XILINX

Vitis Tutorials

Advanced RTL Kernel Integration

Gang Yuan

Sr. Staff Technical Engineer

Xilinx

Adaptive Computing Acceleration with Alveo

Computational Storage

Database and Data Analytics

Financial Technology

High Performance Computing

Network Acceleration

Video and Imaging

Machine Learning

Tools and Services

Alveo U50

Alveo U200

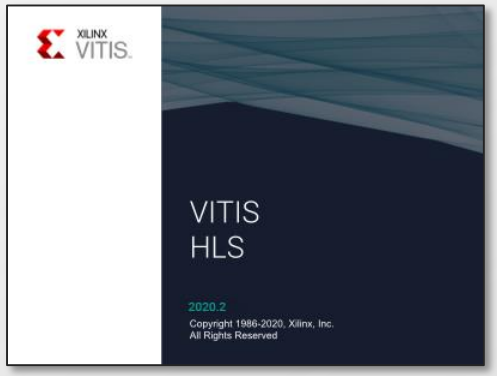
Alveo U250

Alveo U280

Versal

XILINX

Inclusive Hardware Design Methodology



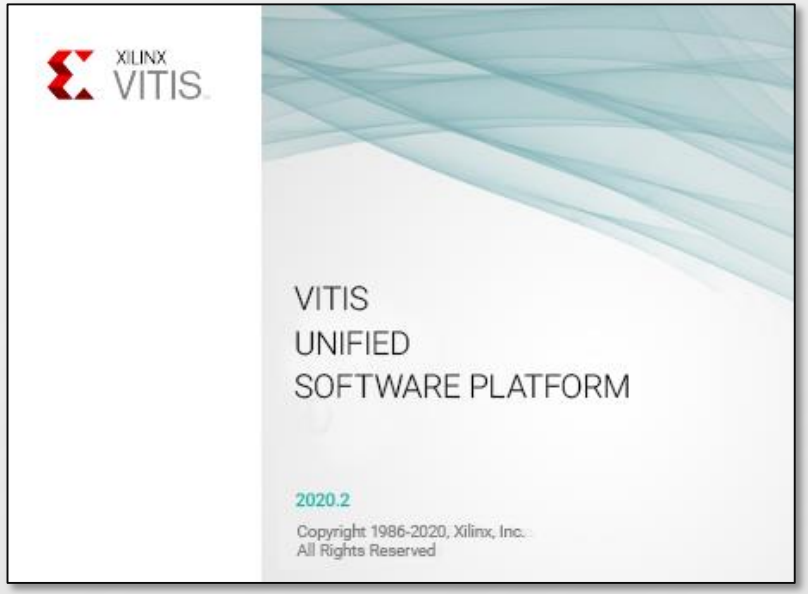
High Level Synthesis (HLS) Modules

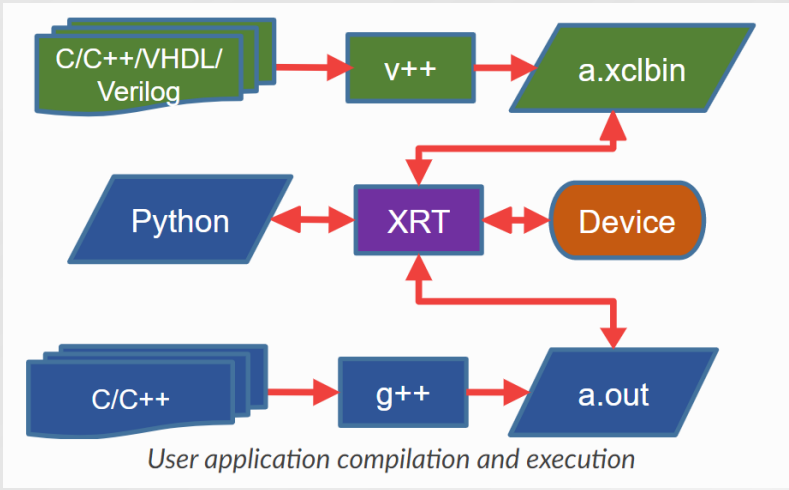
- Image Processing
- Matrix Operation
- Encryption
- Data Compression
- Others..

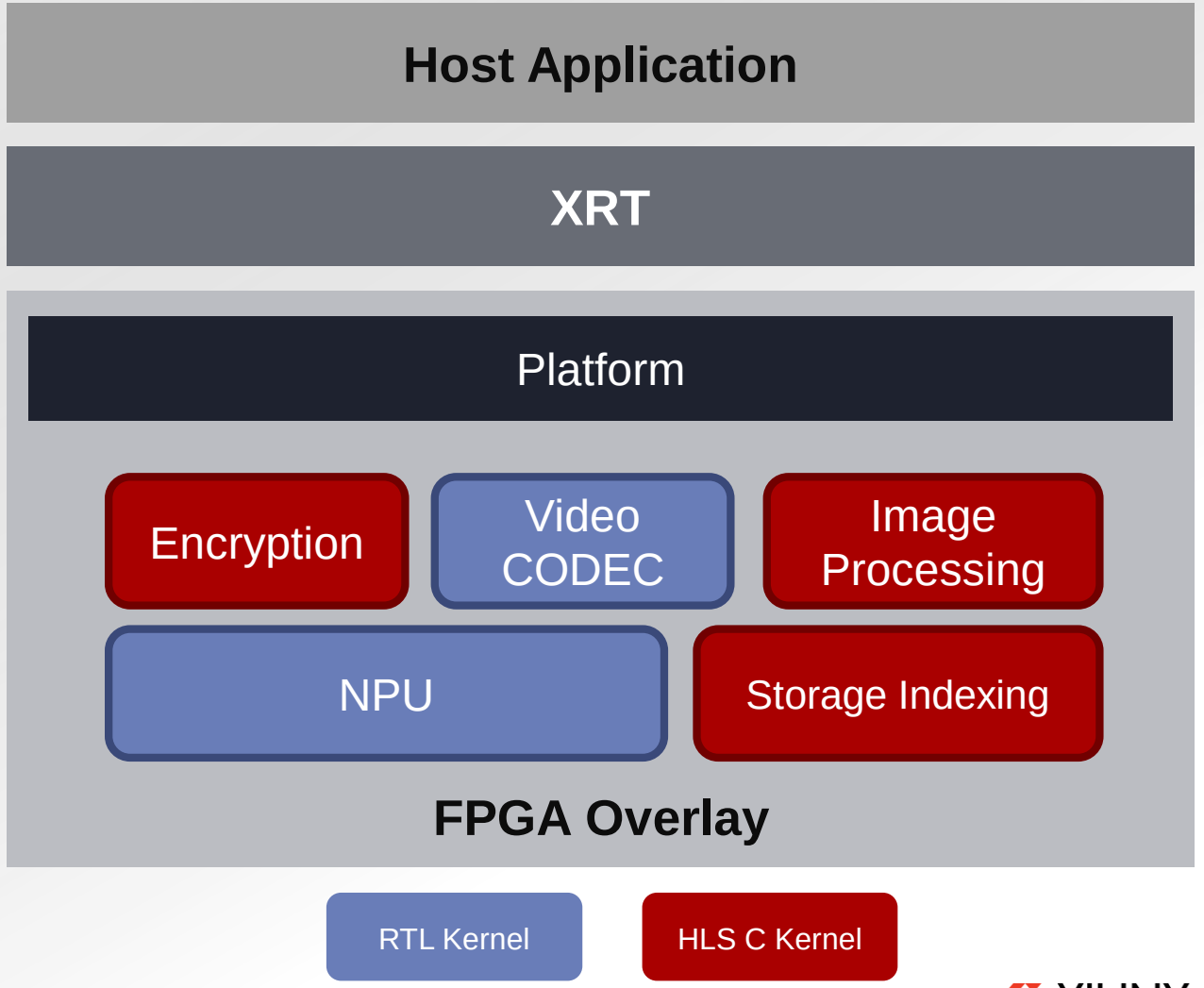
RTL Modules

- Video CODEC
- Image CODEC
- Processor
- NPU
- Others..

