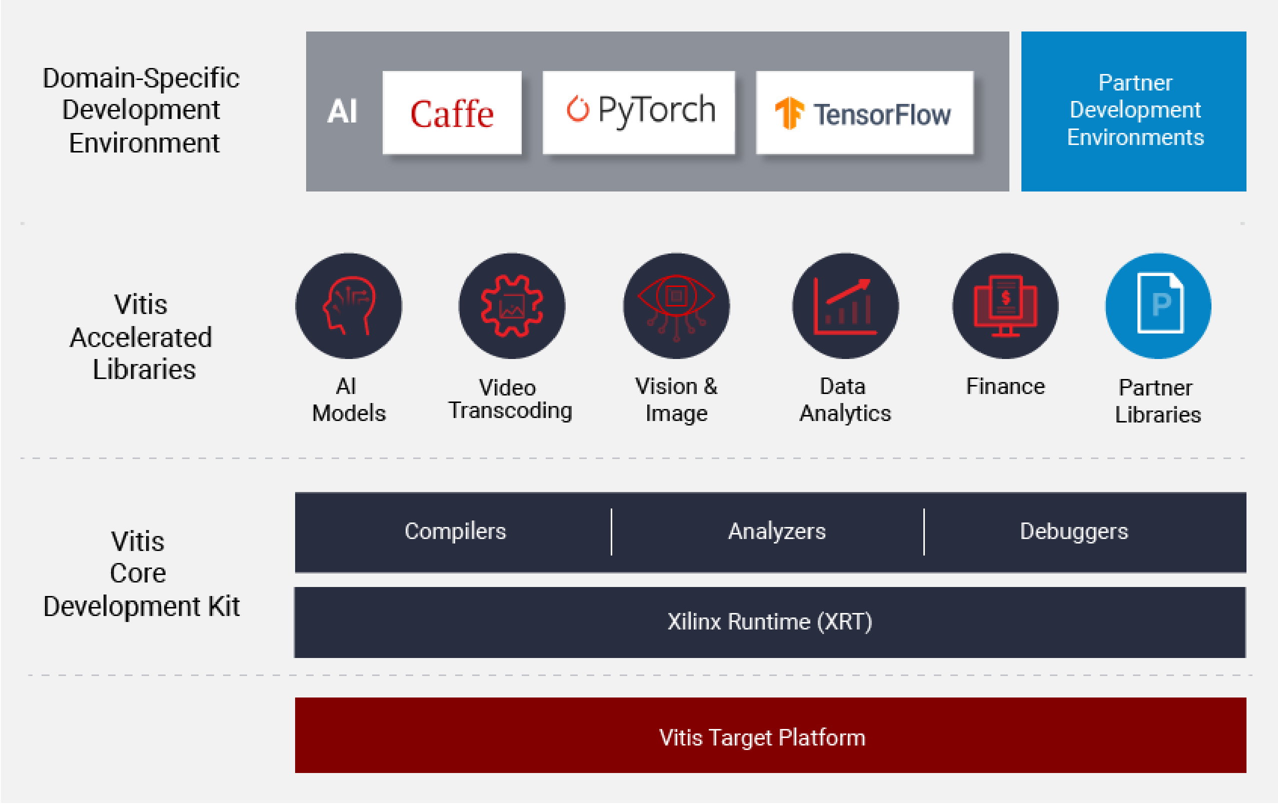
Base Performance



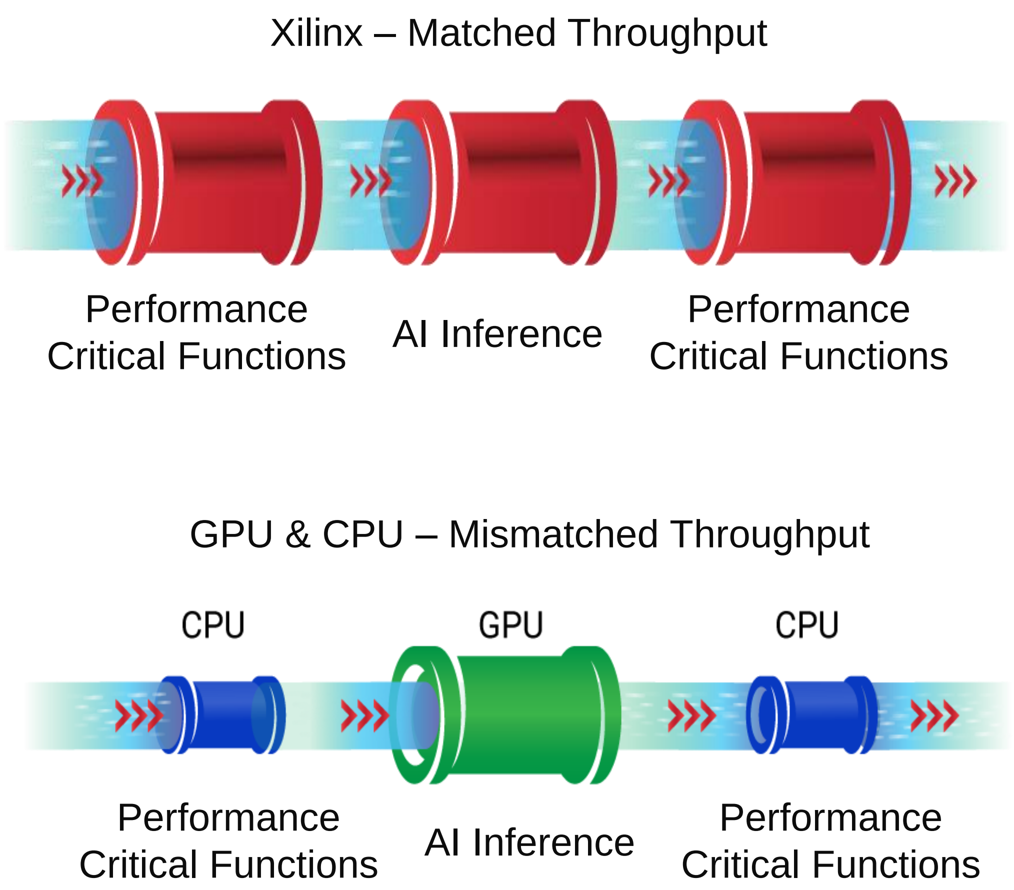
Operation	Time	Interpretation
OpenCV Resize	6.03 ms	Resizing, 640x480 down to 512x256
Normalization + Set Input Image	9.54 ms	Mean value subtraction and scale factor application, processed in software using Neon by Vitis AI Libraries and Copy input image into the DPU's buffers
DPU Run Task	47.26 ms	Actual ML processing time in the B1152F DPU
PostProcessing	3.950 ms	argmax output layers and overlays, processed in software
Total:	66.81 ms	Total Latency

