Advanced SOC Design

Steps for Application Accelerator Development

Example: Robotics Computing – Visual Odometry

Jiin Lai

Purpose

This presentation introduces steps to convert a given application code written in C/C++ or Python to a hardware accelerator run on FPGA. The content is based on an NTHU implementation project, "Robotics Computing - Visual Odometry“

Referenced Example: Robotics Computing – Visual Odometry

Github:

https://github.com/bol-edu/robotics-computing?tab=readme-ov-file

Report:

https://github.com/bol-edu/robotics-computing/blob/main/report_A288.pdf

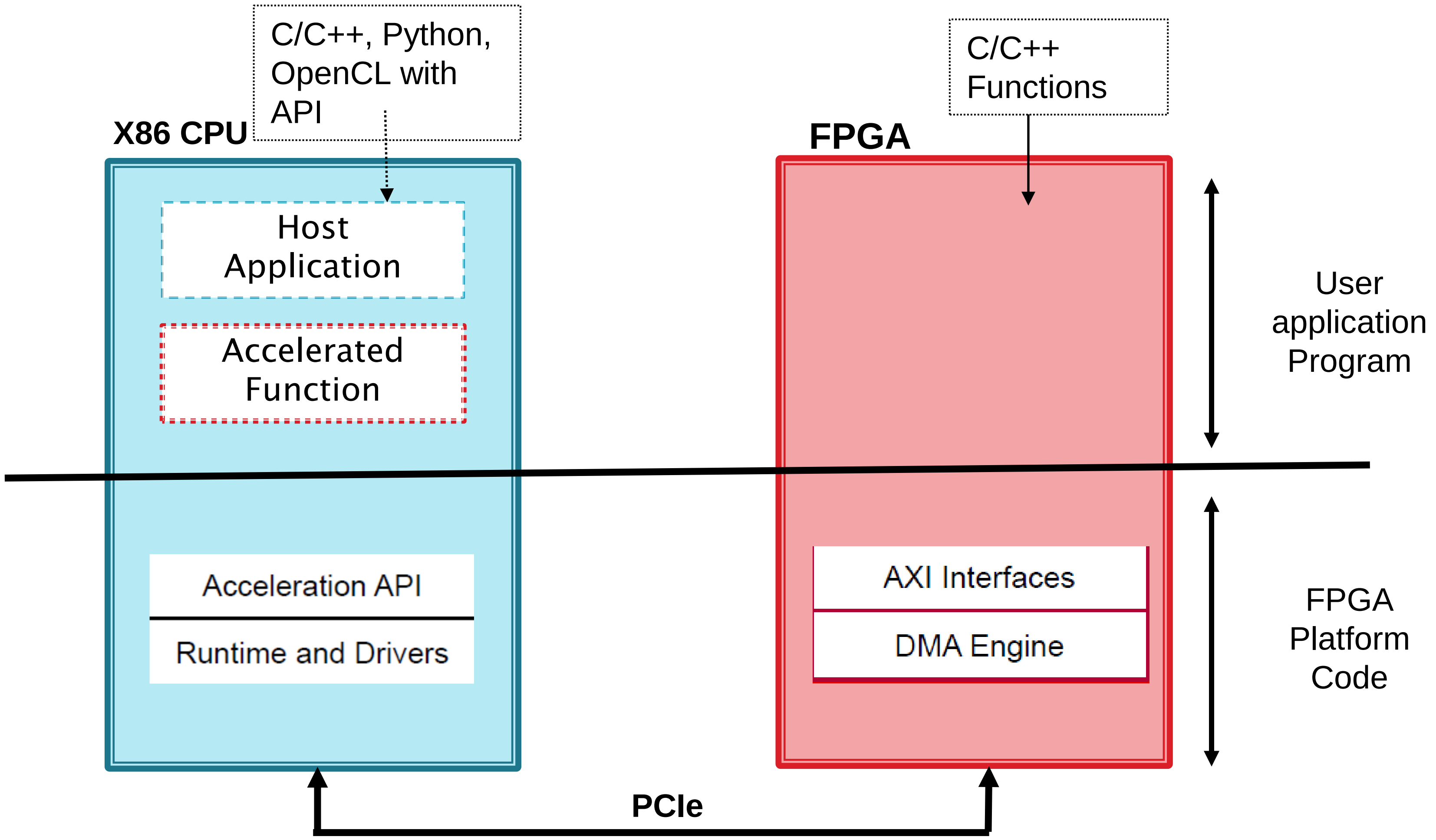
Topics

- Vitis Development Flow Overview
- Steps to Development Application Accelerator

Vitis Development Flow Overview

Reference: Vitis Development Flow
https://www.youtube.com/watch?v=EvBolP6Q2f8&list=PL5CoDA0gtOHWDIbsYuOj10_ocqetwUlWF&index=11

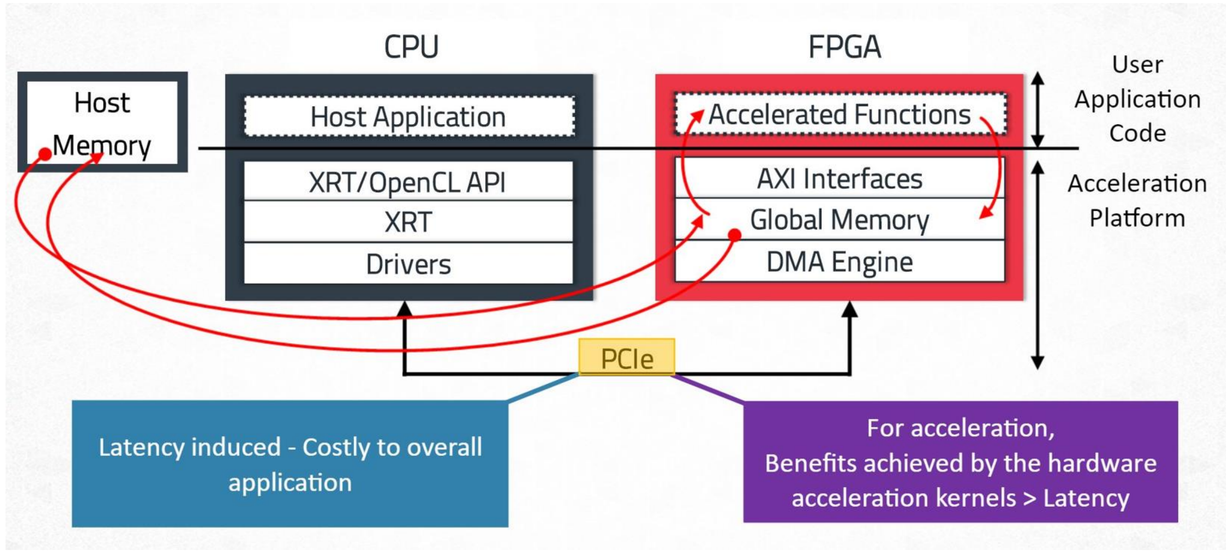
When Part of the Application Moved to FPGA



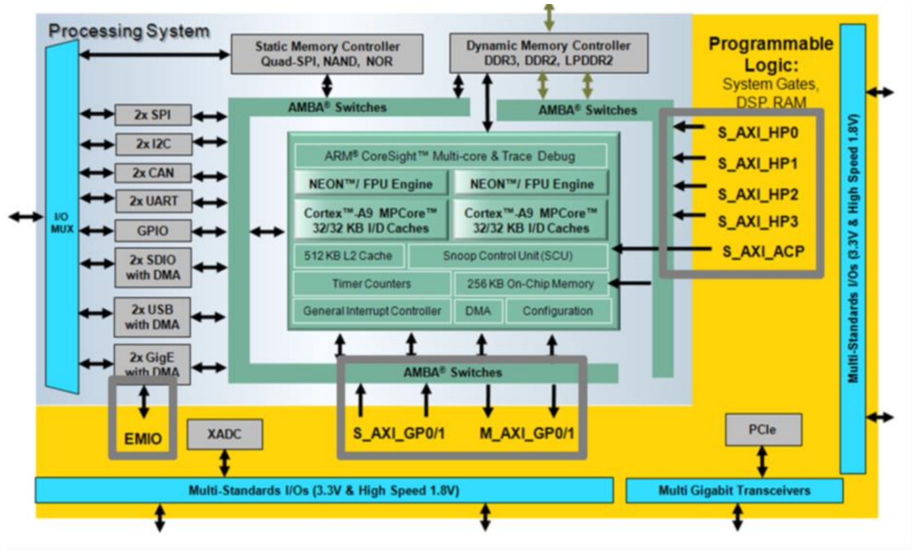
Moving function to FPGA creates a lot of overhead.
Can we really gain performance and reduce power?

Vitis Programming and Execution Model

PCIe FPGA – Data Center



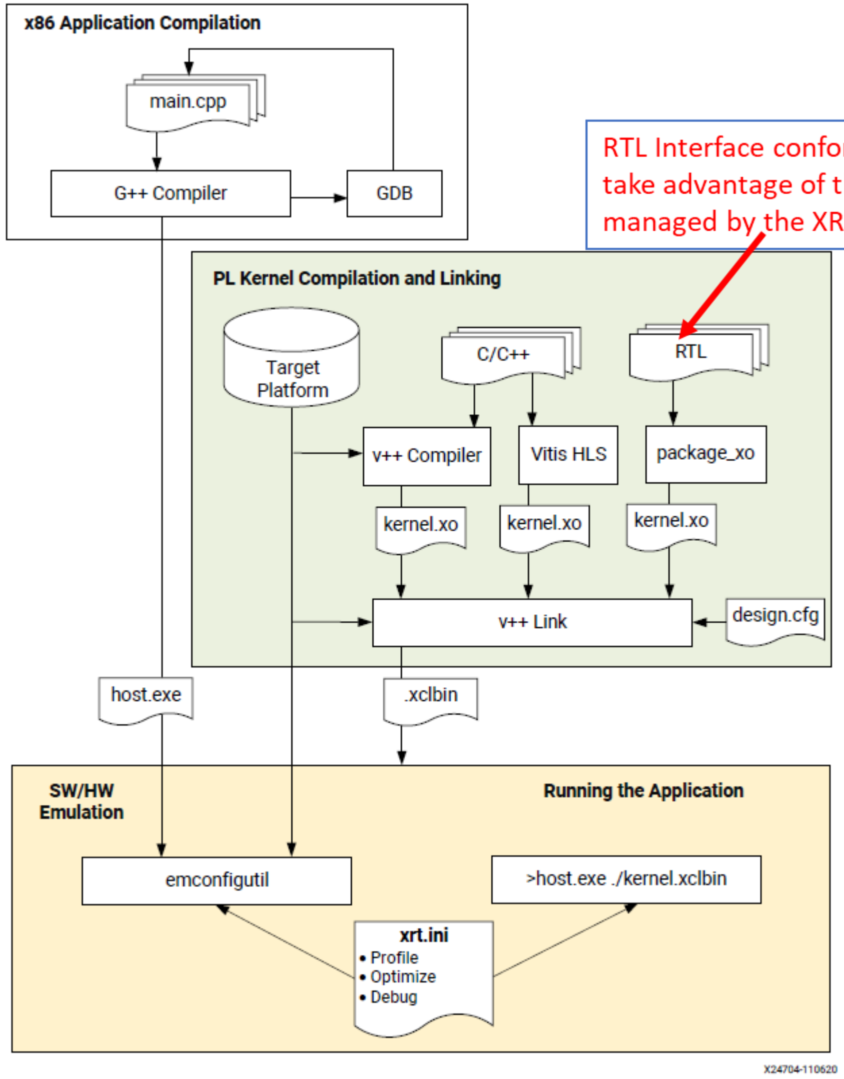
MPSOC – Embedded Platform


Processing System
Static Memory Controller Quad-SPI, NAND, NOR
Dynamic Memory Controller DDR3, DDR2, LPDDR2
AMBA® Switches
AMBA® Switches
Programmable Logic: System Gates, DSP, RAM
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
I/O MUX
ARM® CoreSight™ Multi-core & Trace Debug
NEON™/ FPU Engine
NEON™/ FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
S_AXI_HP0
S_AXI_HP1
S_AXI_HP2
S_AXI_HP3
S_AXI_ACP
AMBA® Switches
EMIO
XADC
S_AXI_GP0/1
M_AXI_GP0/1
PCIe
Multi-Standards I/Os (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers
Multi-Standards I/Os (3.3V & High Speed 1.8V)