Fast Mixed Kernel Integration Flow with Vitis



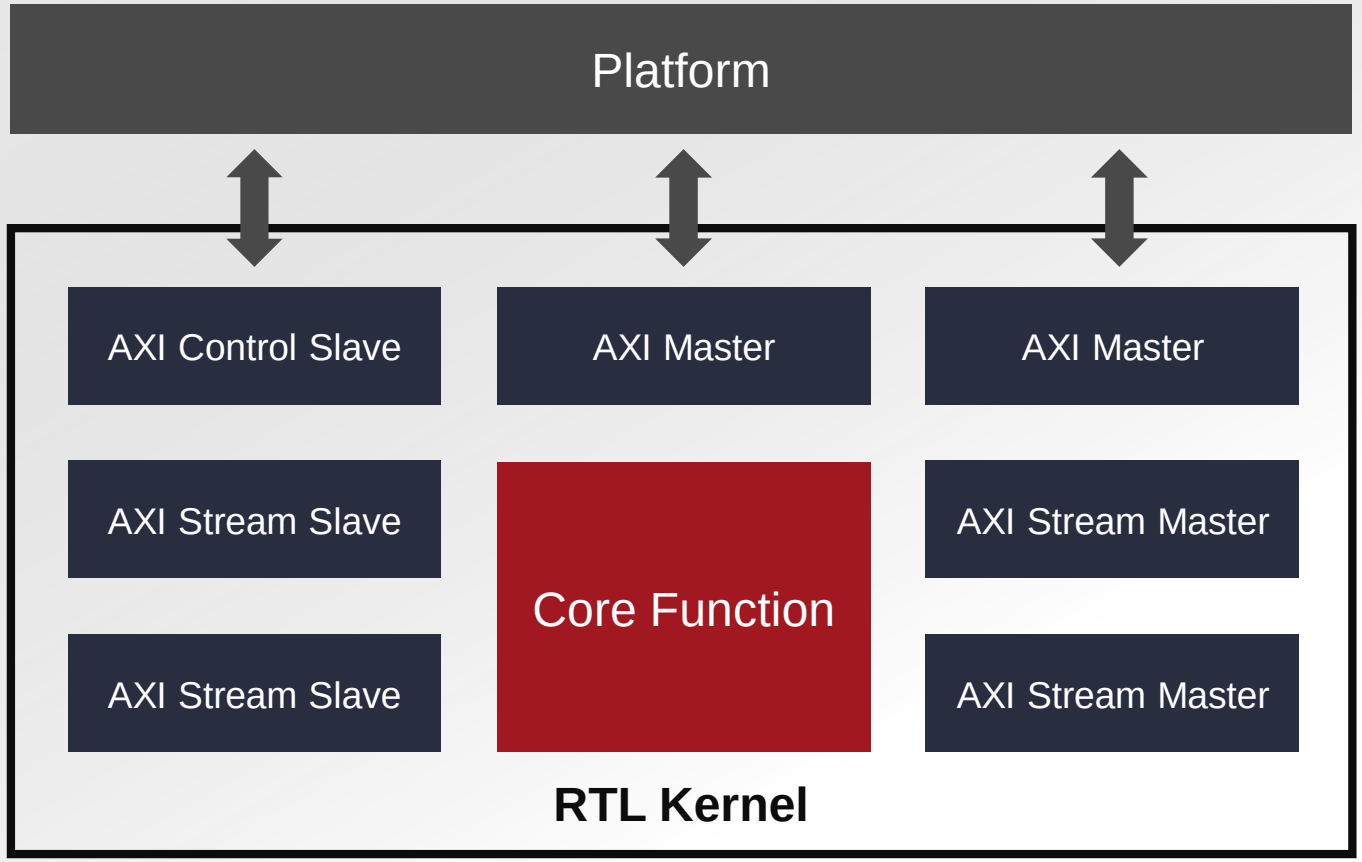




XILINX

Vitis RTL Kernel Bus Interface

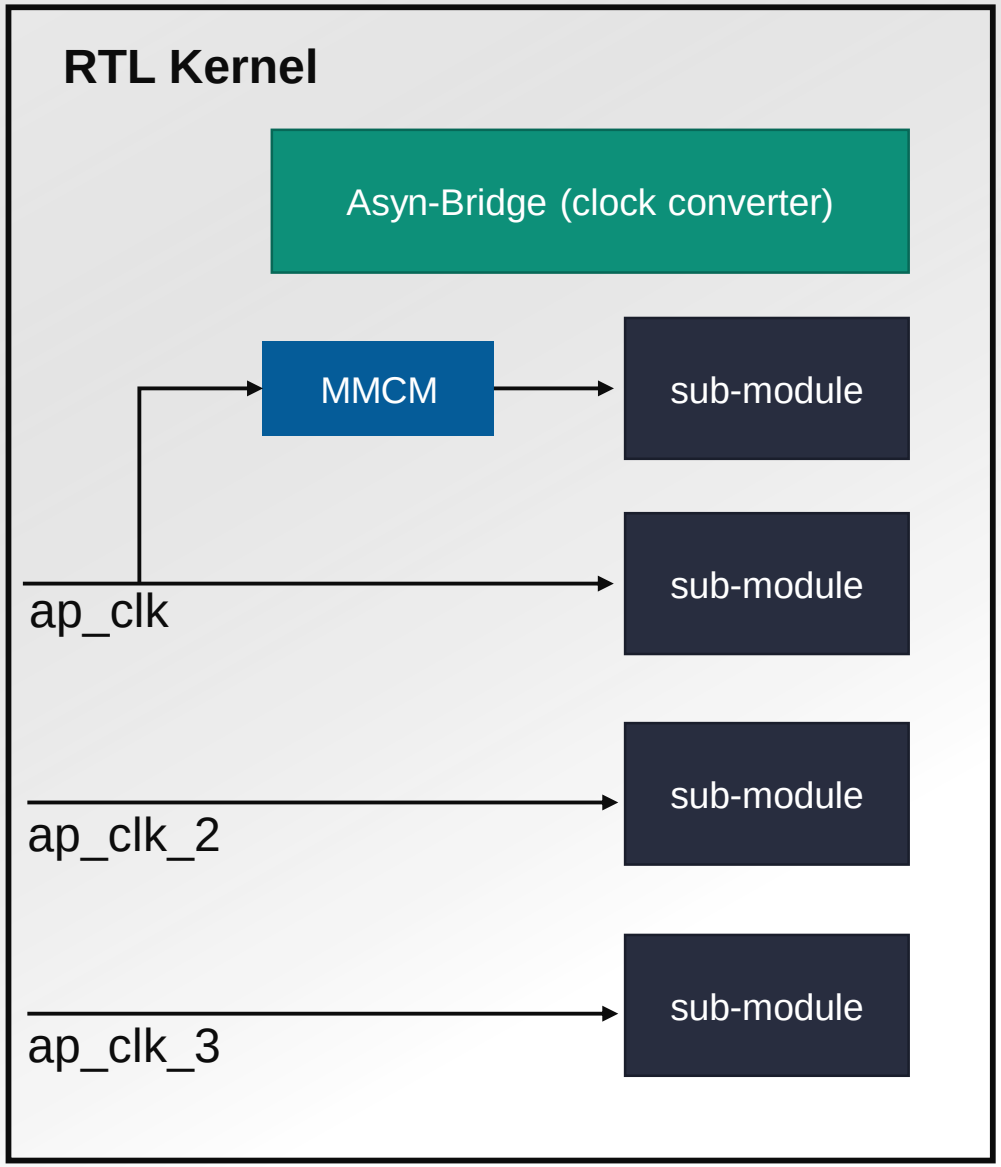
- *Zero or one AXI control slave port*
  - Control registers
  - Memory buffer pointers
  - Kernel start/stop control
- *Zero, one or more AXI master ports*
  - Read/write data buffer in on-board global memory
  - Read/write data buffer in host memory (with Slave Bridge support in the latest platforms)
- *Zero, one or more AXI stream ports*
  - Exchange data between kernels
  - Exchange data between kernel and host (with QDMA support in the latest platforms)



Clocking for RTL Kernel

- *Standard clocks provide by Platform*
  - *ap_clk (300MHz default for U200)*
  - *ap_clk_2 (500MHz default for U200)*
- *Additional clocks during Vitis*
  - *ap_clk_3*
  - *ap_clk_4*
  - *…*
- *Internal clock generated by MMCM/PLL*

The platform use ap_clk as the bus clock, so you may need to use asynchronous bridge if your AXI ports use the internal generated clock.