Accelerating the Whole Application: The Xilinx Advantage

World-Class SW Acceleration Tools and Libraries

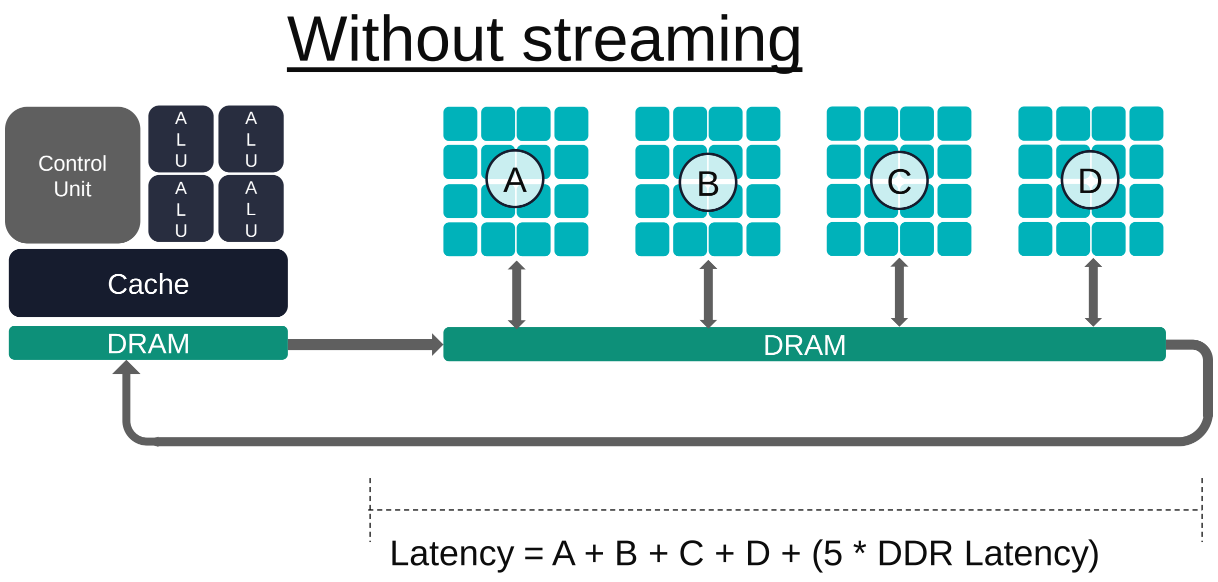


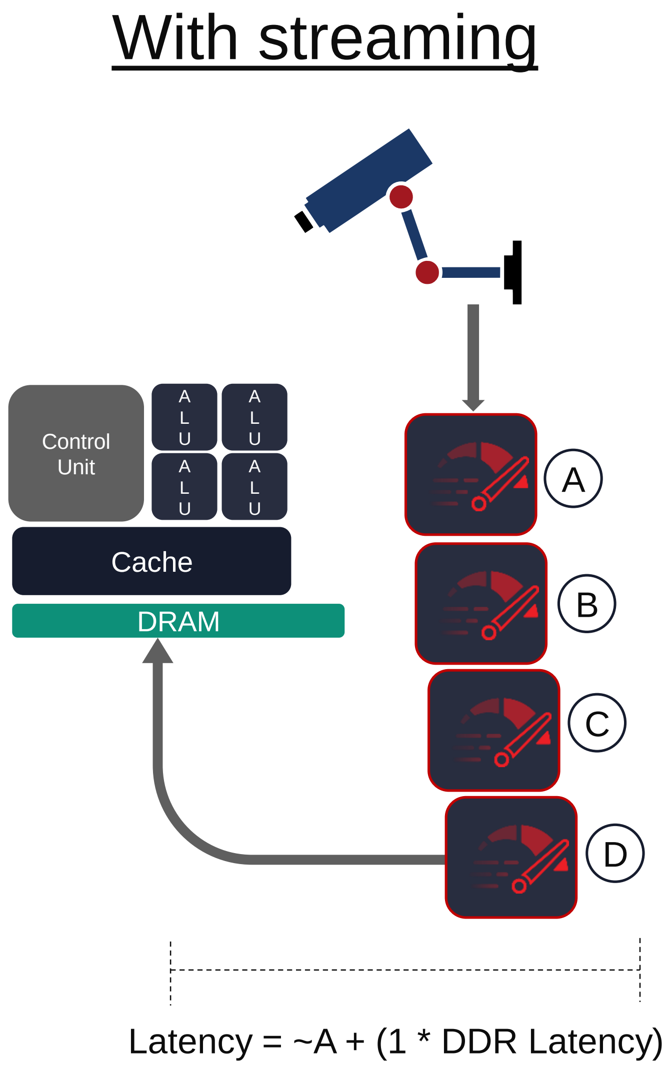
I/O and Memory Flexible Devices



XILINX

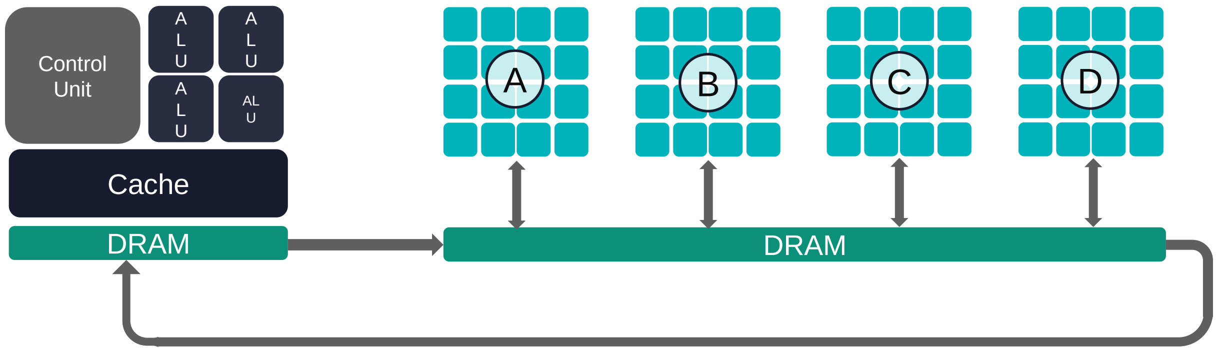
Data Streaming





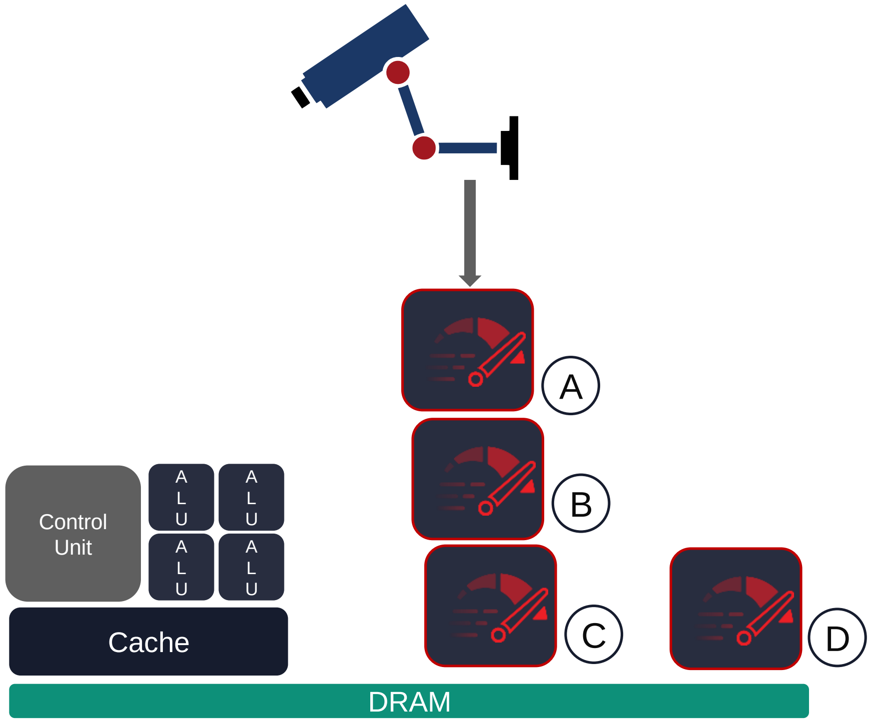
Data Streaming

Without streaming



Latency = A + B + C + D + (5 * DDR Latency)