1. The host program writes the data needed by a kernel into the global memory of the FPGA device.
2. The host program sets up the input parameters of the kernel, and move data to FPGA Global memory.
3. The host program triggers the execution of the kernel.
4. The kernel performs the required computation, accessing global memory to read or write data as necessary. Kernels can also use streaming connections to communicate with other kernels.
5. The kernel notifies the host that it has completed its task.
6. The host program can transfer the data from global memory to host memory or give ownership of the data to another kernel.

What is the difference in execution model between PCIe FPGA v.s. MPSOC FPGA ?

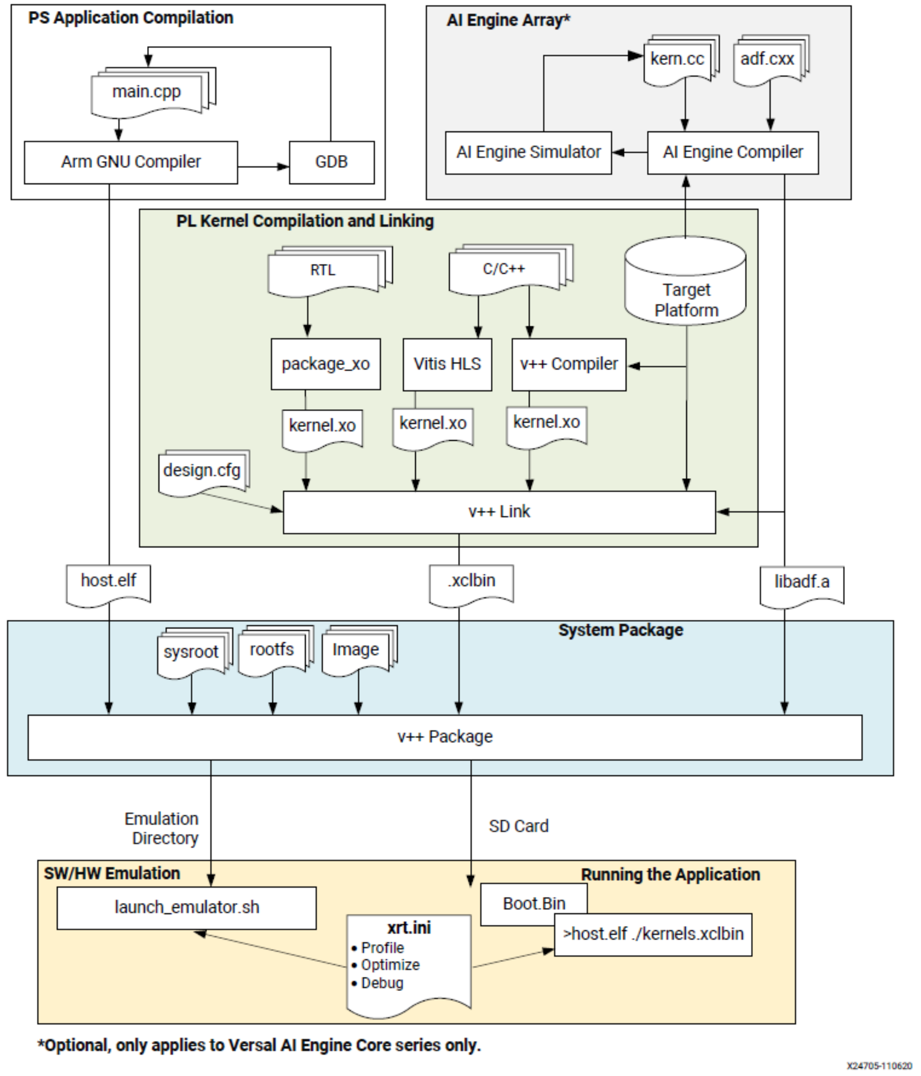
Data Center Build Flow



RTL Interface conform to XRT execution model to take advantage of the automatic execution flow managed by the XRT.

> host.exe ./kernel.xclkbin

Embedded Platform Build Flow

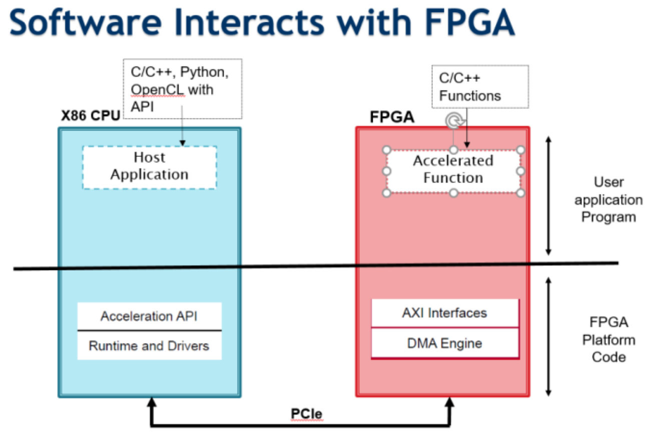


> host.elf ./kernel.xclkbin

Note: We do not use PCIe FPGA (XRT) in our course, but it is the standard Application Accelerator Platform

PYNQ = Embedded Jupyter Notebook

+ Pythonic integration of FPGA & Hard IPs



Jupyter notebooks, browser-based interface

PYNQ enables JupyterLab on Zynq and ZU+

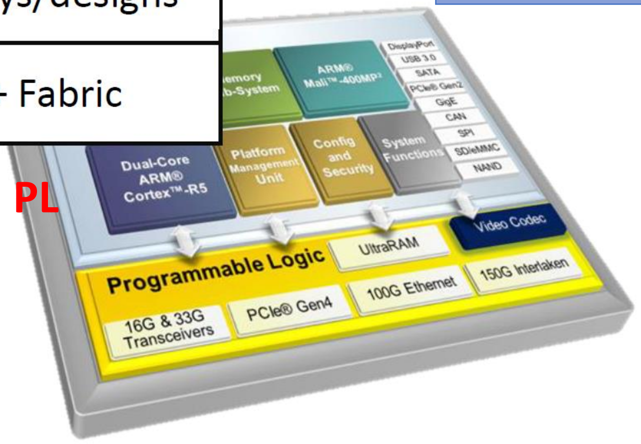
Most competitive, open and advanced GUI

WAN

PS	PL
Jupyter web server	
IPython kernel	
Ubuntu-based Linux	Overlays/designs
ARM A9 / A53	ZU+ Fabric

Hardware C-drivers wrapped in Python packages

Overlay – package of
- Design Bitstream
- Design metadata file (hwh)
- C Driver



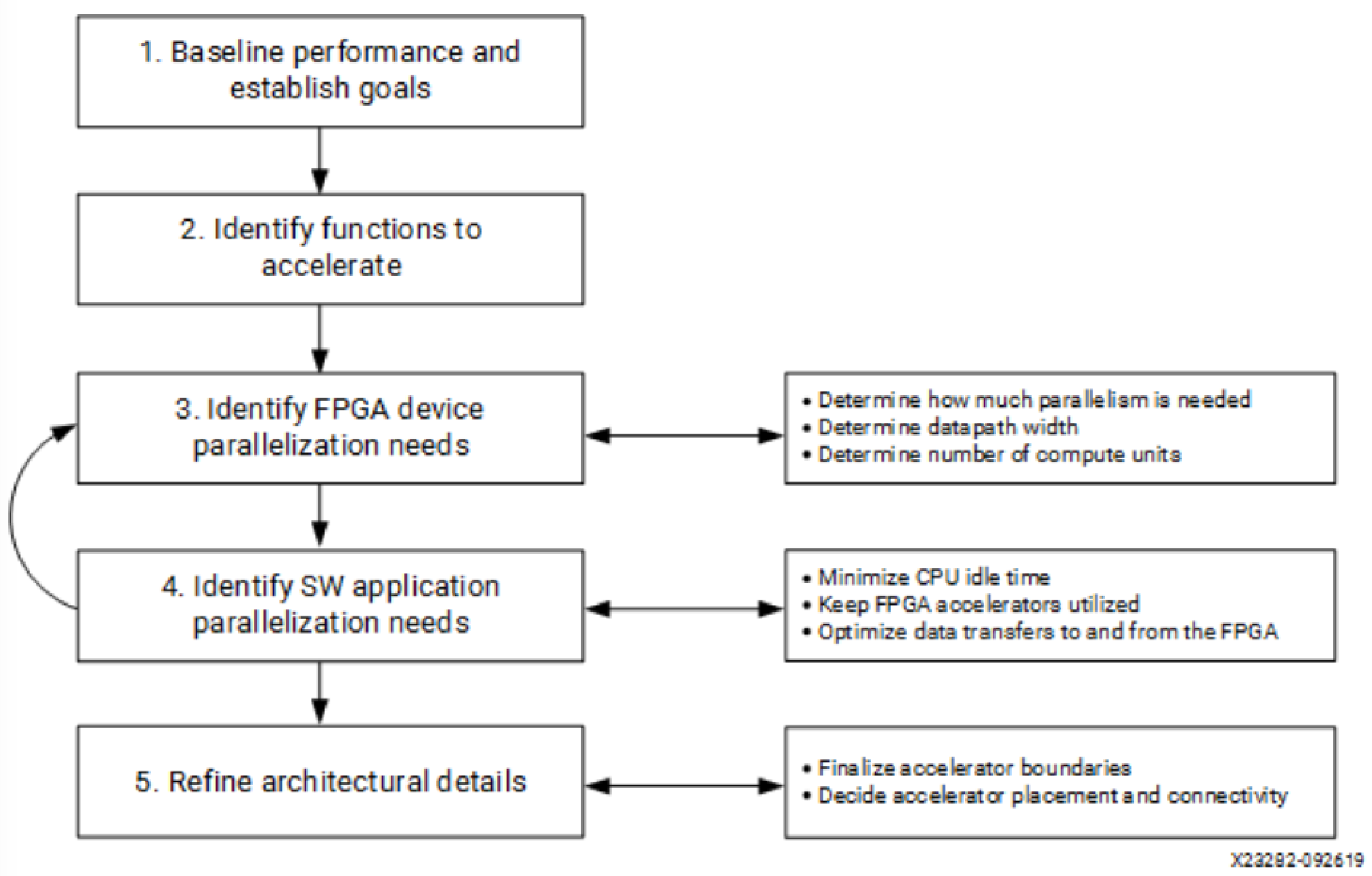
Target Flow

- Vivado IP Flow Target
  - Generate RTL IP for use in Block Design
  - No default interface
- Vitis Kernel Flow Target
  - Generate kernel object (.xo) to interact XRT
    - Sequential Mode – ap_ctrl_hs
    - Pipeline Mode – ap_ctrl_chain
    - Free-Running Mode – ap_ctrl_none
  - Kernel Argument Interface
    - Scalars (s_axilite)
    - Direct data transfer with global memory (m_axi)
    - Streaming (axis)