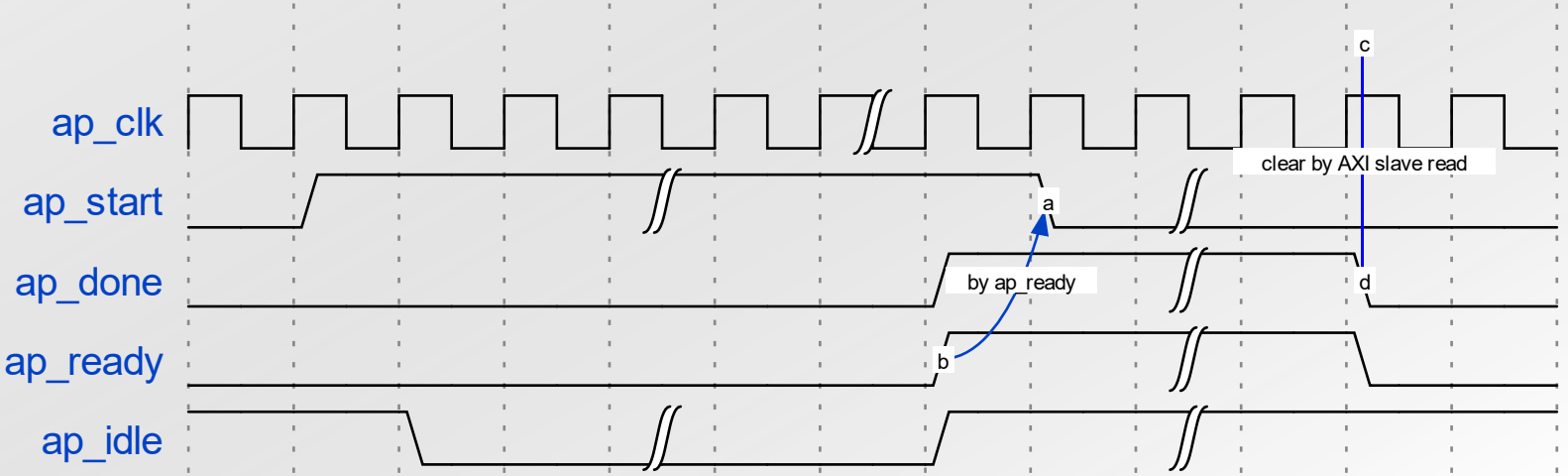


Vitis RTL Kernel Standard Execution Model

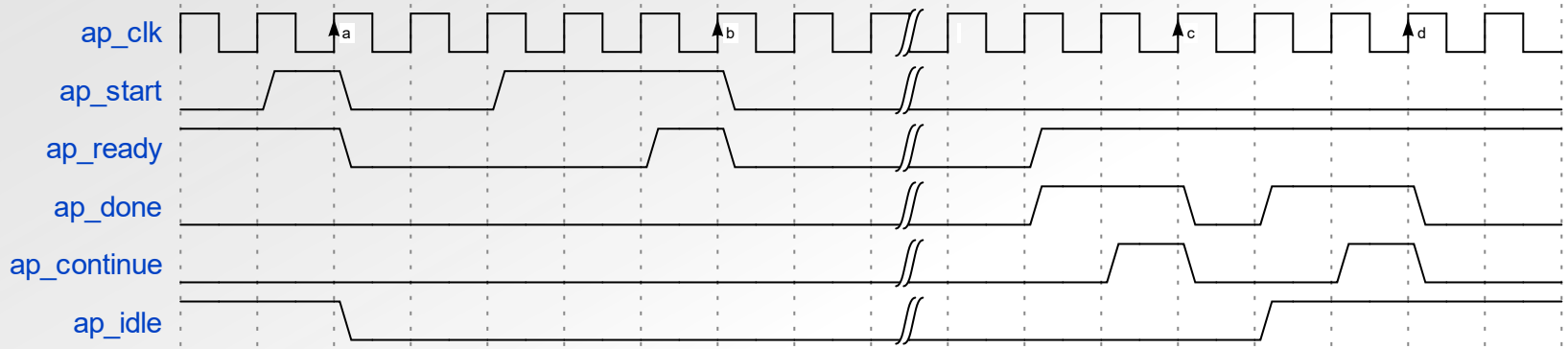
ap_ctrl_none
- No host control
- Free-running

ap_ctrl_hs
- Controlled via AXI slave
- Hand-shake protocol
- Sequential kernel
- Input/output coupled

AXI slave register signals:
ap_start, ap_done, ap_ready, ap_idle, ap_continue, ap_idle



ap_ctrl_chain
- Controlled via AXI slave
- Extension to *ap_ctrl_hs*
- Pipelined kernel
- Input/output decoupled

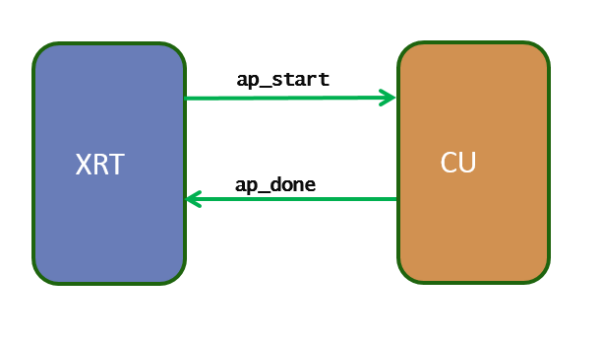


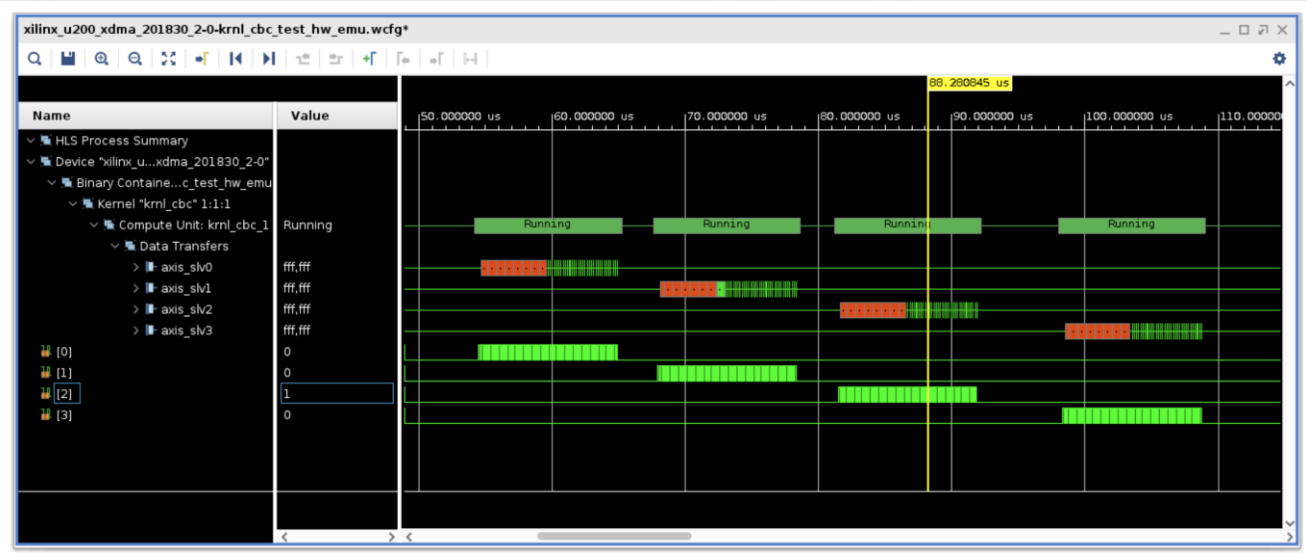
XILINX

Execution Model: ap_ctrl_hs v.s. ap_ctrl_chain

ap_ctrl_hs

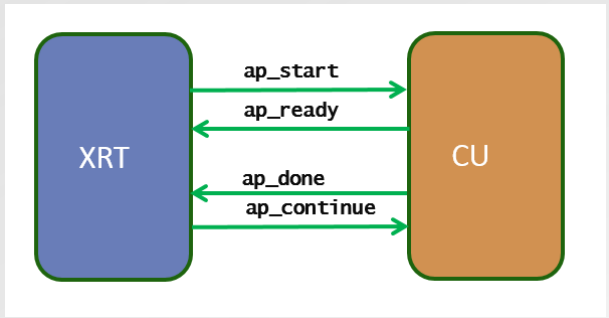
- handshake signals: **ap_start, ap_done**