With streaming and mm



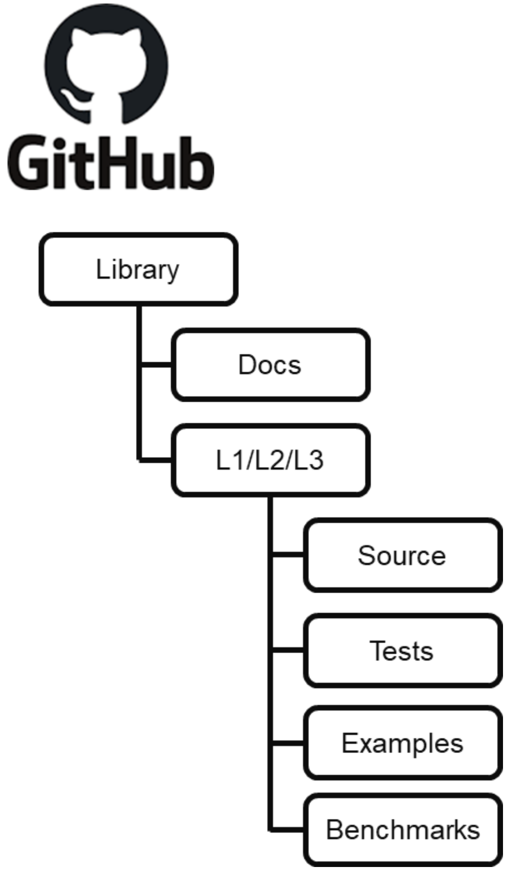
Latency = ~A + D + (3 * DDR Latency)

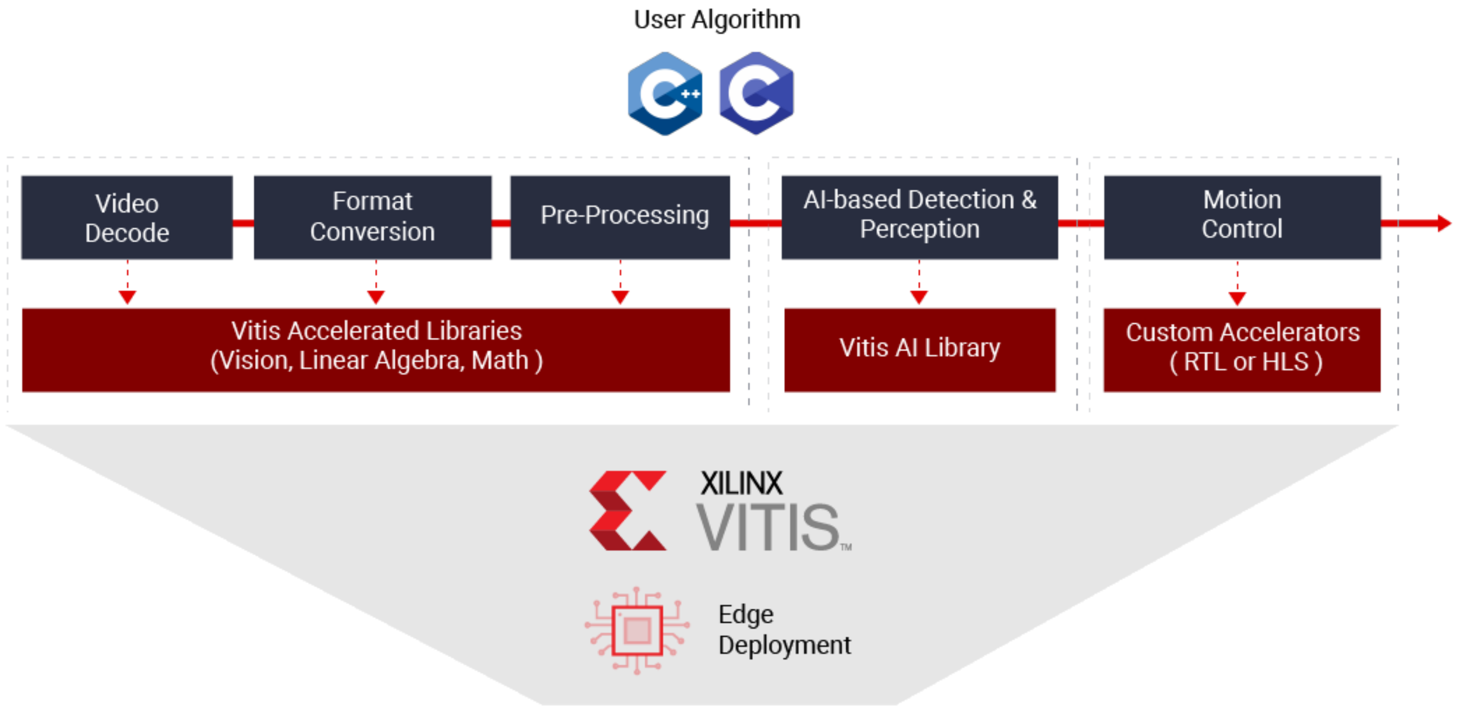
XILINX

Vitis Vision Libraries

https://github.com/Xilinx/Vitis_Libraries/tree/master/vision (Libraries)

https://xilinx.github.io/Vitis_Libraries/vision/ (User Guide)