Guideline in HLS/Vitis Development

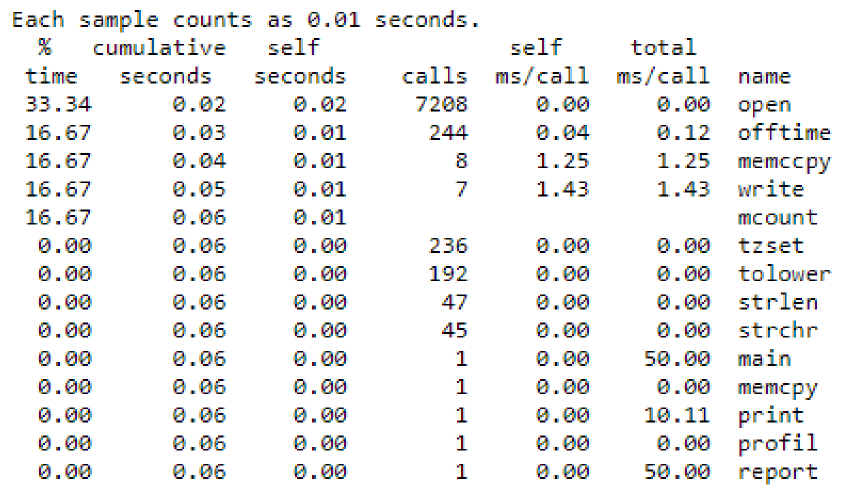
Optimization Guideline

Figure: Methodology for Architecting the Application

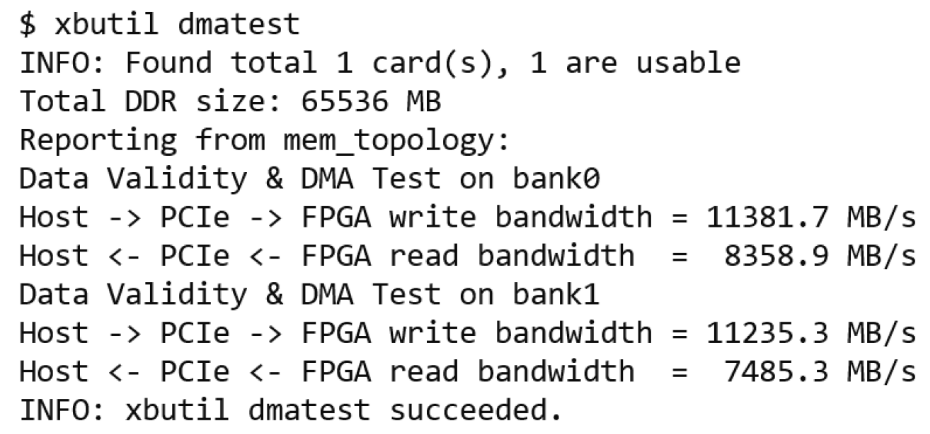

1. Baseline performance and establish goals
2. Identify functions to accelerate
3. Identify FPGA device parallelization needs
4. Identify SW application parallelization needs
5. Refine architectural details
• Determine how much parallelism is needed
• Determine datapath width
• Determine number of compute units
• Minimize CPU idle time
• Keep FPGA accelerators utilized
• Optimize data transfers to and from the FPGA
• Finalize accelerator boundaries
• Decide accelerator placement and connectivity
X23292-092619

Establish a Baseline Application Performance

- Measure Running Time
  - Profiling tools : valgrind, callgrind, gprof
  - Use <time.h>, clock()
- Measure Throughput
- Determine if it is PCIe-bound
- Establish Overall Acceleration Goals



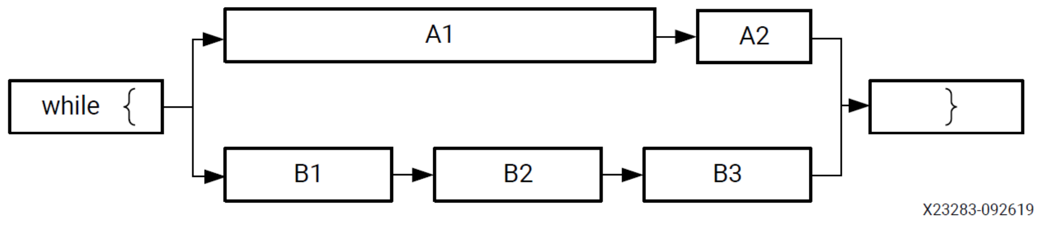
```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
```



```
$ xbutil dmatest
INFO: Found total 1 card(s), 1 are usable
Total DDR size: 65536 MB
Reporting from mem_topology:
Data Validity & DMA Test on bank0
Host -> PCIe -> FPGA write bandwidth = 11381.7 MB/s
Host <- PCIe <- FPGA read bandwidth  =  8358.9 MB/s
Data Validity & DMA Test on bank1
Host -> PCIe -> FPGA write bandwidth = 11235.3 MB/s
Host <- PCIe <- FPGA read bandwidth  =  7485.3 MB/s
INFO: xbutil dmatest succeeded.
```

Identify Functions to Accelerate

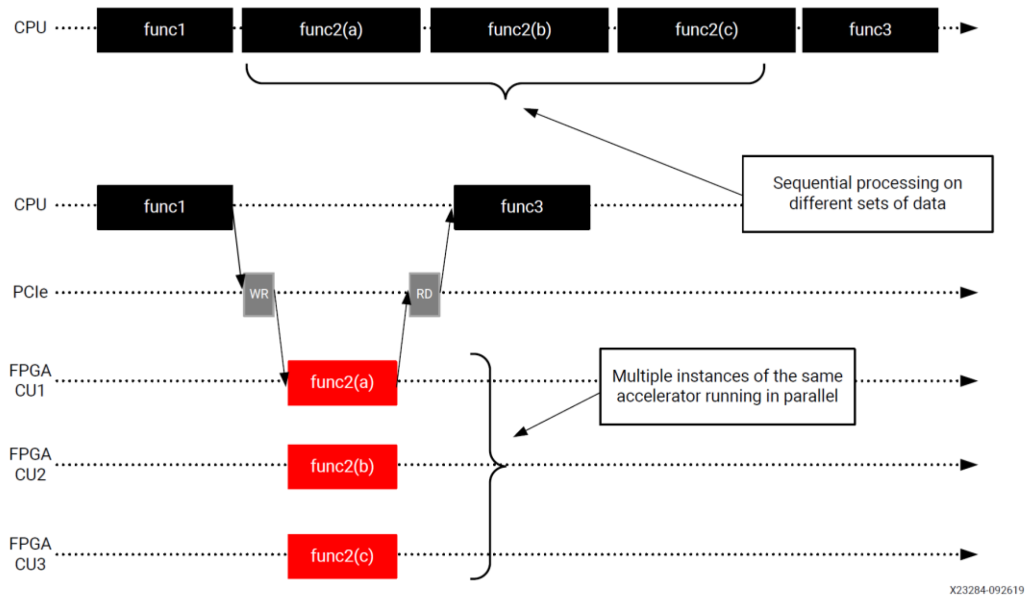
- From the dataflow graph, identify Performance Bottlenecks, consider the entire application's performance



- Identify Acceleration Potential
  - What is the parallelism of the function? Task-level, data-level
  - Computational Intensity? #-of-op / #-of-words-moved-from-slow-memory
  - Data access locality? Spatial / Temporal locality
  - Does Bottleneck come from I/O, computation, data movement, or synchronization?
    - PCIe throughput of 10GBs, software throughput of 50MBs, acceleration factor?

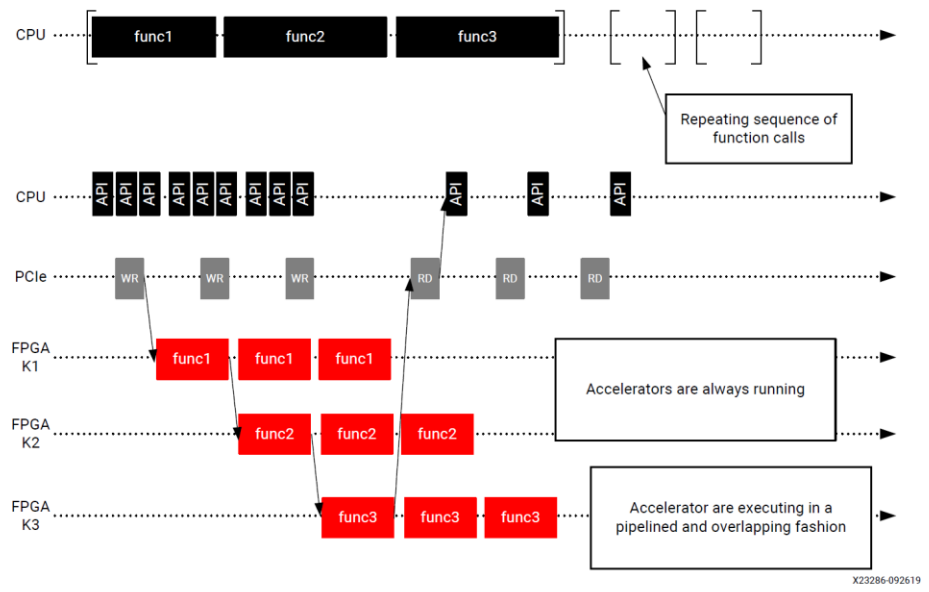
Identify Device Parallelism

- Data-level Parallelism - Wider datapath and processing more samples in parallel (SIMD) – Instantiate multiple Computation Units (CUs)
- Task-level parallelism – Multiple Kernels