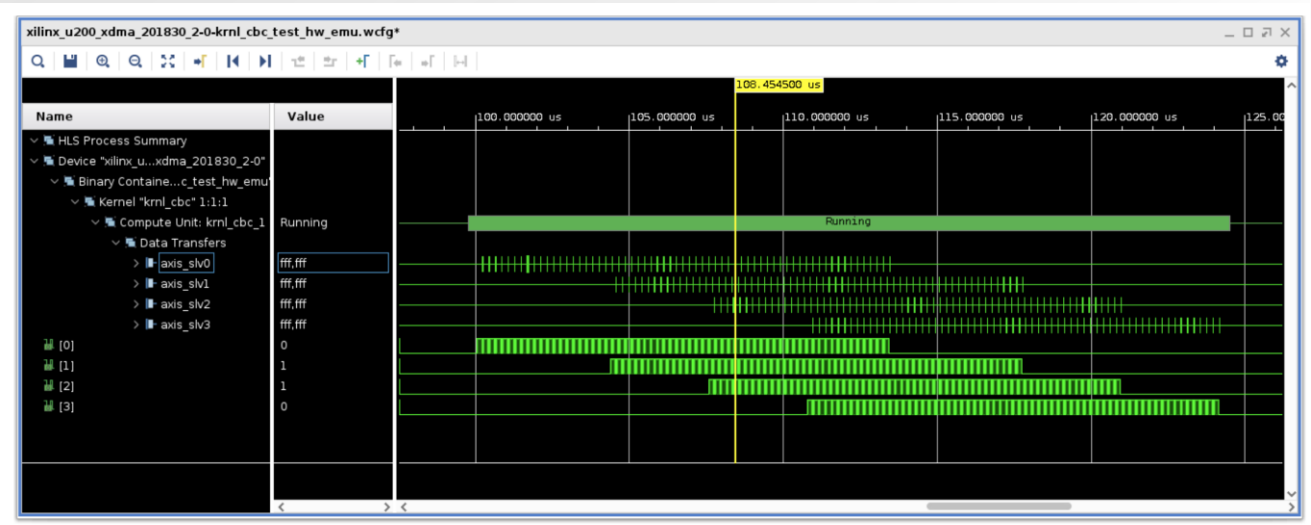




ap_ctrl_chain

- input handshake signals:
  ap_start, ap_ready
- output handshake signals:
  ap_continue, ap_done



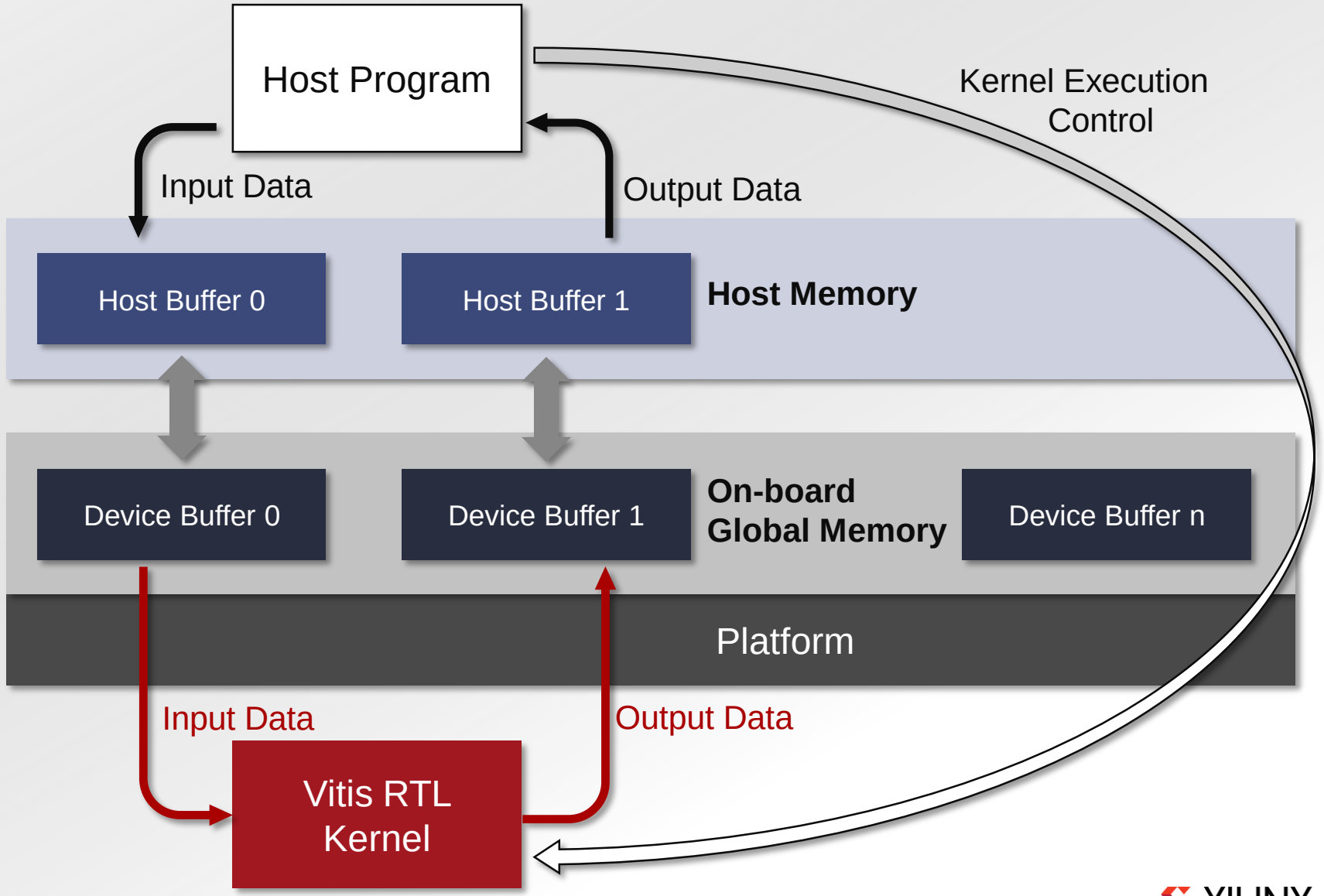


XILINX

RTL Kernel Host Control Flow Example

ap_ctrl_hs, ap_ctrl_chain:

1. Allocate host buffer
2. Allocate device buffer
3. Set Kernel arguments
4. Transfer input data from host buffer to device buffer
5. Trigger Kernel Execution
6. Waiting for Kernel Finish
7. Transfer output data from device buffer to host buffer



RTL Kernel Host Programming

Use easy programming model to control the Vitis RTL kernel execution

XRT Native API

- *xrtPLKernelOpen*
- *xrtBOAllocUserPtr*
- *xrtBOSync*
- *xrtRunSetArg*
- *xrtKernelRun*
- *…*

OpenCL API with Xilinx Extension

- *clCreateKernel*
- *clCreateBuffer*
- *clCreateCommandQueue*
- *clEnqueueMigrateMemObject*
- *clSetKernelArgs*
- *clEnqueueTask*
- *…*

Vitis RTL Kernel Low-level Control

Use Low Level Register Access API to Control Kernel Execution Directly

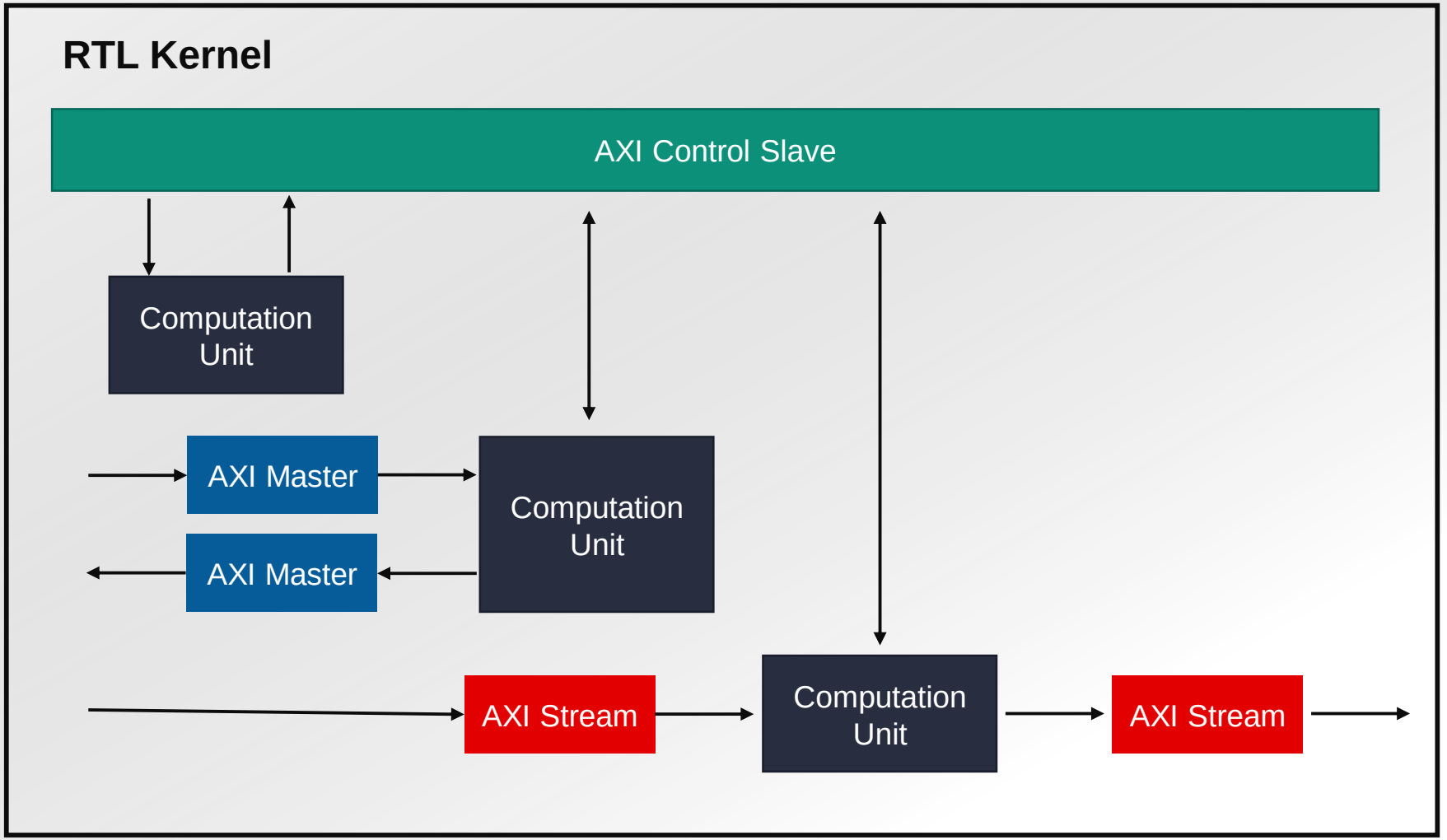
- Configure *ap_none* model kernel parameter
- Re-config kernel arguments without running the kernel
- Control kernel in legacy fashion
- Convenient for porting existing RTL design to Vitis Flow and Alveo Platform