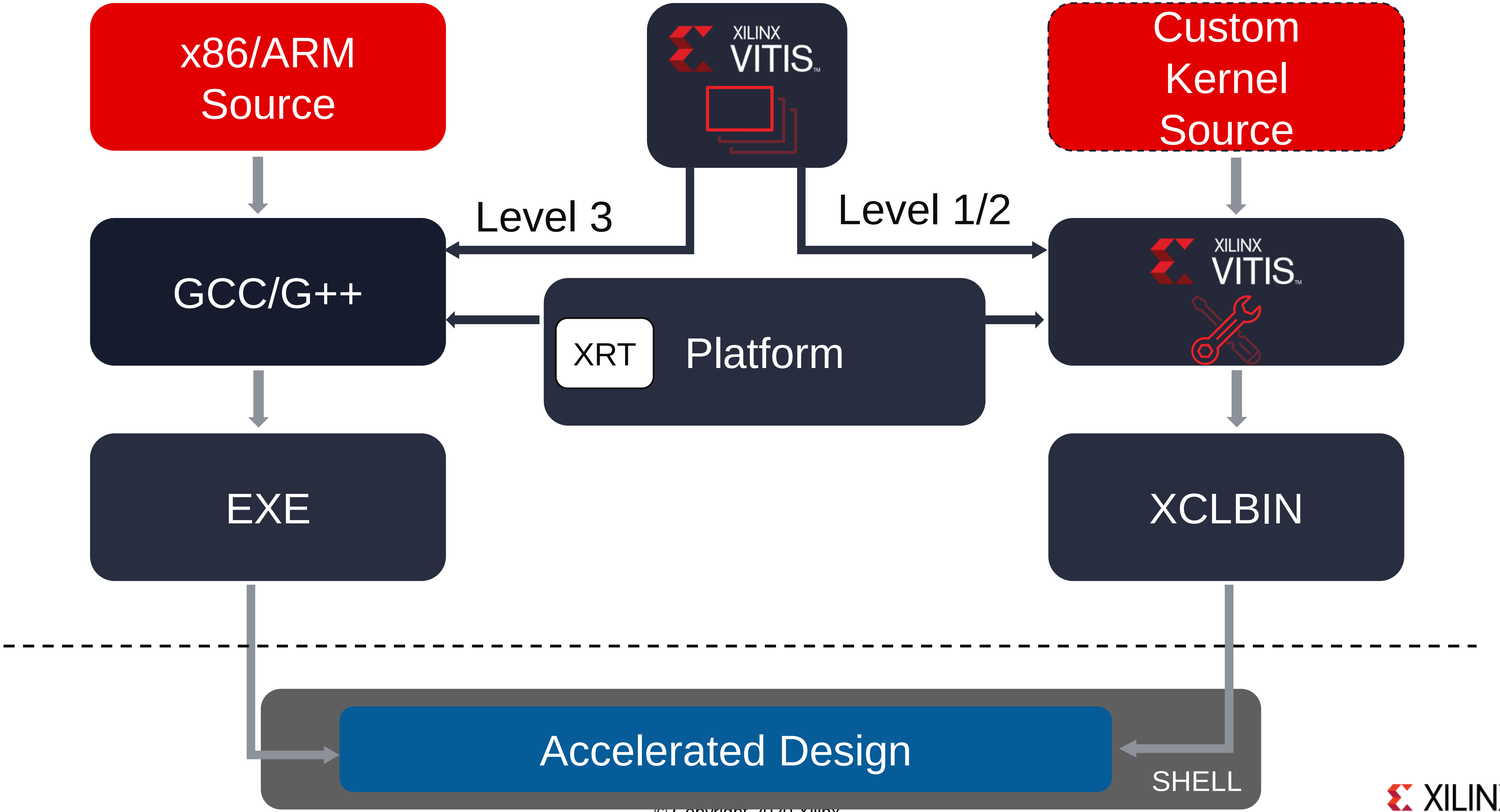




Vitis Vision Libraries (xf::cv::)

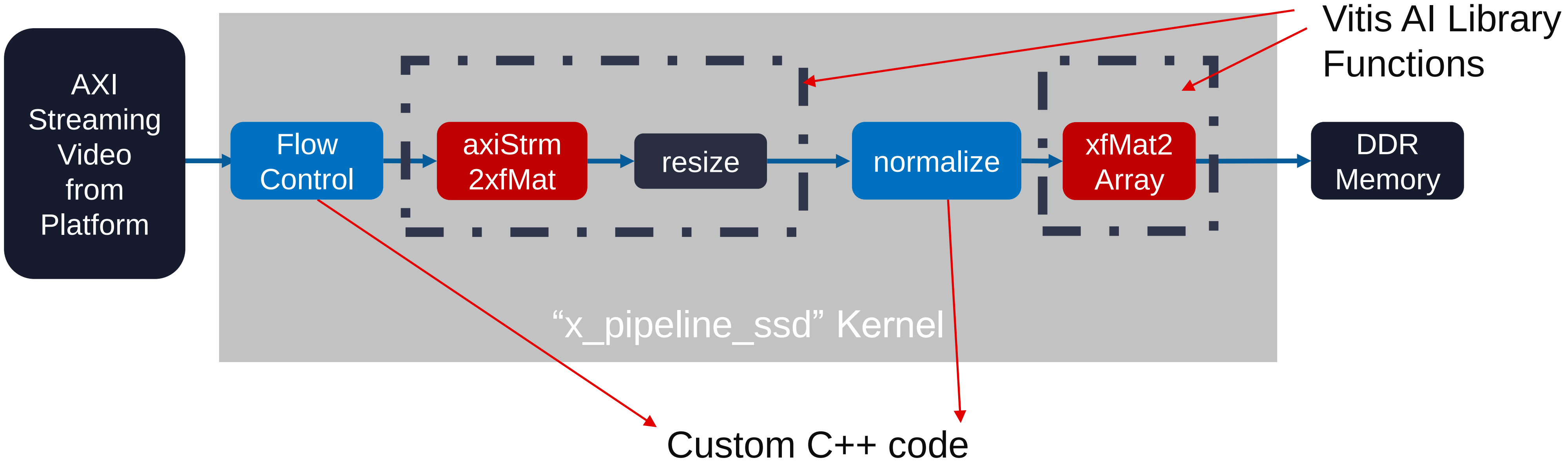
Absolute Difference	Delay	Mean and Standard Deviation	Sum
Accumulate	Demosaicing	Max	SVM
Accumulate Squared	Dilate	MaxS	Thresholding
Accumulate Weighted	Duplicate	Median Blur Filter	Atan2
AddS	Erode	Min	Inverse (Reciprocal)
Addweighted	FAST Corner Detection	MinS	Look Up Table
Autowhitebalance	Gaincontrol	MinMax Location	Square Root
Badpixelcorrection	Gammacorrection	Mean Shift Tracking	WarpTransform
Bilateral Filter	Gaussian Filter	Otsu Threshold	Zero
Bit Depth Conversion	Gradient Magnitude	Paintmask	
Bitwise AND	Gradient Phase	Pixel-Wise Addition	
Bitwise NOT	Harris Corner Detection	Pixel-Wise Multiplication	
Bitwise OR	Histogram Computation	Pixel-Wise Subtraction	
Bitwise XOR	Histogram Equalization	Reduce	
Box Filter	HOG	Remap	
BoundingBox	HoughLines	Resolution Conversion (Resize)	
Canny Edge Detection	Preprocessing for Deep Neural Networks	BGR2HSV	
Channel Combine	Pyramid Up	convertScaleAbs	
Channel Extract	Pyramid Down	Scharr Filter	
Color Conversion	InitUndistortRectifyMapInverse	Set	
Color Thresholding	InRange	Sobel Filter	
Compare	Integral Image	Semi Global Method for Stereo Disparity Estimation	
CompareS	Dense Pyramidal LK Optical Flow	Stereo Local Block Matching	
Crop	Dense Non-Pyramidal LK Optical Flow	SubRS	
Custom Convolution	Kalman Filter	SubS	