CPU
func1
func2(a)
func2(b)
func2(c)
func3
Sequential processing on different sets of data
PCIe
WR
RD
FPGA CU1
FPGA CU2
FPGA CU3
Multiple instances of the same accelerator running in parallel
X23284-092619

Data-level Parallelism - Multiple CUs

Figure 11: **Pipelined Execution of Accelerators**


CPU
func1
func2
func3
Repeating sequence of function calls
API
PCIe
WR
RD
FPGA K1
FPGA K2
FPGA K3
Accelerators are always running
Accelerator are executing in a pipelined and overlapping fashion
X23286-092619

Task-level Parallelism – Multiple Kernels

Incorporate Software Application Parallelism to Improve Kernel Efficiency

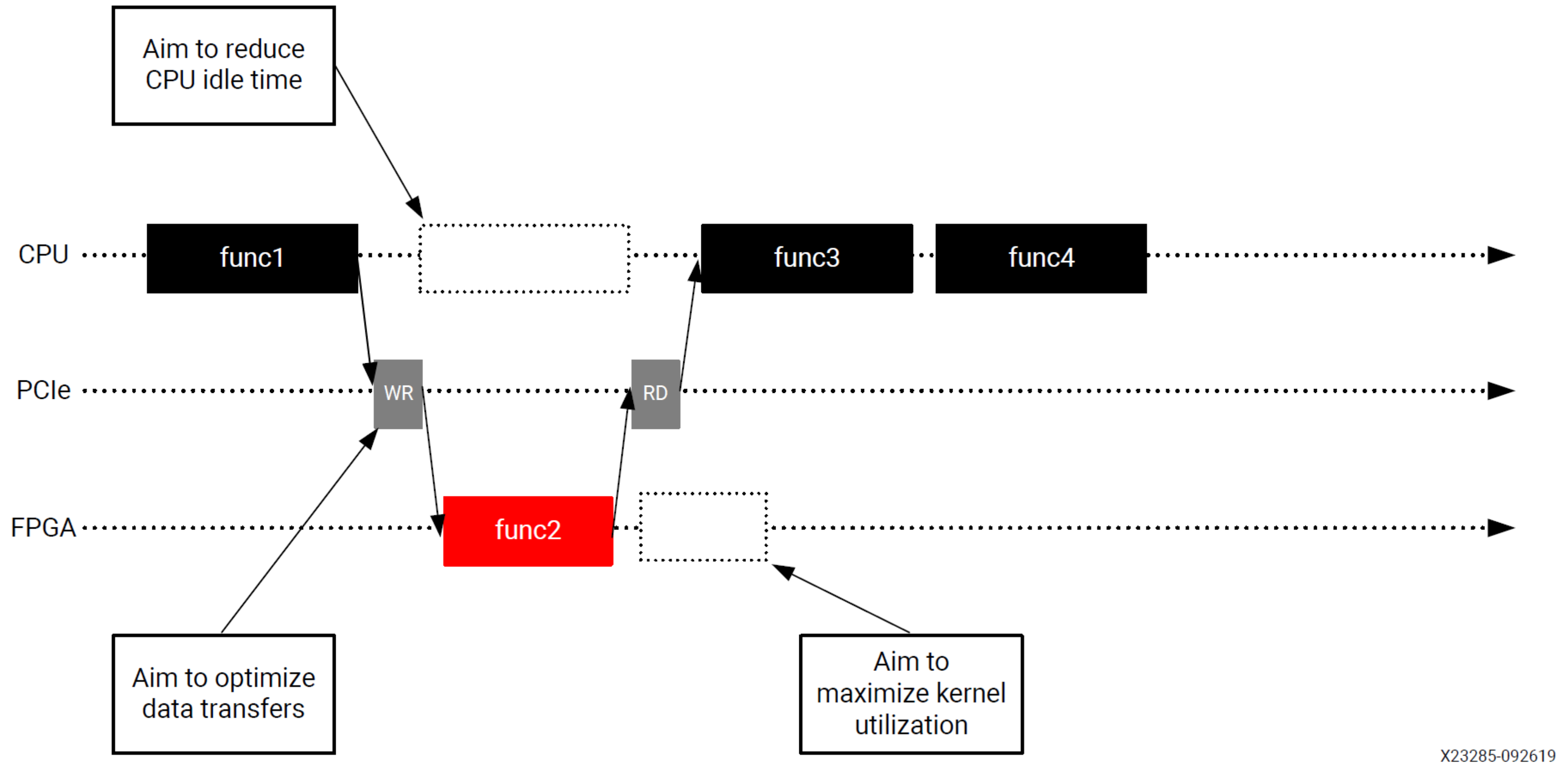

Aim to reduce CPU idle time
CPU
func1
func3
func4
PCIe
WR
RD
FPGA
func2
Aim to optimize data transfers
Aim to maximize kernel utilization
X23285-092619

Figure: Host Application Timeline

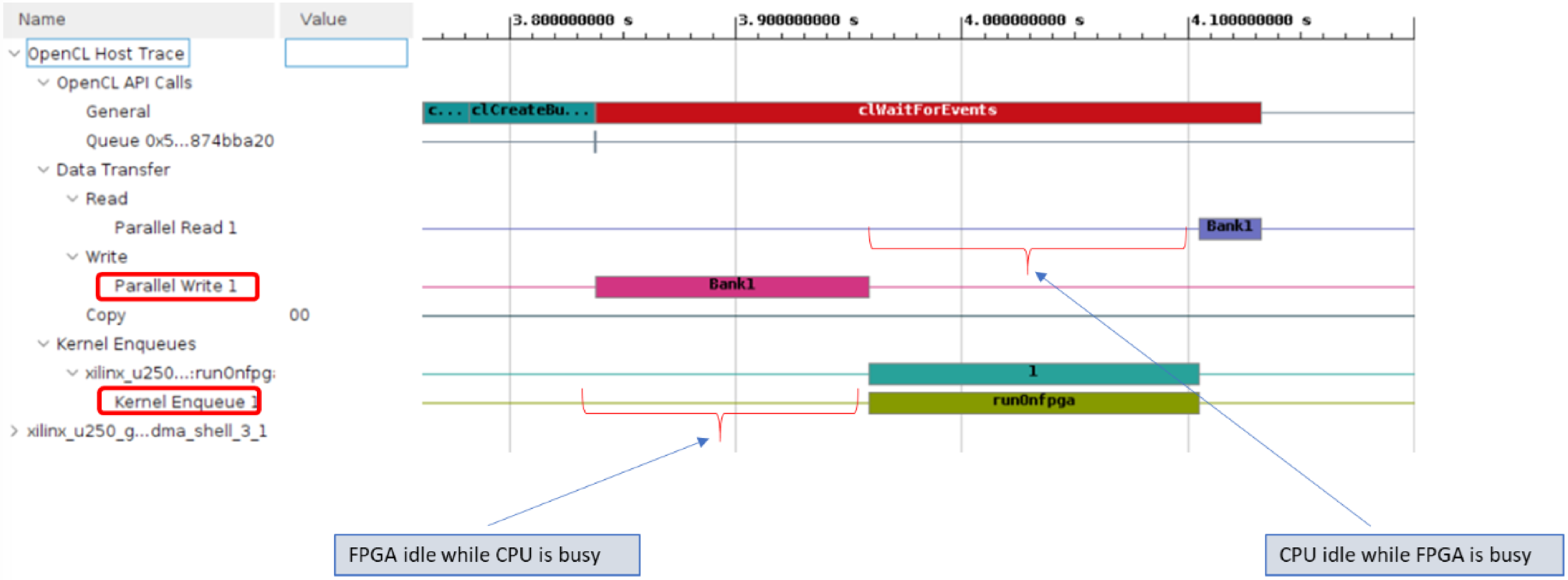
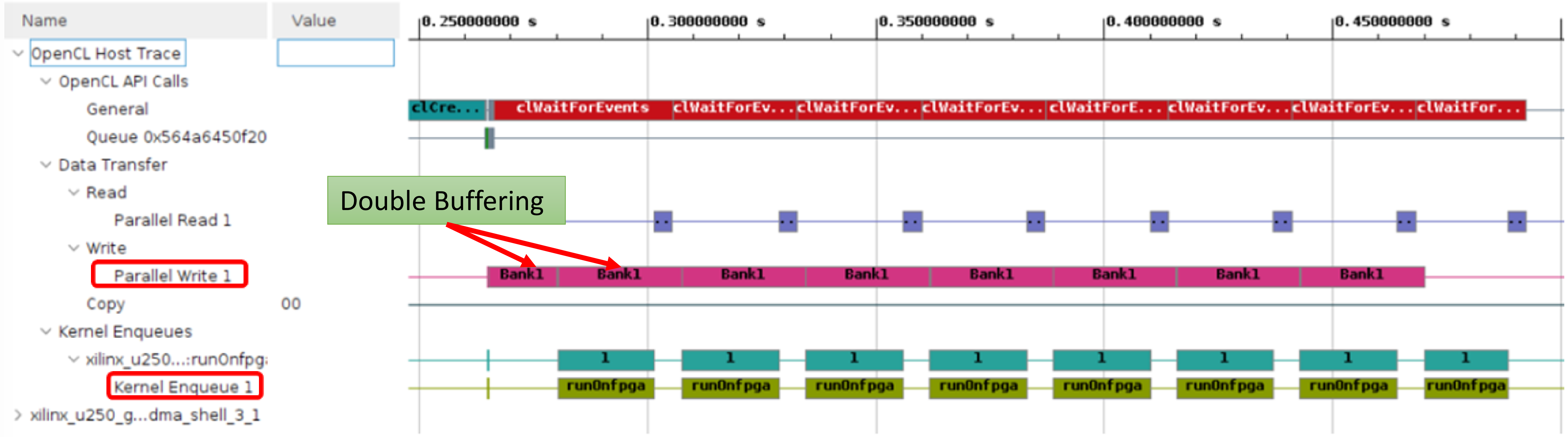


Figure: Improved Timeline



Higher utilization of FPGA and CPU

The Role of Host Program

- System Topology – Scalable
  - Multiple CUs in one FPGA
  - Multiple FPGAs in one machine
  - Across network
- Functions ~ Operation System
  - Interface with Application (API)
  - Kernel Scheduling
  - Memory management and data movement scheduling
  - Data preprocessing/post-processing
  - Correctness check
  - System monitoring, tuning, metering

HLS Kernel Development Flow

- **C-simulation** – Compile, Simulate, and Debug
  - Use Debugger Perspective to debug the code
  - Use **Pre-synthesis Control Flow Viewer** to identify where the optimization is needed
- **Synthesis**
  - Use directive or pragma to guide HLS optimization process (**Minimize** the code restructure to meet performance)
  - Study **Synthesis Summary Report**
  - Use **Schedule Viewer**, **Property viewer**, **Dataflow viewer** to analyze datapath and control flow
- **C/RTL Co-simulation** – Verify RTL implementation (don't skip this step)
  - Use **Waveform viewer** or **Co-simulation deadlock viewer** if deadlock happens

Regression & Auto-check

Vitis Build Targets

- **Software Emulation** - (C-simulation)
  - Host program & PL kernels are natively compiled and running on host machine
  - Source-level debugging of the kernel code running together with application and verifying the behavior of the system.
- **Hardware Emulation** (C/RTL co-simulation)
  - The host program runs natively on x86, kernel code is compiled into an RTL & run in simulator
  - Provide cycle-accurate view of kernel logic. Getting initial performance estimate.
  - Some issue in hardware may not be reproducible in hardware emulation.
  - PCIe is a functional model, It does not reflect the actual PCIe bandwidth in hardware
  - DDR is a functional model. It does not model the latency in hardware.
- **Hardware** -
  - The kernel code is compiled into FPGA bitstream which runs in FPGA
  - Reflect the actual performance of your accelerated application.
  - Profiling data from this hardware run, and further optimize system performance.