XRT Native API

- *xrt::ip::read_register*
- *xrt::ip:write_register*

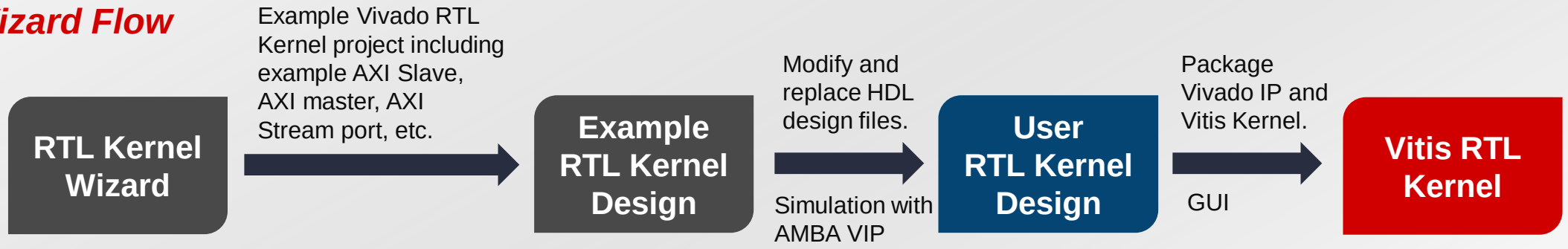
Vitis RTL Kernel Free-style

You can mix different data flow and execution model in an RTL kernel.

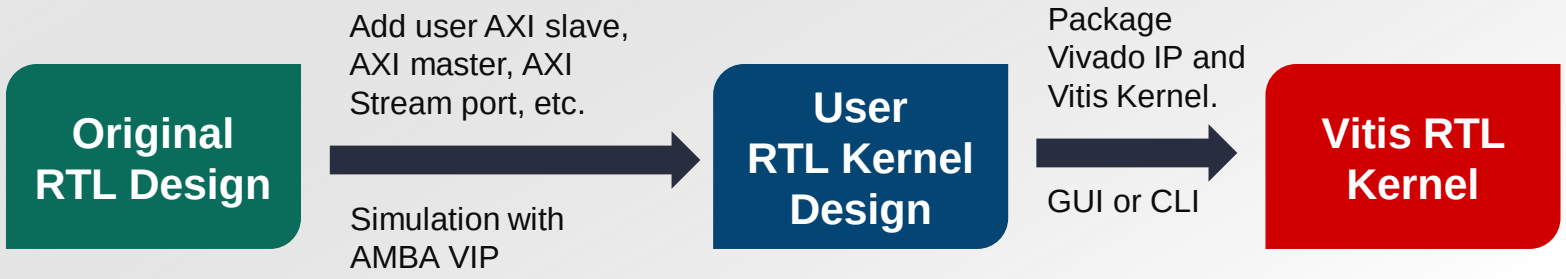


RTL Kernel Development Flow

RTL Kernel Wizard Flow



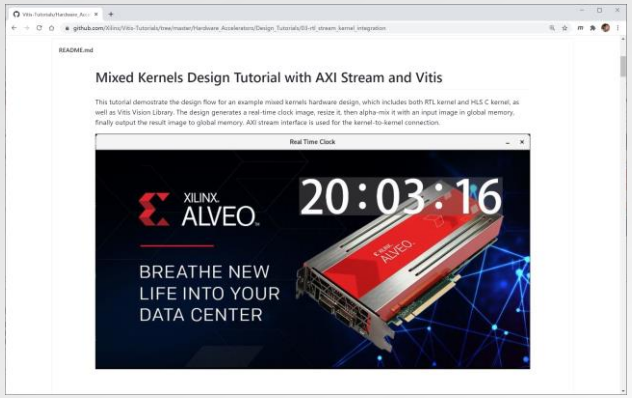
Bottom-up Flow



RTL Kernel Flow Example in Vitis Tutorials

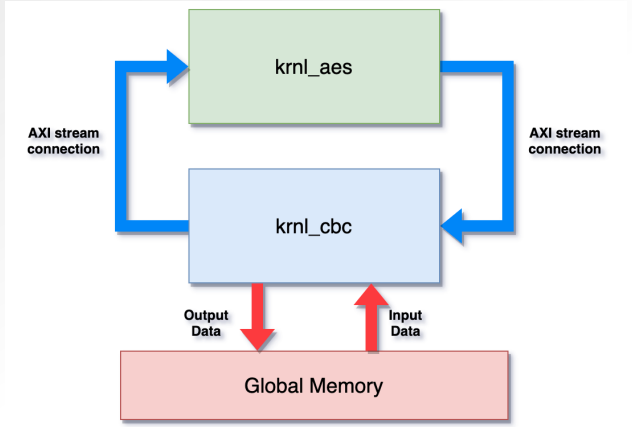
https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Hardware_Accelerators/Design_Tutorials/03-rtl_stream_kernel_integration

RTL Kernel Wizard Based Flow



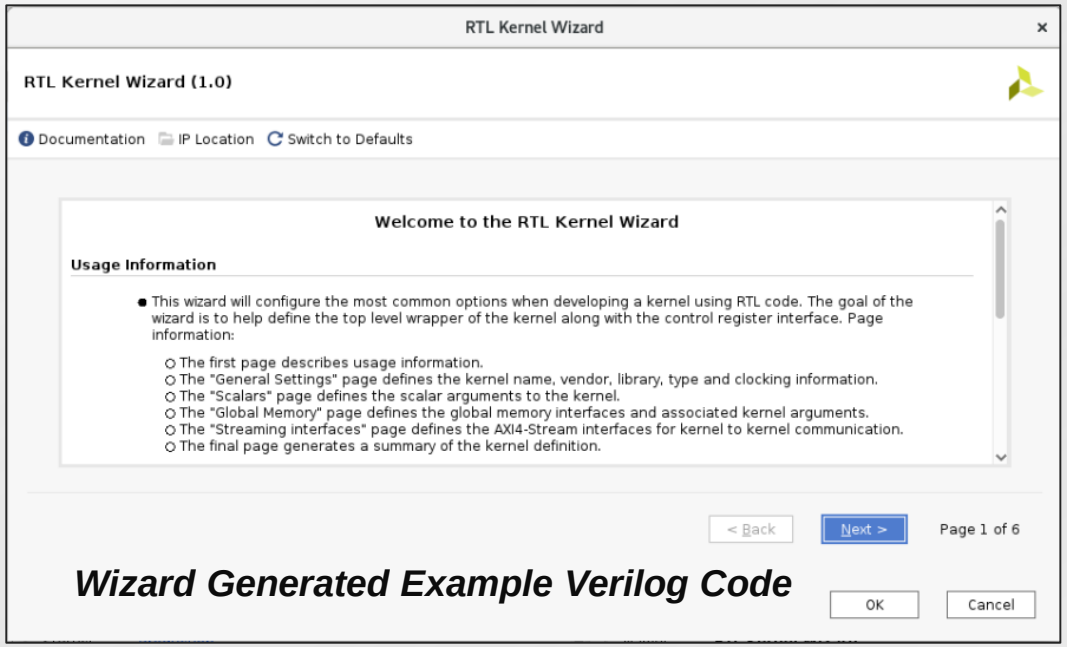
https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Hardware_Accelerators/Design_Tutorials/05-bottom_up_rtl_kernel