Design Flows



Hierarchical Accelerator: Built for composability/accessibility

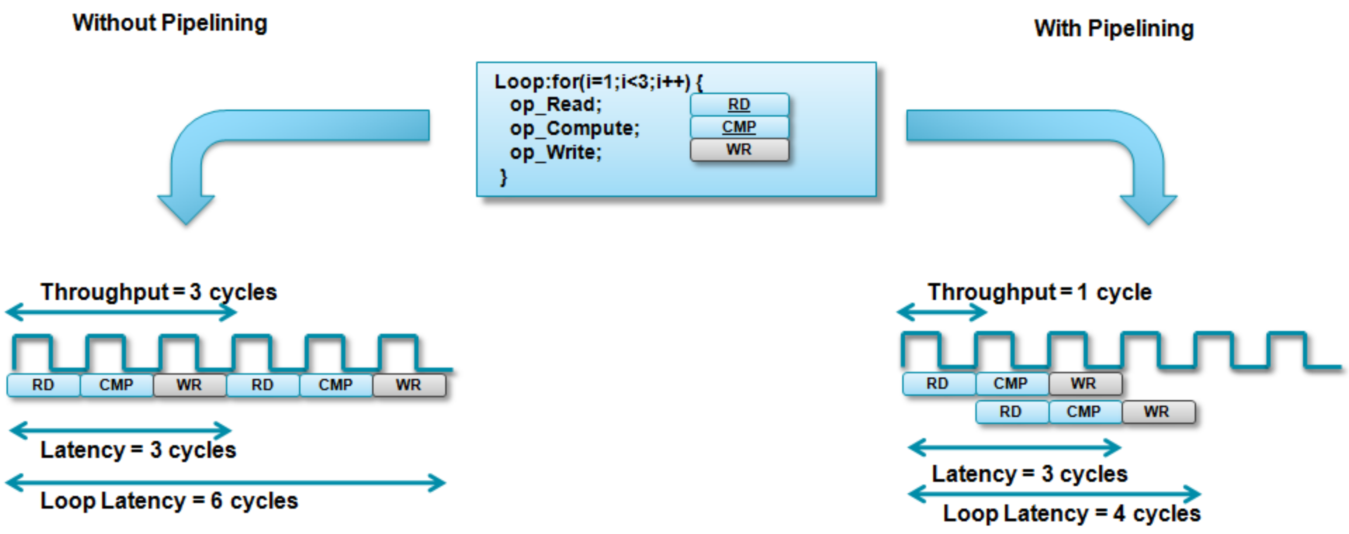
Function Prototypes
x_pipeline_ssd(image_in, image_out, width_in, height_in, width_out, height_out, use_mean, mean, scale)
flow_control(axi_stream_image_in, axis_video_in_synced, height, width)
xf::cv::axiStrm2xfMat(axis_video_in_synced, in_mat)
xf::cv::resize(xf::cv::Mat src, xf::cv::Mat out_rgb)
image_normalize(out_rgb, out_mat, use_mean, scale_r, scale_g, scale_b, mean_r, mean_g, mean_b)
xf::cv::xfMat2Array(xf::cv::Mat out_mat, image_out)



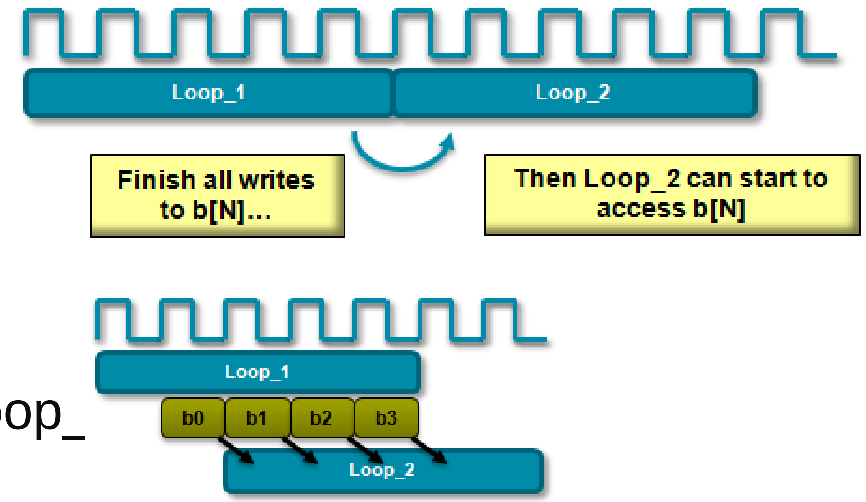
XILINX.

Vitis HLS: Key Pragmas

- Pipelining:




```
int a[N], b[N], c[N];

Loop_1: for (i=0;i<=N-1;i++) {
    b[i] = a[i] + in1;
}

Loop_2: for (i=0;i<=N-1;i++) {
    c[i] = b[i] * in2;
}
```



- Dataflow:
  - Arrays are changed to FIFOs to allow concurrent execution of Loop_1 and Loop_

Code Snippet

Top Level Function Definition

Interface definitions

Dataflow the processing

Internal streaming variable declarations

Synchronize to start of frame

Convert from axi stream to xf::Mat

Resize the image

Subtract the mean values and apply scale

Convert from xf::Mat to memory mapped interface

```
void x_pipeline_ssd(hls::stream<ap_axiu<24, 3, 1, 1> >& image_in,
                    ap_uint<AXI_WIDTH> *image_out,
                    int width_in,  int height_in, int width_out,
                    int height_out, int use_mean,  float scale_r,
                    float scale_g, float scale_b, unsigned char mean_r,
                    unsigned char mean_g, unsigned char mean_b)
{
#pragma HLS INTERFACE axis port = image_in
#pragma HLS INTERFACE m_axi port = image_out offset = slave bundle = image_out_gmem depth = 131072
#pragma HLS INTERFACE s_axilite port = image_out bundle = control
#pragma HLS INTERFACE s_axilite port = width_in bundle = control
#pragma HLS INTERFACE s_axilite port = height_in bundle = control
#pragma HLS INTERFACE s_axilite port = width_out bundle = control
#pragma HLS INTERFACE s_axilite port = height_out bundle = control
#pragma HLS INTERFACE s_axilite port = use_mean bundle = control
#pragma HLS INTERFACE s_axilite port = scale_r bundle = control
#pragma HLS INTERFACE s_axilite port = scale_g bundle = control
#pragma HLS INTERFACE s_axilite port = scale_b bundle = control
#pragma HLS INTERFACE s_axilite port = mean_r bundle = control
#pragma HLS INTERFACE s_axilite port = mean_g bundle = control
#pragma HLS INTERFACE s_axilite port = mean_b bundle = control
#pragma HLS INTERFACE s_axilite port = return bundle = control

#pragma HLS DATAFLOW

    hls::stream<ap_axiu<24, 0, 0, 0> > px_in_synced;
    xf::cv::Mat<XF_8UC3, MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC> in_mat(height_in, width_in);
    #pragma HLS STREAM variable=in_mat.data depth=256 dim=1
    xf::cv::Mat<XF_8UC3, MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC> in_rgb(height_in, width_in);
    #pragma HLS STREAM variable=in_rgb.data depth=256 dim=1
    xf::cv::Mat<XF_8UC3, MAX_OUT_HEIGHT, MAX_OUT_WIDTH, NPC> out_rgb(height_out, width_out);
    #pragma HLS STREAM variable=out_rgb.data depth=256 dim=1
    xf::cv::Mat<XF_8UC3, MAX_OUT_HEIGHT, MAX_OUT_WIDTH, NPC> out_mat(height_out, width_out);
    #pragma HLS STREAM variable=out_mat.data depth=256 dim=1

    flow_control(image_in, px_in_synced, height_in, width_in);

    xf::cv::axiStrm2xfMat<24, XF_8UC3, MAX_IN_HEIGHT, MAX_IN_WIDTH, NPC>(px_in_synced,in_mat);

    xf::cv::resize<XF_INTERPOLATION_NN,
              XF_8UC3,
              MAX_IN_HEIGHT,
              MAX_IN_WIDTH,
              MAX_OUT_HEIGHT,
              MAX_OUT_WIDTH,
              NPC,
              MAX_DOWN_SCALE>(in_mat, out_rgb);
    image_normalize(out_rgb, out_mat, use_mean, scale_r, scale_g, scale_b, mean_r, mean_g, mean_b);

    xf::cv::xfMat2Array<AXI_WIDTH, XF_8UC3, MAX_OUT_HEIGHT, MAX_OUT_WIDTH, NPC>(out_mat, image_out);
  }
}
```


XILINX

ML+X Design Example Results

- Clock Frequency: set to 250 MHz
- HLS determines max latency is 320,002 cycles which is 12,802 cycles added onto the 640x480 number of clocks (307,200)
- At 250MHz this is 51.208us
- Operates in parallel with DPU to increase throughput