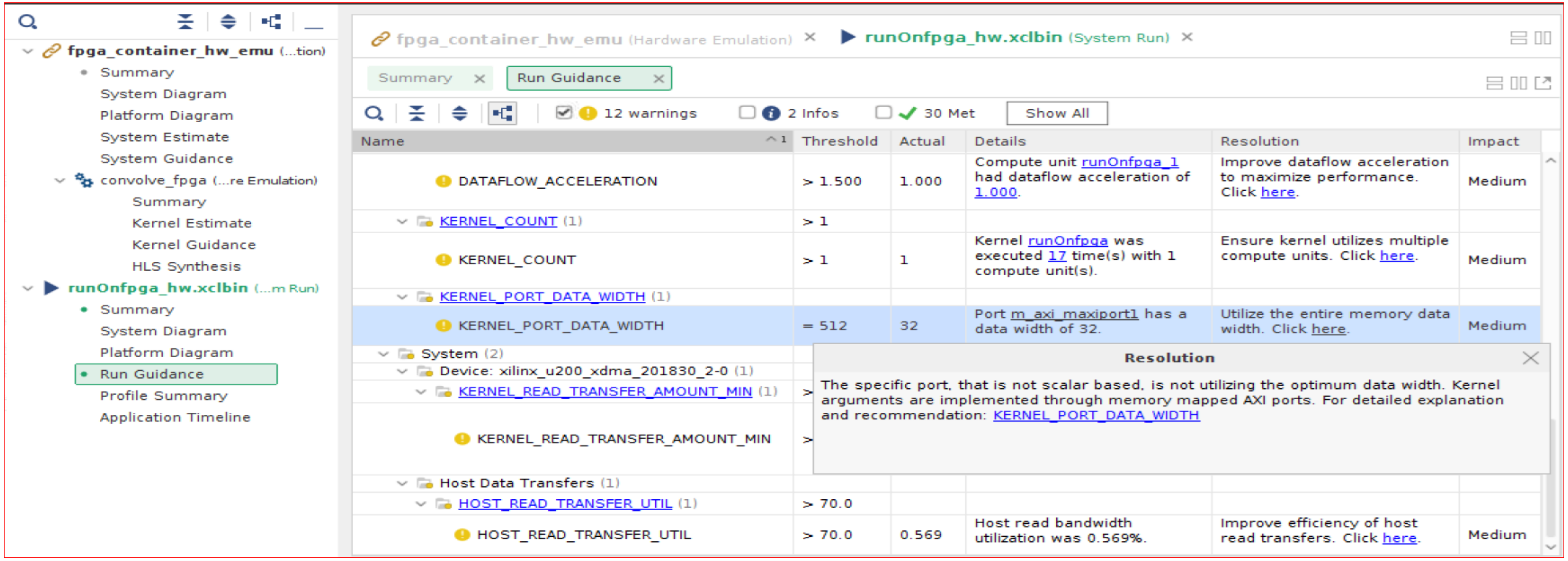
Vitis Analyzer

- **Guidance** - shows sub-optimalities in your application and provides actionable feedback on how to improve it.
- **Profile Summary** - annotated details for the overall application performance.
- **Application Timeline** (timeline_trace.csv)- displace host and device events on a common timeline
- **HLS Report** – insights to guide kernel optimization
- **Waveform View and Live Waveform Viewer** – insight into performance bottlenecks

Guidance

Provide informational messages

- Report violations
- Brief suggested resolutions
- Web link for detail resolution



```
fpga_container_hw_emu (...tion)
    Summary
    System Diagram
    Platform Diagram
    System Estimate
    System Guidance
    convolve_fpga (...re Emulation)
        Summary
        Kernel Estimate
        Kernel Guidance
        HLS Synthesis
runOnfpga_hw.xclbin (...m Run)
    Summary
    System Diagram
    Platform Diagram
    Run Guidance
    Profile Summary
    Application Timeline
```

fpga_container_hw_emu (Hardware Emulation) × runOnfpga_hw.xclbin (System Run) ×

Summary × Run Guidance ×

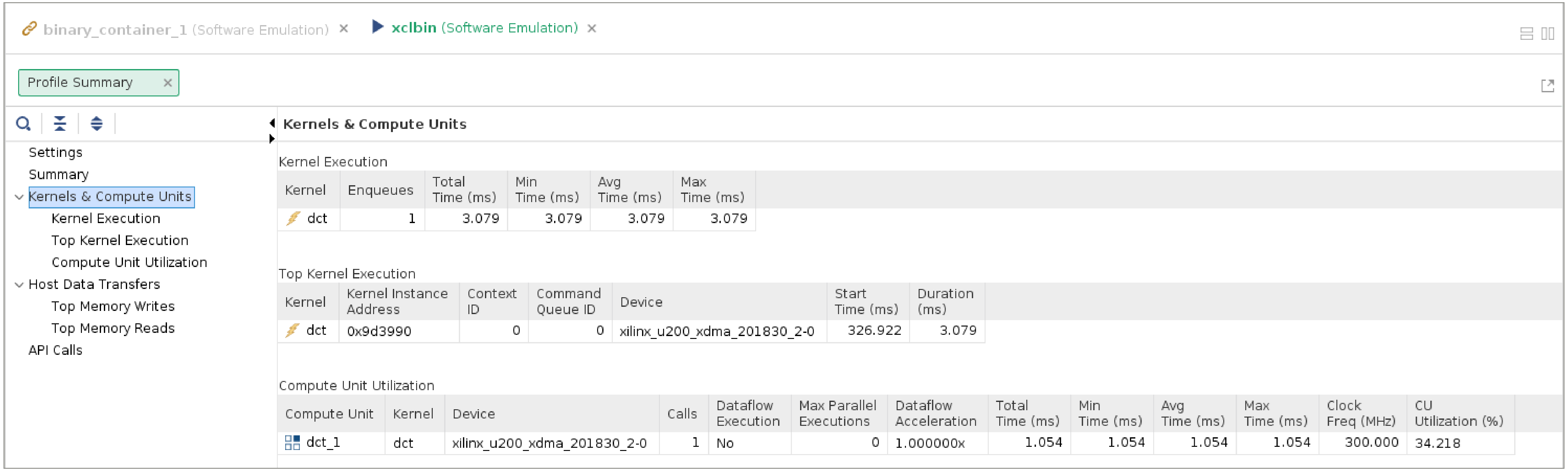
12 warnings | 2 Infos | 30 Met | Show All

Name	Threshold	Actual	Details	Resolution	Impact
DATAFLOW_ACCELERATION	> 1.500	1.000	Compute unit runOnfpga_1 had dataflow acceleration of 1.000.	Improve dataflow acceleration to maximize performance. Click here.	Medium
KERNEL_COUNT (1)	> 1				
KERNEL_COUNT	> 1	1	Kernel runOnfpga was executed 17 time(s) with 1 compute unit(s).	Ensure kernel utilizes multiple compute units. Click here.	Medium
KERNEL_PORT_DATA_WIDTH (1)					
KERNEL_PORT_DATA_WIDTH	= 512	32	Port m_axi_maxiport1 has a data width of 32.	Utilize the entire memory data width. Click here.	Medium
System (2)					
Device: xilinx_u200_xdma_201830_2-0 (1)					
KERNEL_READ_TRANSFER_AMOUNT_MIN (1)	>				
KERNEL_READ_TRANSFER_AMOUNT_MIN	>				
Host Data Transfers (1)					
HOST_READ_TRANSFER_UTIL (1)	> 70.0				
HOST_READ_TRANSFER_UTIL	> 70.0	0.569	Host read bandwidth utilization was 0.569%.	Improve efficiency of host read transfers. Click here.	Medium

Resolution

The specific port, that is not scalar based, is not utilizing the optimum data width. Kernel arguments are implemented through memory mapped AXI ports. For detailed explanation and recommendation: KERNEL_PORT_DATA_WIDTH

Profile Summary Report



binary_container_1 (Software Emulation) ×   xclbin (Software Emulation) ×

Profile Summary ×

Settings
Summary
Kernels & Compute Units
Kernel Execution
Top Kernel Execution
Compute Unit Utilization
Host Data Transfers
Top Memory Writes
Top Memory Reads
API Calls

Kernels & Compute Units

Kernel Execution

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
dct	1	3.079	3.079	3.079	3.079

Top Kernel Execution

Kernel	Kernel Instance Address	Context ID	Command Queue ID	Device	Start Time (ms)	Duration (ms)
dct	0x9d3990	0	0	xilinx_u200_xdma_201830_2-0	326.922	3.079