Bottom-up Flow with GUI or CLI

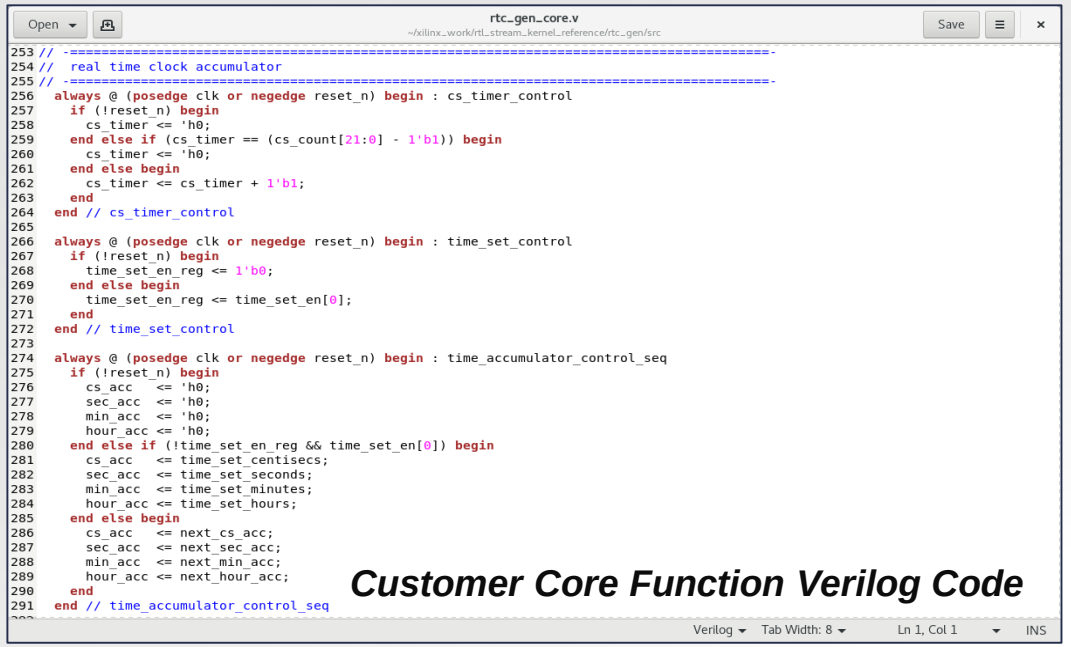


XILINX

RTL Kernel Wizard Flow for Kernel Creation



Wizard Generated Example Verilog Code



Customer Core Function Verilog Code

Vivado Kernel Packing Tool

Use *RTL Kernel Wizard* to start RTL kernel design easily!

Vitis RTL Kernel

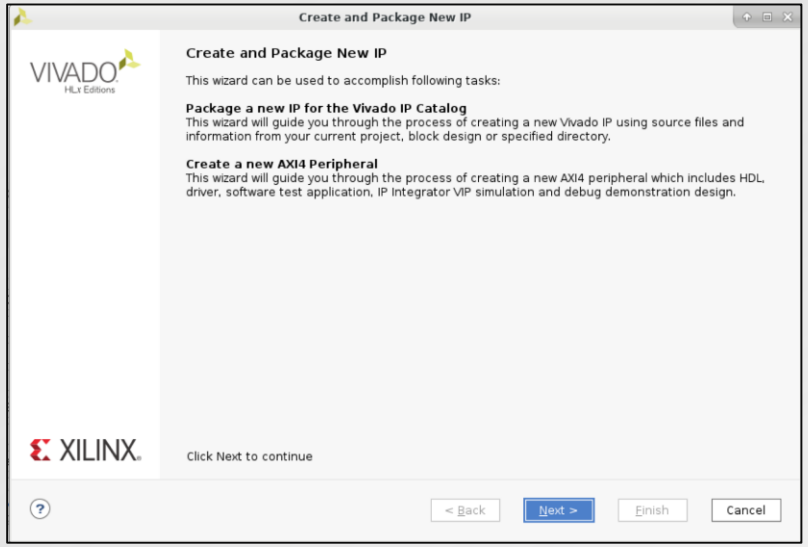
RTL Kernel Wizard **will generate example AXI control slave, example AXI master, integration wrapper, example testbench, …**

XILINX

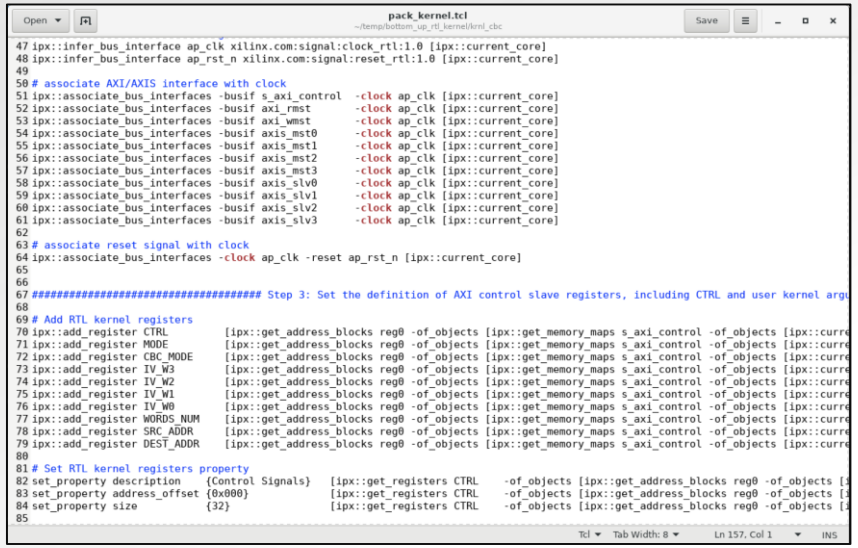
Bottom-up Flow for RTL Kernel Packing

Use Vivado IP Packaging tool to package the kernel

- Both GUI and CLI are supported
- All three kernel execution models are supported
- AXI slave and master interface and automatically inferenced by tool (some requirements exist for AXI signals naming)
- IP/kernel XML files can be generated automatically by tool



Create Vivado project and use IP Packaging tool with GUI



Create Vivado project and use IP Packaging tool with Tcl

Package Vitis RTL Kernel with Tcl

Main Steps

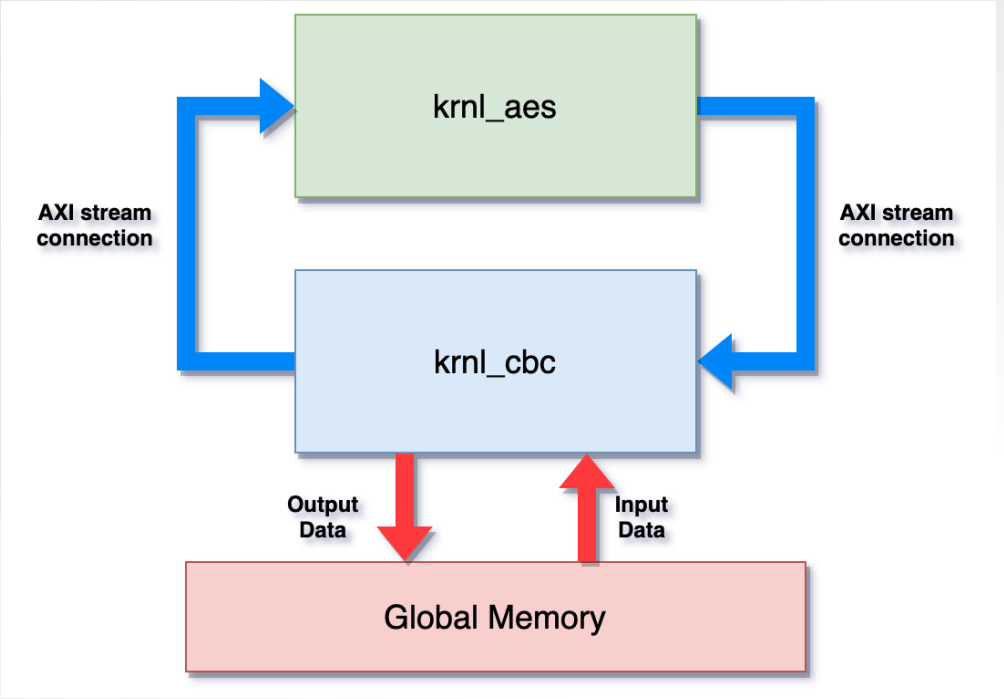
- **Create IP project and IP packaging project**
  - `create_project`
  - `add_files`
  - `ipx::package_project`
- **Inference clock, reset and associate with AXI ports**
  - `ipx::infer_bus_interface`
  - `ipx::associate_bus_interface`
- **Set kernel arguments (registers) definition in AXI control slave**
  - `ipx::add_register`
  - `set_property`
- **Associate AXI master to pointer arguments in AXI control slave**
  - `ipx::add_register_parameter`
  - `set_property`
- **Packaging Vivado IP**
  - `ipx::update_source_project_archive`
  - `ipx::save_core`
- **Generate Vitis kernel file (XO)**
  - `package_xo`

These Tcl commands can run in Vivado Tcl mode, batch more, or GUI mode Tcl console.