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.50 ns	2.555 ns	0.94 ns

Latency

Summary

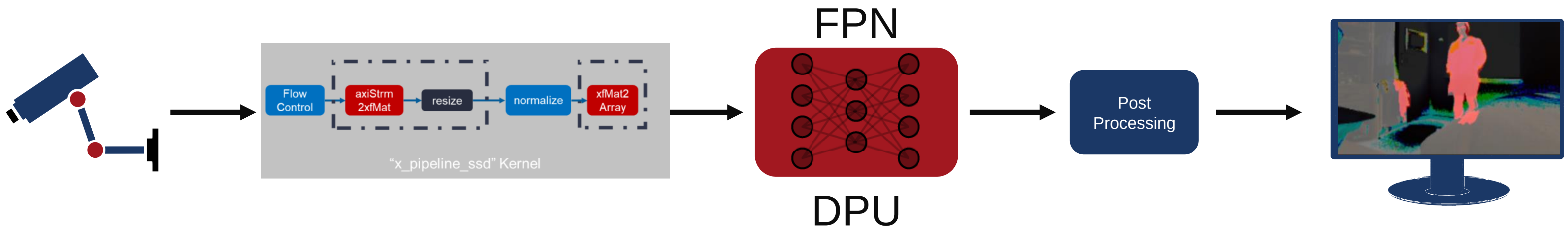
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
308647	320002	1.080 ms	1.120 ms	308644	319984	dataflow

Detail

Instance

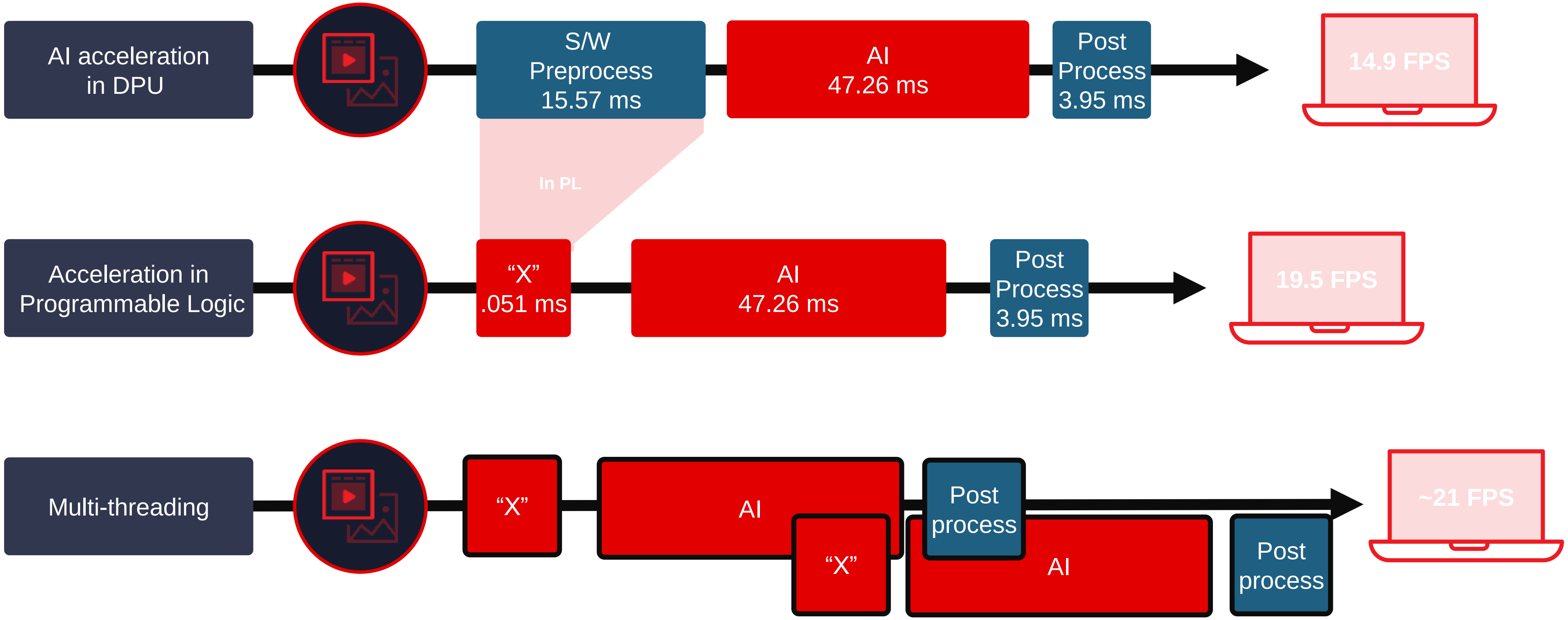
		Latency (cycles)		Latency (absolute)		Interval (cycles)		
Instance	Module	min	max	min	max	min	max	Type
xfMat2Array_128_9_256_512_1_U0	xfMat2Array_128_9_256_512_1_s	19	131081	66.500 ns	0.459 ms	19	131081	none
resize_0_9_480_640_256_512_1_2_U0	resize_0_9_480_640_256_512_1_2_s	169	319983	0.592 us	1.120 ms	169	319983	none
axiStrm2xfMat_24_9_480_640_1_U0	axiStrm2xfMat_24_9_480_640_1_s	20	307219	70.000 ns	1.075 ms	20	307219	none
Loop_1_proc_U0	Loop_1_proc	7	131085	24.500 ns	0.459 ms	7	131085	none
Block_Mat_exit74_proc25339_U0	Block_Mat_exit74_proc25339	3	3	10.500 ns	10.500 ns	3	3	none
flow_control_U0	flow_control	308643	308643	1.080 ms	1.080 ms	308643	308643	none

Accelerated Performance



Operation	Original Latency	Accelerated Latency	Interpretation
X_pipeline_ssd	6.03 ms	0.051 ms	Streaming accelerator resize, normalize, and copy to DDR.
Set Input Image	9.54 ms	5.63 ms	Copy input image into the DPU's buffers
DPU Run Task	47.26 ms	47.26 ms	Actual ML processing time in the B1152F DPU
PostProcessing	3.950 ms	3.950 ms	argmax output layers and overlays, processed in software
Total:	66.81 ms	56.891 ms	Total Latency

Adaptive Architecture for System Acceleration



Xilinx Runtime: System Management

- Find Xilinx devices
- Create processing Queue
- Load kernel to adaptable resources