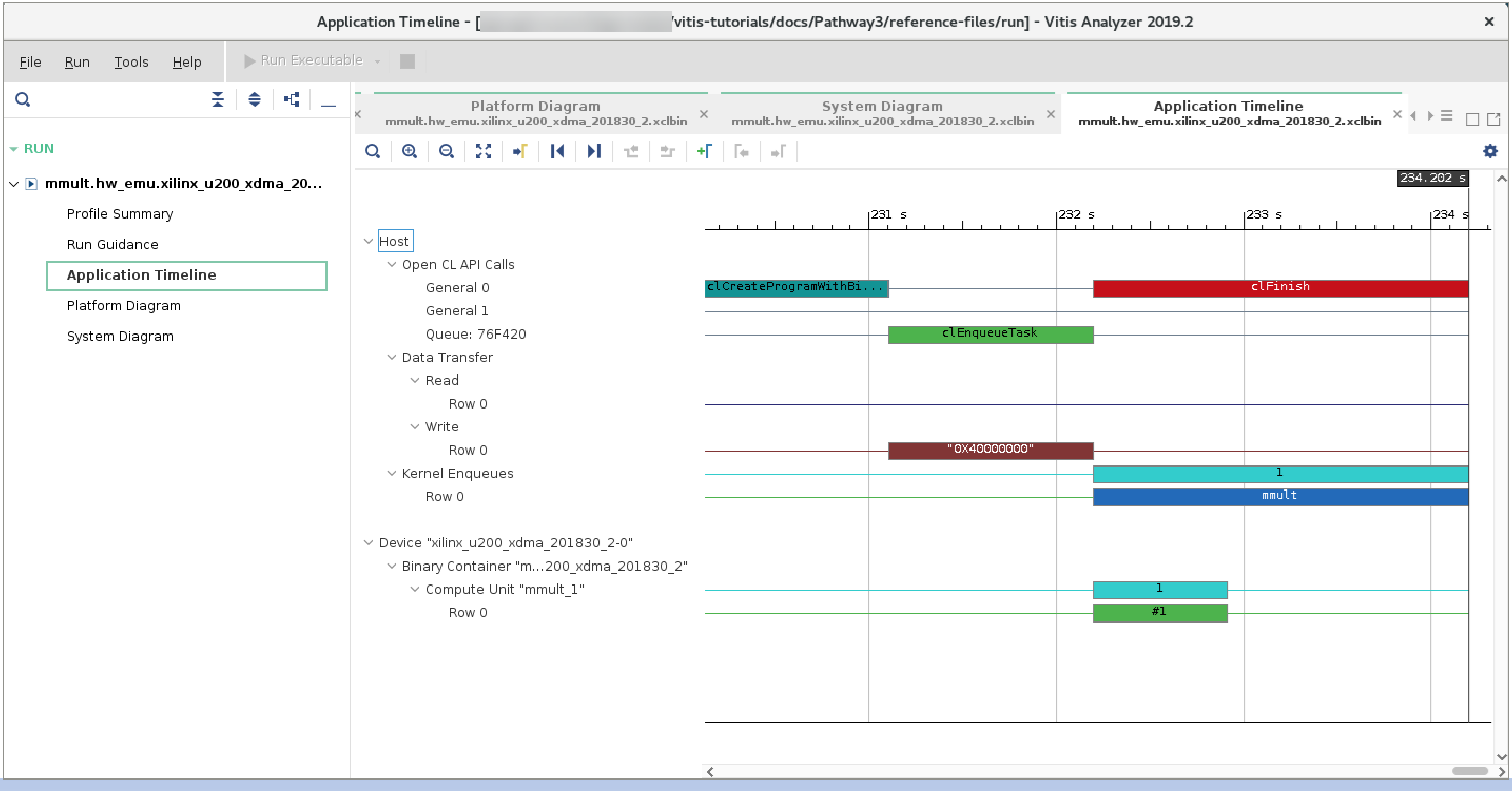
Compute Unit Utilization

Compute Unit	Kernel	Device	Calls	Dataflow Execution	Max Parallel Executions	Dataflow Acceleration	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)	Clock Freq (MHz)	CU Utilization (%)
dct_1	dct	xilinx_u200_xdma_201830_2-0	1	No	0	1.000000x	1.054	1.054	1.054	1.054	300.000	34.218

Profiling (Hardware Emulation, Hardware)

- Host and device timeline events
- OpenCL™ or XRT native API call sequences
- Kernel execution sequence
- Kernel start and stop signals
- FPGA trace data including AXI transactions
- User event and range profiling

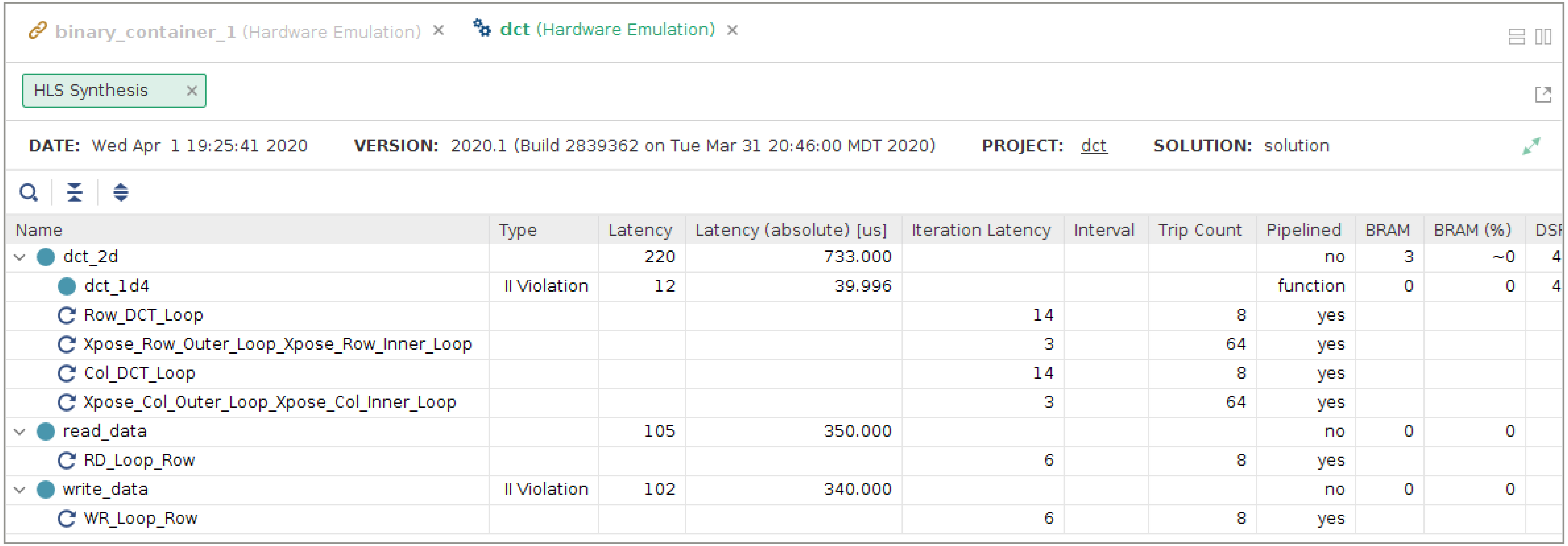
Application Timeline



Application Timeline - [/vitis-tutorials/docs/Pathway3/reference-files/run] - Vitis Analyzer 2019.2
File
Run
Tools
Help
Run Executable
RUN
mmult.hw_emu.xilinx_u200_xdma_20...
Profile Summary
Run Guidance
Application Timeline
Platform Diagram
System Diagram
Platform Diagram mmult.hw_emu.xilinx_u200_xdma_201830_2.xclbin
System Diagram mmult.hw_emu.xilinx_u200_xdma_201830_2.xclbin
Application Timeline mmult.hw_emu.xilinx_u200_xdma_201830_2.xclbin
234.202 s
231 s
232 s
233 s
234 s
Host
Open CL API Calls
General 0
General 1
Queue: 76F420
Data Transfer
Read
Row 0
Write
Row 0
Kernel Enqueues
Row 0
Device "xilinx_u200_xdma_201830_2-0"
Binary Container "m...200_xdma_201830_2"
Compute Unit "mmult_1"
Row 0
clCreateProgramWithBi...
clFinish
clEnqueueTask
"0X40000000"
1
mmult
1
#1

©BOLEDU

HLS Report



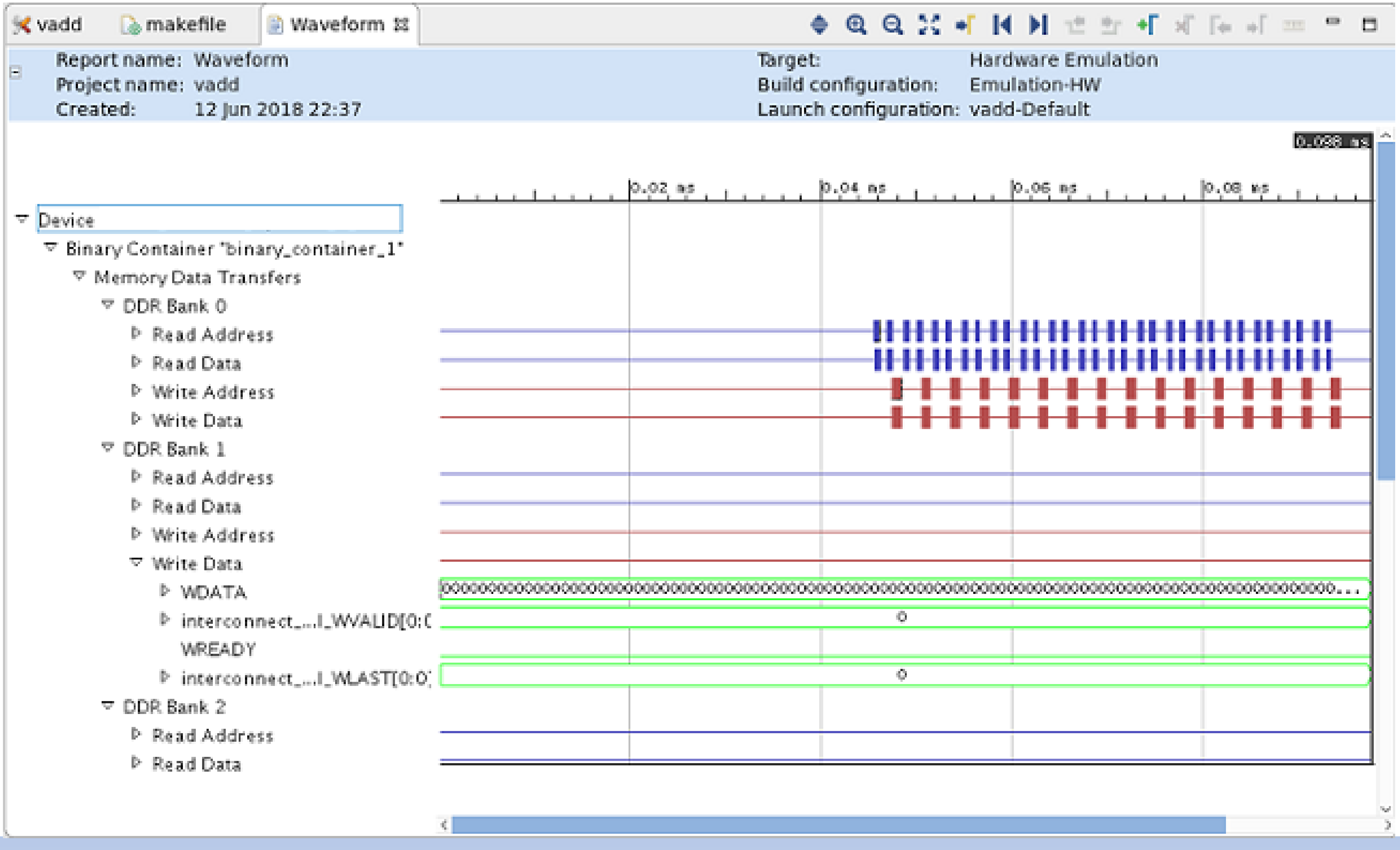
binary_container_1 (Hardware Emulation) × dct (Hardware Emulation) ×