Command Line Based Example in Vitis Tutorials

https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Hardware_Accelerators/Design_Tutorials/05-bottom_up_rtl_kernel

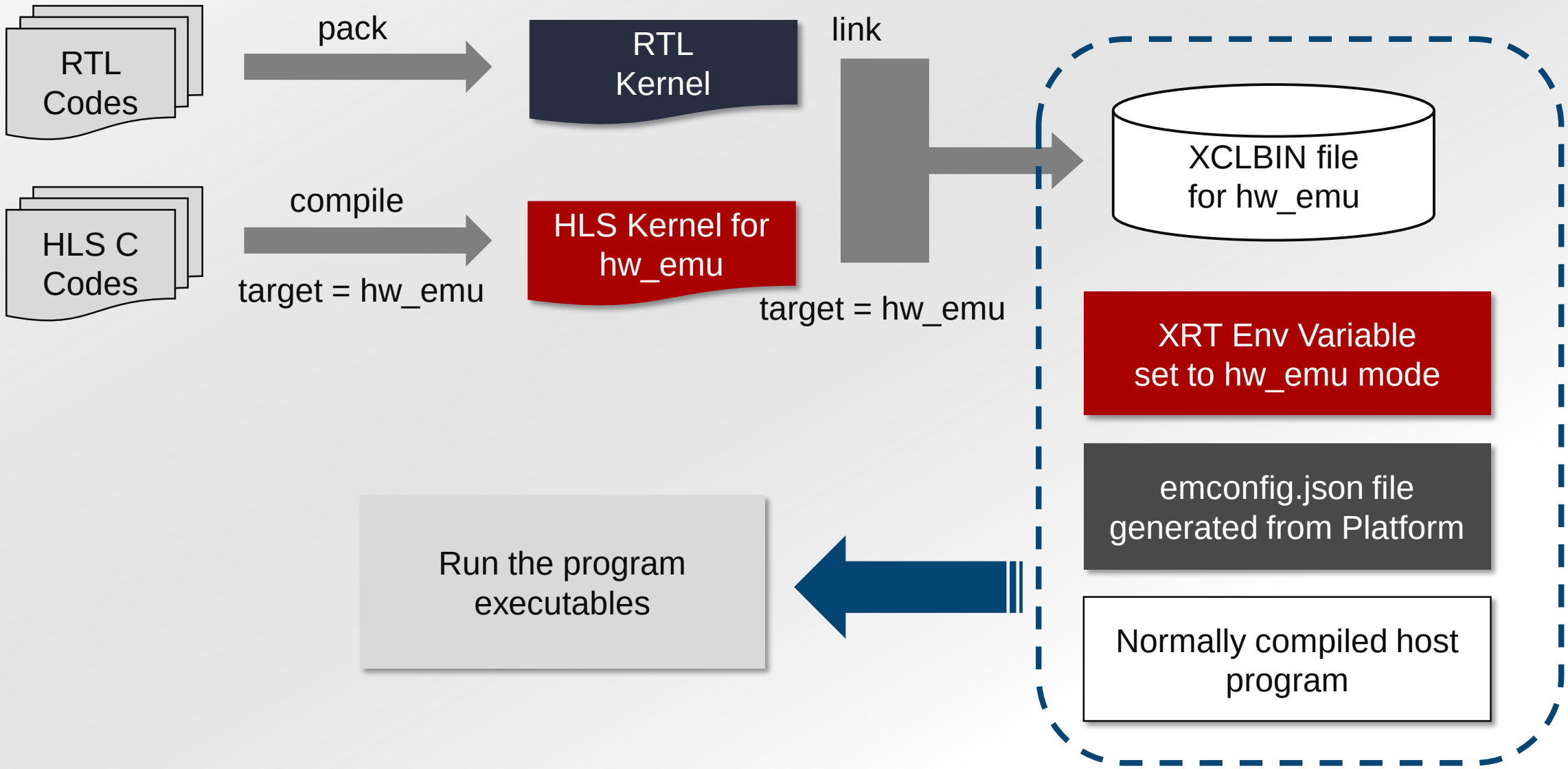
- Realize an Vitis RTL kernel for AES-256 and CBC mode acceleration
- Full CLI flow from RTL kernel creation to final system integration
- Demonstrate the implementation of *ap_ctrl_hs* and *ap_ctrl_chain* mode in RTL kernel



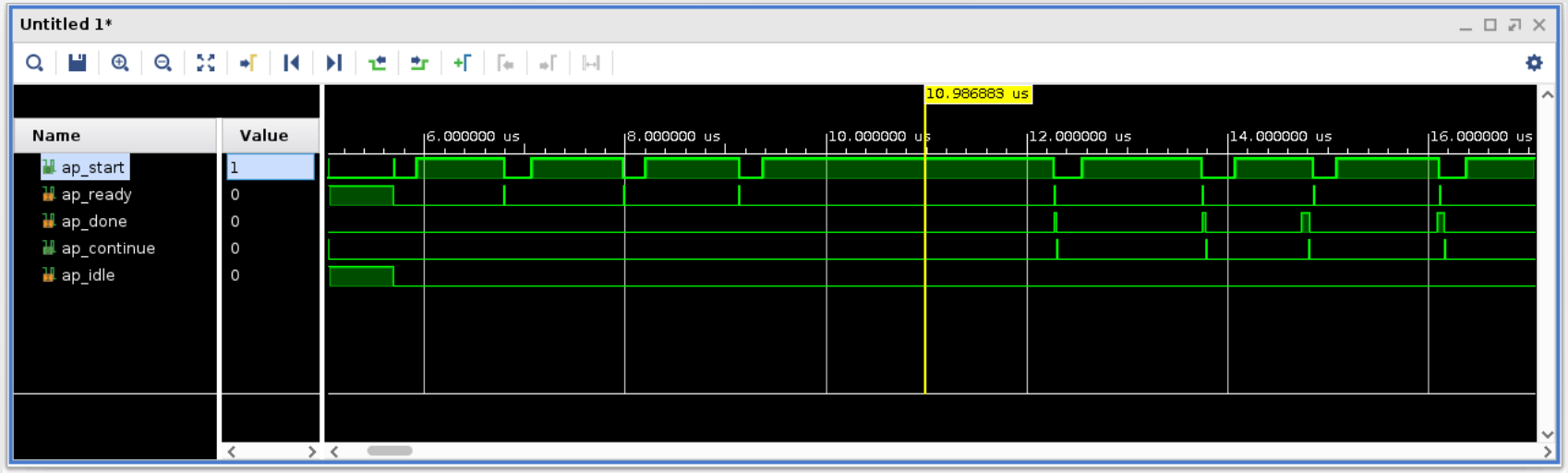
Use Hardware Emulation for Vitis RTL Kernel Designs