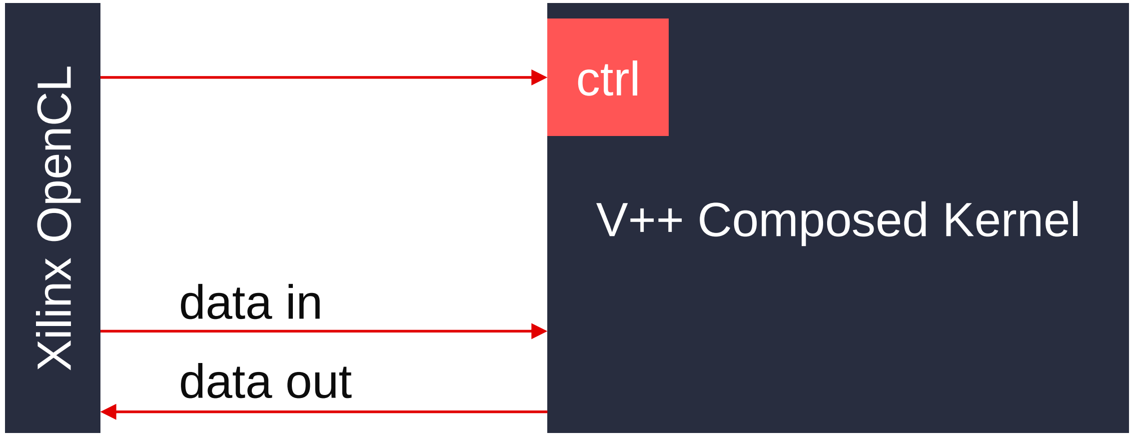
```
// get_xil_devices() is a utility API which will find the xilinx
// platforms and will return list of devices connected to Xilinx platform
auto devices = xcl::get_xil_devices();

// Selecting the first available Xilinx device
device = devices[0];
auto platform_id = device.getInfo<CL_DEVICE_PLATFORM>(&err);

//Initialization of streaming class is needed before using it.
xcl::Stream::init(platform_id);

// Creating Context
OCL_CHECK(err, context = cl::Context(device, NULL, NULL, NULL, &err));

// Creating Command Queue
OCL_CHECK(
    err,
    q = cl::CommandQueue(context, device, CL_QUEUE_PROFILING_ENABLE, &err));
```

```
// Creating Program
OCL_CHECK(err, program = cl::Program(context, devices, bins, NULL, &err));

// Creating Kernel
OCL_CHECK(err, krnl_vadd = cl::Kernel(program, "krnl_stream_vadd", &err));
```

```
// Launch the Kernel
cl::Event b_wait_event;
OCL_CHECK(err, err = q.enqueueTask(krnl_vadd, NULL, &b_wait_event));
```

Xilinx Runtime: Data and Control

- XRT abstracts adaptable acceleration to standard OpenCL interface
- Set runtime parameters via AXI-Lite interface
- Control Host → Accelerator data transactions



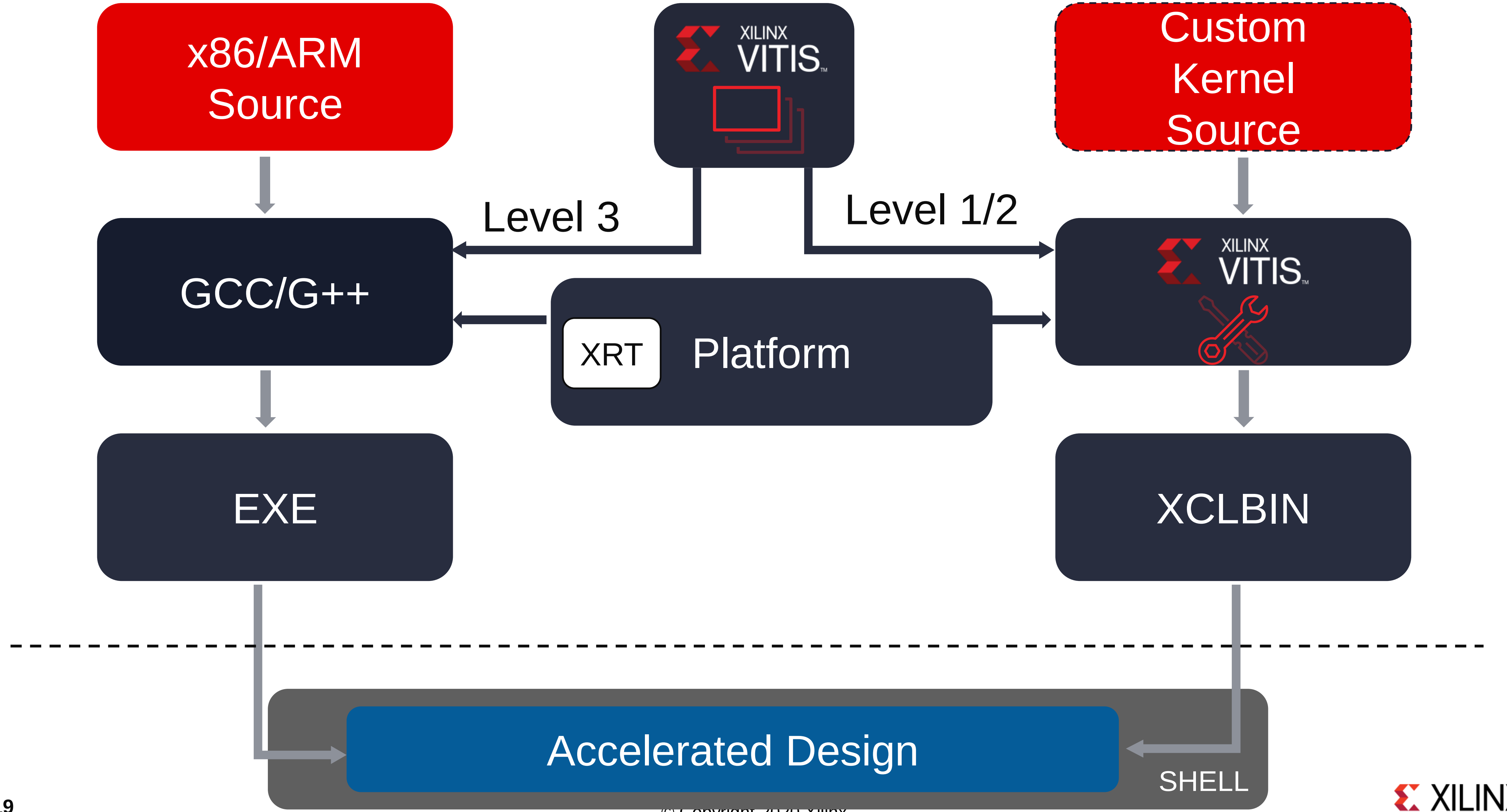
Kernel Snapshot

```
void vadd(const unsigned int *in1, // Read-Only Vector 1
          const unsigned int *in2, // Read-Only Vector 2
          unsigned int *out_r,     // Output Result
          int size                 // Size in integer
) {
```

```
    OCL_CHECK(err, err = krnl_vector_add.setArg(0, buffer_in1));
    OCL_CHECK(err, err = krnl_vector_add.setArg(1, buffer_in2));
    OCL_CHECK(err, err = krnl_vector_add.setArg(2, buffer_output));
    OCL_CHECK(err, err = krnl_vector_add.setArg(3, size));

    // Copy input data to device global memory
    OCL_CHECK(err,
              err = q.enqueueMigrateMemObjects({buffer_in1, buffer_in2},
                                               0 /* 0 means from host*/));
```

Developing an Application with AI Library

Design Flows



AI Libraries: Unified API Interface

Get an instance of derived class

Get width and height for required by Algorithm

DPU run and get results

```
class YOLOv3 {
public:
  static std::unique_ptr<YOLOv3> create(const std::string &model_name,
                                        bool need_preprocess = true);
protected:
  explicit YOLOv3();
  YOLOv3(const YOLOv3 &) = delete;
public:
  virtual ~YOLOv3();
public:
  virtual int getInputWidth() const = 0;
  virtual int getInputHeight() const = 0;

  virtual YOLOv3Result run(const cv::Mat &image) = 0;

};
```