HLS Synthesis ×

DATE: Wed Apr 1 19:25:41 2020 **VERSION:** 2020.1 (Build 2839362 on Tue Mar 31 20:46:00 MDT 2020) **PROJECT:** dct **SOLUTION:** solution

Name	Type	Latency	Latency (absolute) [us]	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP
dct_2d		220	733.000				no	3	~0	4
dct_1d4	II Violation	12	39.996				function	0	0	4
Row_DCT_Loop				14		8	yes			
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop				3		64	yes			
Col_DCT_Loop				14		8	yes			
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop				3		64	yes			
read_data		105	350.000				no	0	0	
RD_Loop_Row				6		8	yes			
write_data	II Violation	102	340.000				no	0	0	
WR_Loop_Row				6		8	yes			

Waveform View and Live Waveform Viewer



Report name: Waveform
Project name: vadd
Created: 12 Jun 2018 22:37

Target: Hardware Emulation
Build configuration: Emulation-HW
Launch configuration: vadd-Default

Device
Binary Container "binary_container_1"
Memory Data Transfers
DDR Bank 0
Read Address
Read Data
Write Address
Write Data
DDR Bank 1
Read Address
Read Data
Write Address
Write Data
WDATA
interconnect_...I_WVALID[0:0
WREADY
interconnect_...I_WLAST[0:0]
DDR Bank 2
Read Address
Read Data

Review Logfile for C-simulation, Synthesis, C/RTL Co-simulation

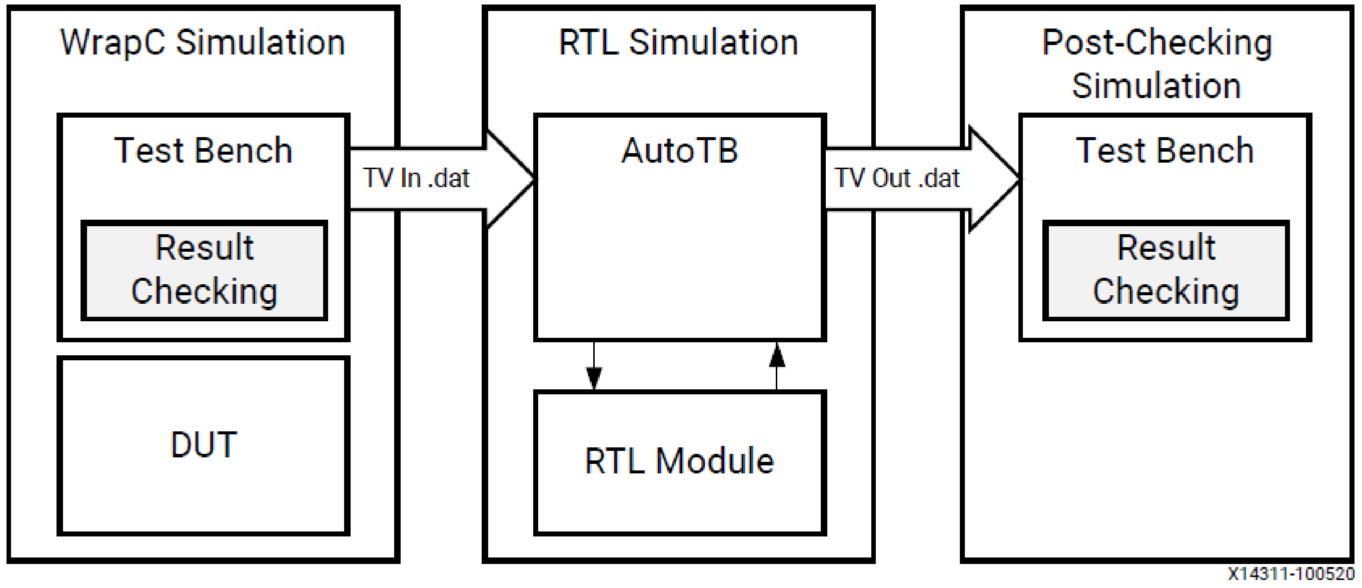
- Logfile
- C-simulation
- Synthesis – it shows the synthesis steps, various checking status, optimization result
  - Project and solution initialization loads source and constraints files, and configures the active solution for synthesis.
  - Start compilation reads source files into memory.
  - Interface detection and setup reviews and generates port and block interfaces for the function.
  - Burst read and write analysis for ports/interfaces.
  - Compiler transforms code to operations.
  - Performs Synthesizability checks.
  - Automatic pipelining of loops at trip-count threshold.
  - Unrolling loops, both automatic and user-directed.
  - Balance expressions using associative and commutative properties.
  - Loop flattening to reduce loop hierarchy.
  - Partial write detection (writing part of a memory word)
  - Finish architecture synthesis, start scheduling.
  - End scheduling, generate RTL code.
  - Report F-Max and loop constraint status.
- C/RTL Co-simulation

Guideline for Writing a Test Bench

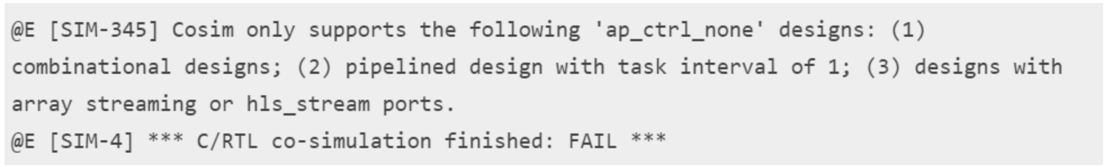
- Separate the top function for synthesis from the test bench and header files.
- If there are multiple kernel functions, group them in one top-level function for synthesis.
- The test bench should execute the top-level function for multiple transactions, allowing different data values to be applied and verified. (test coverage)
- You will use the same test bench for C/RTL Co-simulation. If you want to calculate II during the RTL simulation, you also need to provide multiple transactions to calculate II accurately.
- Checking correctness of kernel run is important. The return value of function main() indicates the following:
  - Zero: Results are correct.
  - Non-zero value: Results are incorrect. The non-zero results indicate the type of failure.

C/RTL Co-simulation

0. Different from Event-Driven Hardware Simulator
1. Phase#1: C simulation is executed to prepare the “input vectors” to the top-level function.
2. Phase#2: RTL simulation starts; it takes the input vectors and generates the “output vectors”
3. Phase#3: C simulation of the test bench main() function continues. It takes the “output vectors” returned from the RTL simulation and performs verification of the result.



Requirement for Co-simulation



```
@E [SIM-345] Cosim only supports the following 'ap_ctrl_none' designs: (1)
combinational designs; (2) pipelined design with task interval of 1; (3) designs with
array streaming or hls_stream ports.
@E [SIM-4] *** C/RTL co-simulation finished: FAIL ***
```

- Interface Synthesis Requirement
  - Top-level function using ap_ctrl_chain or ap_ctrl_hs (can no use ap_ctrl_none)
  - Or design must be purely combinational (evaluate in one cycle)
  - Top function must have an initiation interval of 1. No handshake in the data argument. Input test vector are fed into RTL all at once
  - Interface must be all arrays streaming with axis or ap_hs
- **Provide sufficient FIFO depth to avoid stall or deadlock**
  - **Depth declared for the interface is too small. Create Deadlock**
- Certain unsupported Optimization for array or structs
  - Multiple transformation on array on the interface or array within structs
    - Vertical mapping, Reshape, Partition
  - Not support –
    - Horizontal mapping, Vertical mapping
    - Conditional access on the AXI4-Stream with register slice enabled
    - Mapping arrays to streams
- Use Waveform Viewer or Co-simulation deadlock viewer

- https://www.boledu.org/textbooks/hls-textbook/application-acceleration-development-flow/crtl-co-simulation#crtl-co-simlation
- https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Interface-Synthesis-Requirements

Provide Sufficient FIFO Depth to Prevent Co-sim Deadlock

```
#pragma HLS interface mode=m-axi port=<name> bundle=<string> \
register register_mode=<mode> depth=<int> offset=<string> latency=<value>\
clock=<string> name=<string> storage_type=<value>\
num_read_outstanding=<int> num_write_outstanding=<int> \
max_read_burst_length=<int> max_write_burst_length=<int>
```

```
#pragma HLS stream variable=<variable> type=<type> depth=<int>
```

- **depth: Specifies the maximum number of samples for the test bench to process.**

Steps to Development Application Accelerator

Development Flow

1. Identify applicable code, e.g., Github
2. Reproduce the Github result
3. Identify function to accelerate – host/kernel partition
4. Make kernel code self-contained, i.e. no library call
5. Run C-simulation; the result should match step#2
6. Convert to Synthesizable HLS C/C++
7. Run Co-sim with generated RTL code – compare with Step#5
8. Performance / Area optimization by analyzing the timing profile
9. Port to FPGA and verify the result and performance of the whole application scenario

BOLEDU HLS Textbook: application acceleration development flow

https://www.boledu.org/textbooks/hls-textbook/application-acceleration-development-flow/introduction

Step#1: Identify Application Code Base

- Identify the application and implementation algorithm
- Search for a good Github repository, e.g., active, update frequency, collaborator.
  - https://www.quora.com/How-do-you-measure-a-good-github-repository
- Test data set is important