Pre-Processing in AI Library

Pre-processing

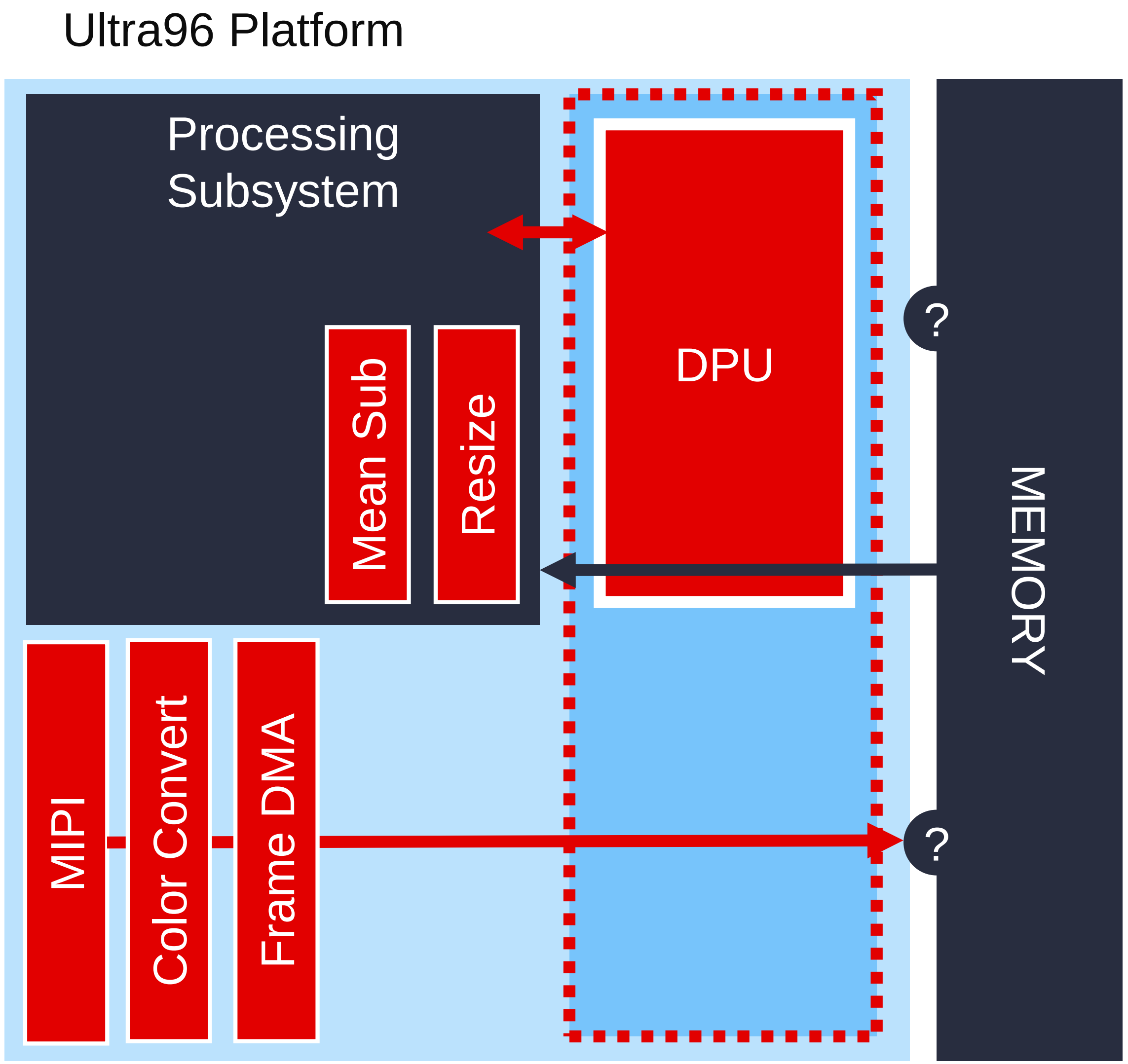
Running DPU

Post-processing

```
YOLOv3Result YOLOv3Imp::run(const cv::Mat &input_image) {
  cv::Mat image;
  int sWidth = getInputWidth();
  int sHeight = getInputHeight();
  auto mAP = configurable_dpu_task_->getConfig().yolo_v3_param().test_map();
  LOG_IF(INFO, false) << "tf_flag_ " << tf_flag_ << " " //
                      << "mAp " << mAP << " "           //
                      << std::endl;
  if (mAP) {
    if (!tf_flag_) {
      int channel = configurable_dpu_task_->getInputTensor()[0][0].channel;
      float scale = xilinx::ai::tensor_scale(
          configurable_dpu_task_->getInputTensor()[0][0]);
      int8_t *data =
          (int8_t *)configurable_dpu_task_->getInputTensor()[0][0].data;
      LOG_IF(INFO, false) << "scale " << scale << " "     //
                          << "sWidth " << sWidth << " "   //
                          << "sHeight " << sHeight << " " //
                          << std::endl;
      yolov3::convertInputImage(input_image, sWidth, sHeight, channel, scale, data);
    } else {
      image = yolov3::letterbox_tf(input_image, sWidth, sHeight).clone();
      configurable_dpu_task_->setInputImageRGB(image);
    }
  } else {
    auto size = cv::Size(sWidth, sHeight);
    if (size != input_image.size()) {
      cv::resize(input_image, image, size, 0, 0, cv::INTER_LINEAR);
    } else {
      image = input_image;
    }
    //  convert_RGB(image);
    __TIC__(YOLOV3_SET_IMG)
    configurable_dpu_task_->setInputImageRGB(image);
    __TOC__(YOLOV3_SET_IMG)
  }
  __TIC__(YOLOV3_DPU)
  configurable_dpu_task_->run(0);
  __TOC__(YOLOV3_DPU)

  __TIC__(YOLOV3_POST_ARM)

  auto ret = xilinx::ai::yolov3_post_process(
      configurable_dpu_task_->getInputTensor()[0],
      configurable_dpu_task_->getOutputTensor()[0],
      configurable_dpu_task_->getConfig(), input_image.cols, input_image.rows);

  __TOC__(YOLOV3_POST_ARM)
  return ret;
}
```

What Preprocessing Functions Need Acceleration?

- Camera Input is 640x480 RGB
- FPN model needs 512x256
- Mean Value Subtraction
- (optional) Input Scaling



XILINX

Implement Custom Pre-Processing

Pre-processing

```
void x_pipeline_ssd(hls::stream<ap_axiu<24, 3, 1, 1> >& image_in,
                        ap_uint<AXI_WIDTH> *image_out,
                        int width_in,
                        int height_in,
                        int width_out,
                        int height_out,
                        int use_mean,
                        unsigned char mean[3])
```

```
void grabFrameFromXPipeline(cl::CommandQueue* q, cl::Kernel* x_pipeline_ssd, ui
  cl_int err;
  input->enqueue();
  vector<cl::Event> tasks;

  for (int _pr=0; _pr<64; _pr++) {
    cl::Event event;
    OCL_CHECK(err, err = x_pipeline_ssd->setArg(1, *(buffer_outputs[_pr%8])));
    OCL_CHECK(err, err = q->enqueueTask(*x_pipeline_ssd, NULL, &event));
    tasks.push_back(event);
  }
  for (int _pr=0; _pr<64; _pr++) {
    input->enqueue();
    std::this_thread::sleep_for(std::chrono::milliseconds(5));

    // Use stream switch to divert a frame to accelerator
    *(enabled + 0x10) = 0x1;

    input->startStream();
    input->rotateBuf();
    tasks.at(_pr).wait();

    mtxQueueAccel.lock();
    queueAccel.push(mat_objects[_pr%8]);
    mtxQueueAccel.unlock();
  }
```

Thank You