For example Visual Odometry project uses

https://github.com/FoamoftheSea/KITTI_visual_odometry

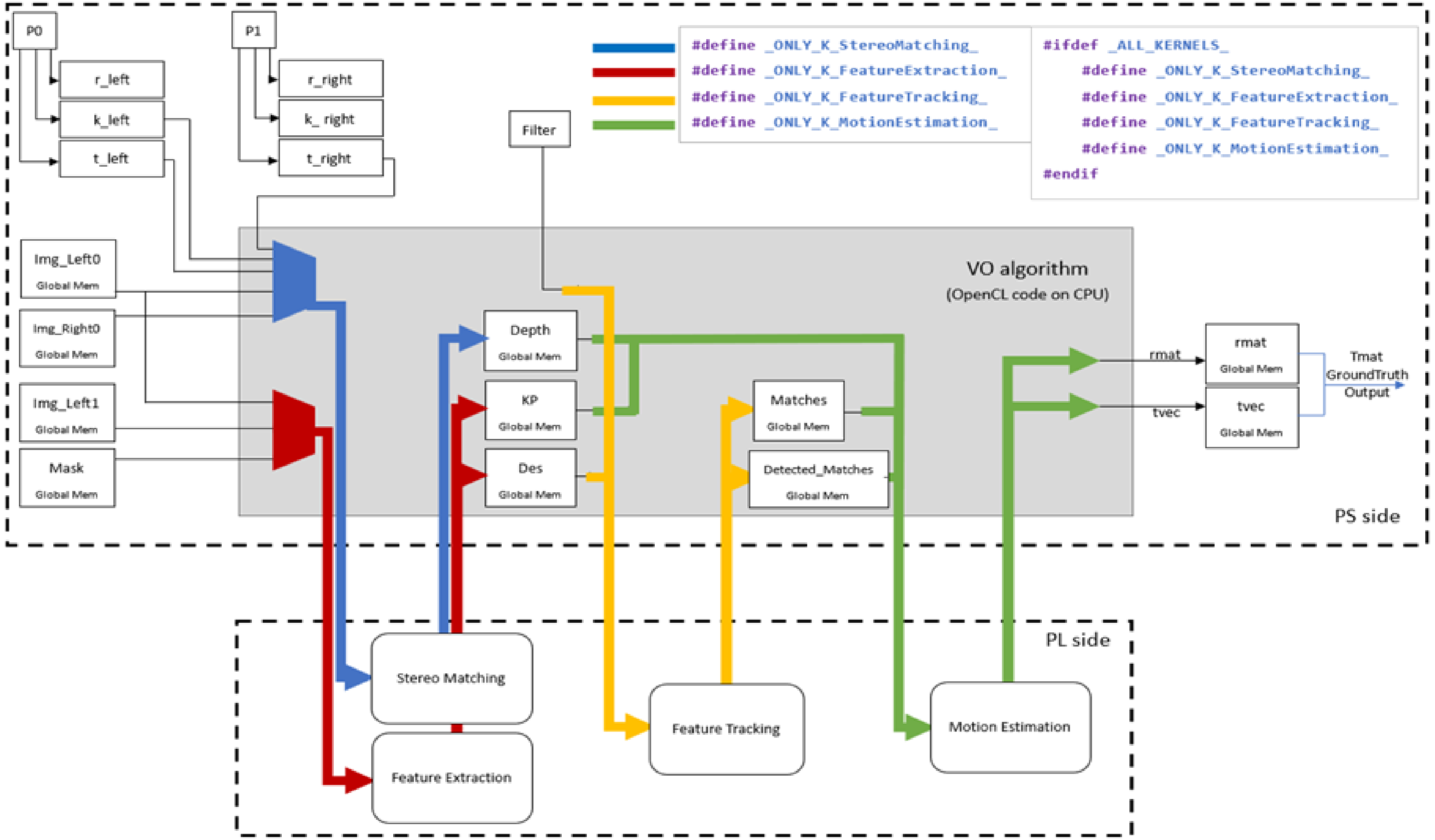
Step#2: Reproduce the Github result

- Compile and run the application to replicate the result
  - What if the Github content is not complete, or the tool version mismatch ?
- Compiler & Link with GNU Compiler tool chain, make it also run in Vitis IDE environment
  - https://github.com/bol-edu/robotics-computing?tab=readme-ov-file#toolchain-and-prerequisites
- Establish a baseline application performance
  - Measure Running Time
    - Profiling tools : valgrind, callgrind, gprof
    - Use <time.h>, clock()
  - Measure Throughput
  - Determine if it is memory-bound
  - Establish Overall Acceleration Goals

Step#3: Identify function to accelerate – host / kernel partition

- Based on profiling result
- Identify Acceleration Potential
  - What is the parallelism of the function? Task-level, data-level
  - Computational Intensity? #-of-op / #-of-words-moved-from-slow-memory
  - Data access locality ? Spatial / Temporal locality
  - Bottleneck come from I/O, computation, data movement or synchronization ?
    - PCIe throughput of 10GBs, software throughput of 50MBs, acceleration factor?

Step#3-1: Design partition and its Verification Methodology



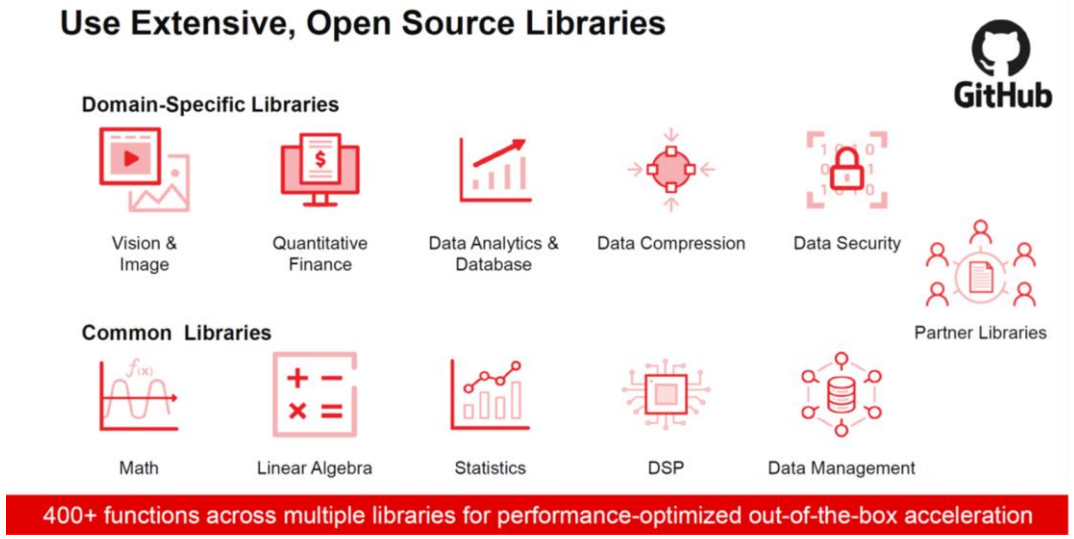
```
#define _ONLY_K_StereoMatching_
#define _ONLY_K_FeatureExtraction_
#define _ONLY_K_FeatureTracking_
#define _ONLY_K_MotionEstimation_
```

```
#ifdef _ALL_KERNELS_
    #define _ONLY_K_StereoMatching_
    #define _ONLY_K_FeatureExtraction_
    #define _ONLY_K_FeatureTracking_
    #define _ONLY_K_MotionEstimation_
#endif
```

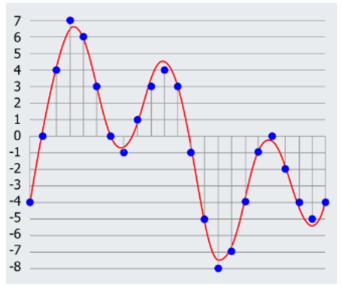
- **Each subfunction (Stereo Matching, Feature Extraction, Feature Tracking & Motion Estimation) has C and HLS-C version.**
- **Allow individual subfunction separately tested, concurrent development, including C-sim, Co-sim, and FPGA validation.**
- **Test data for each module is extracted from integrated test.**

Step#4: Make kernel code self-contained, i.e. no library call

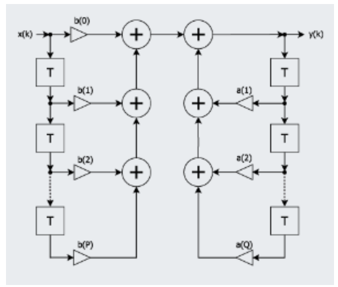
- Use of HLS Library
  - Vitis Accelerated Library: https://github.com/Xilinx/Vitis_Libraries
  - Catapult HLSLibs: https://hlslibs.org/
- Reference above library source code, rewrite and optimize to your use
- Ensure the kernel code is self-contained, i.e. no library call


Use Extensive, Open Source 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
400+ functions across multiple libraries for performance-optimized out-of-the-box acceleration

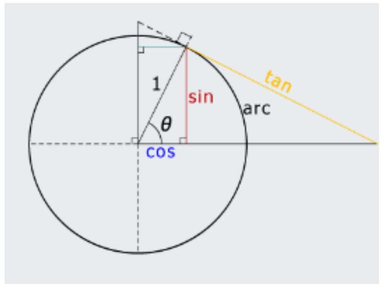
AC DataType



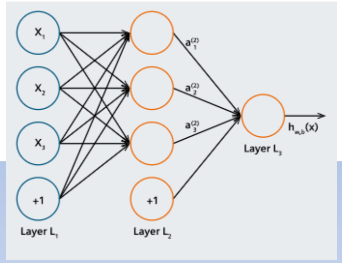
AC DSP



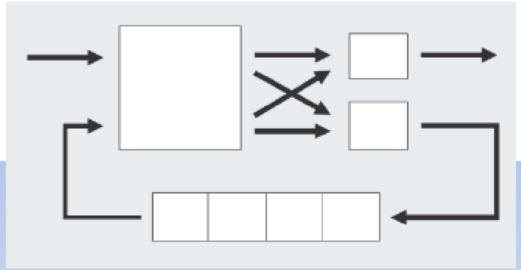
AC Math



AC ML



MATCHLIB Connection



Step#4-1: How to extract self-contained kernel code?

Open-source application code may contain complicated deep function calls. How to extract self-contained kernel code?

1. Elimination - based on the whole package code, investigate and remove unnecessary code.

2. Addition - Based on the algorithm datapath, only add the code to implement the function.

Which method will you use?

Reference: SGBM development log

Step#5: Run C-simulation, the result should match step#2

- Mismatch could happen due to
  - Different Bit-accuracy
  - Float/Double handling
    - Accumulation of rounding error
    - Library function approximations
    - FPU support of extended precision affect on rounding of results
    - Constant propagation/folding effects
    - Handling of subnomals
  - Datapath difference
- Define allowed error tolerance

Step#6: Convert to Synthesizable HLS

- Unsupported C/C++ constructs
  - No system calls to the operating system
  - No dynamic memory allocation, use fixed array
    - The C/C++ constructs must be of a fixed or bounded size.
    - The implementation of those constructs must be unambiguous.
  - Pointer usage
- Add interface pragma
- Dataflow Architecture
- Datatype optimization

Ref: synthesis issues: Feature extraction:

https://github.com/bol-edu/robotics-computing/blob/main/feature-extraction/ppt/feature%20extraction%20log.pdf

Step#7: Run Co-sim with generated RTL code – compare with Step#5

- Cosim runs very slow, minimize the data size
- Use timeline trace 、schedule viewer and waveform to investigate if the execution as expected.
- Possible causes of mismatch
  - Register/storage initialization
  - Data dependency
  - Buffer overflow
  - ….

Step#8: Performance / Area optimization by analyzing timing profile

- Using analysis tool
  - HLS Report
  - Profile Summary
  - Application Timeline
  - Waveform viewer
- Use dataflow
- Maximum Memory bandwidth
- Refer to the following ppt
  - hls-best-practice
  - hls-memory-architecture
  - hls_architecture-design-examples

Step#9: FPG Validation

- Run individual kernels on FPGA before integrating all kernel on FPGA
- Possible Issues
  - Exceed FPGA resource
- Performance comparison: CPU v.s. FPGA