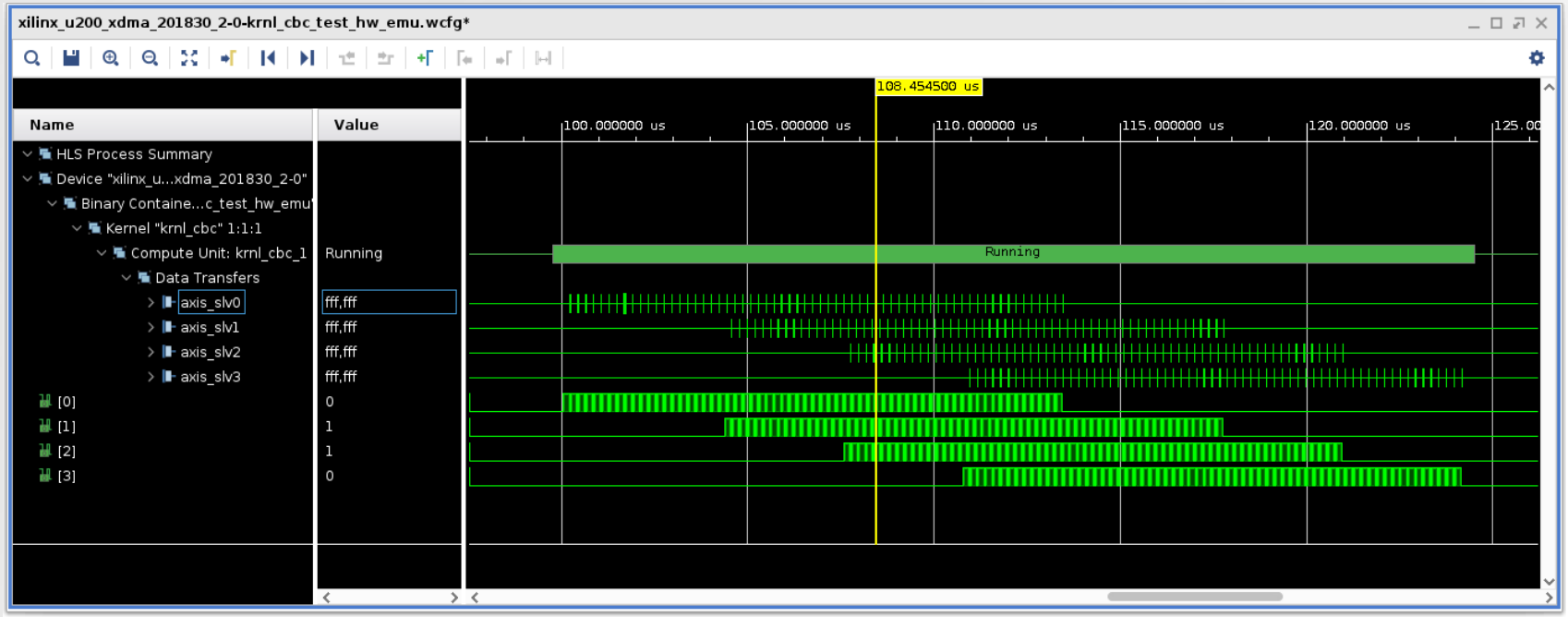
- Verify RTL Kernel in simulation environment close to real board
  - Simulation with complete Alveo card scenario
  - Simulation with real host program
  - Automatically generated testbench and simulation stimulus
- In-depth debug for board-level issues
  - Each signal in RTL kernel can be dumped into waveform
  - Effective for ‘hang’ analyze in board-level running
- Analyze and locate performance bottle-neck
  - Easy to observe host-to-kernel and kernel-to-kernel interactions
  - Easy to analyze kernel running time and dependency lock

Use Hardware Emulation for Vitis RTL Kernel Designs



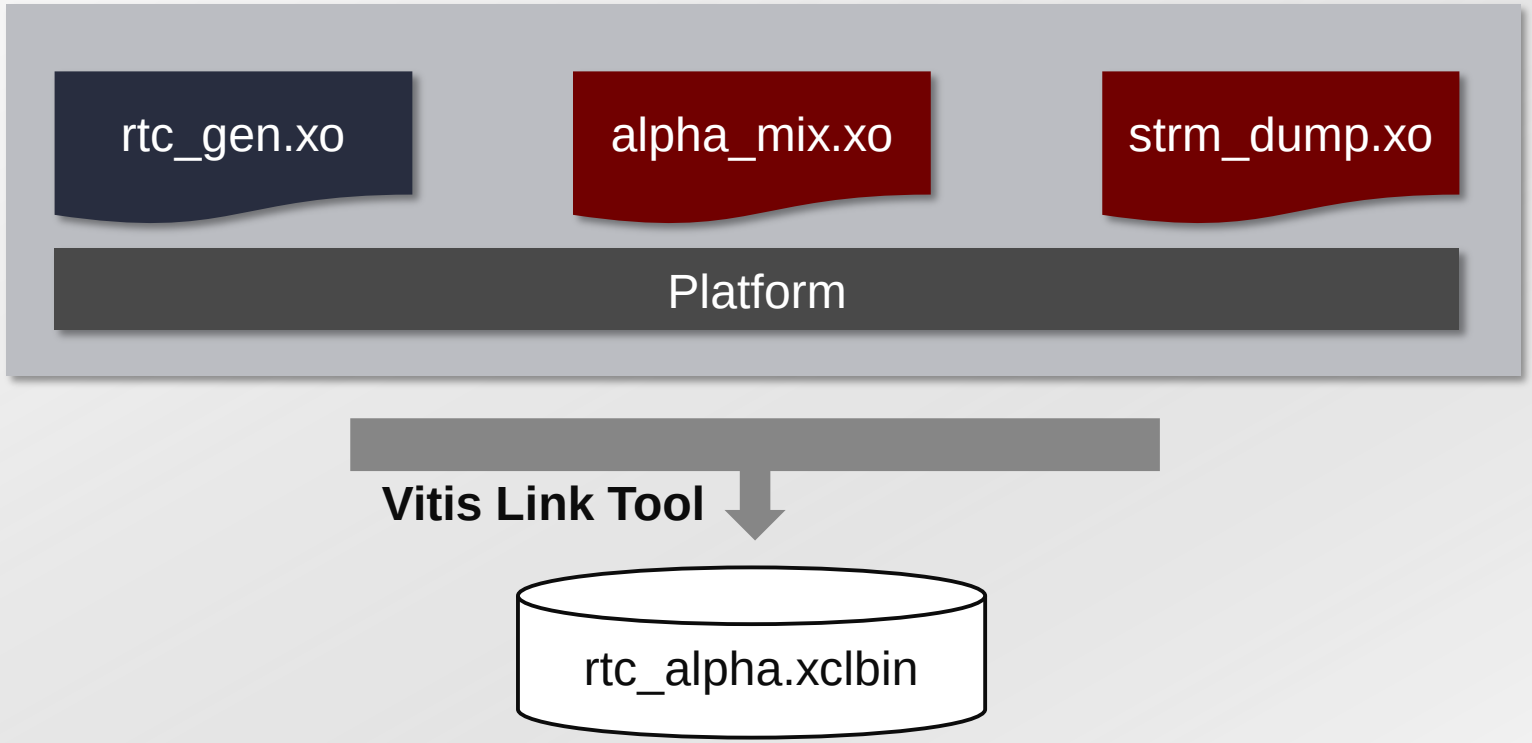
Use Hardware Emulation for Vitis RTL Kernel Designs

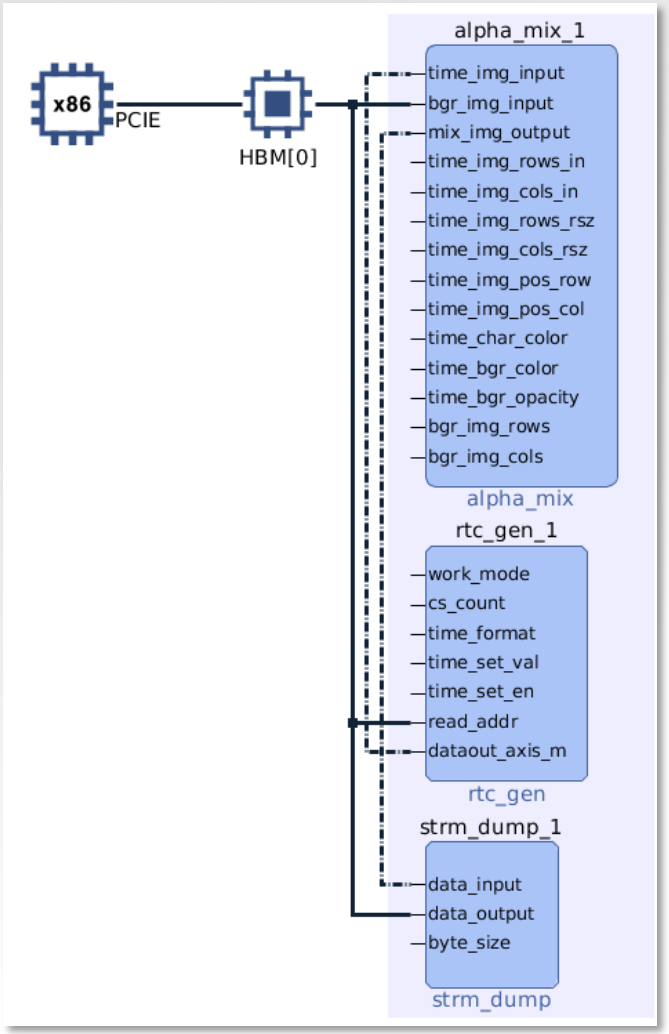




XILINX.

Use Vitis to Build Hardware Overlay (XCLBIN)






```
v++ -l                                          \
    -platform xilinx_u50_gen3x16_xdma_201920_3  \
    -config xclbin_rtc_alpha.ini                \
    -o rtc_alpha.xclbin                         \
    rtc_gen.xo strm_dump.xo alpha_mix.so
```

xclbin_rtc_alpha.ini

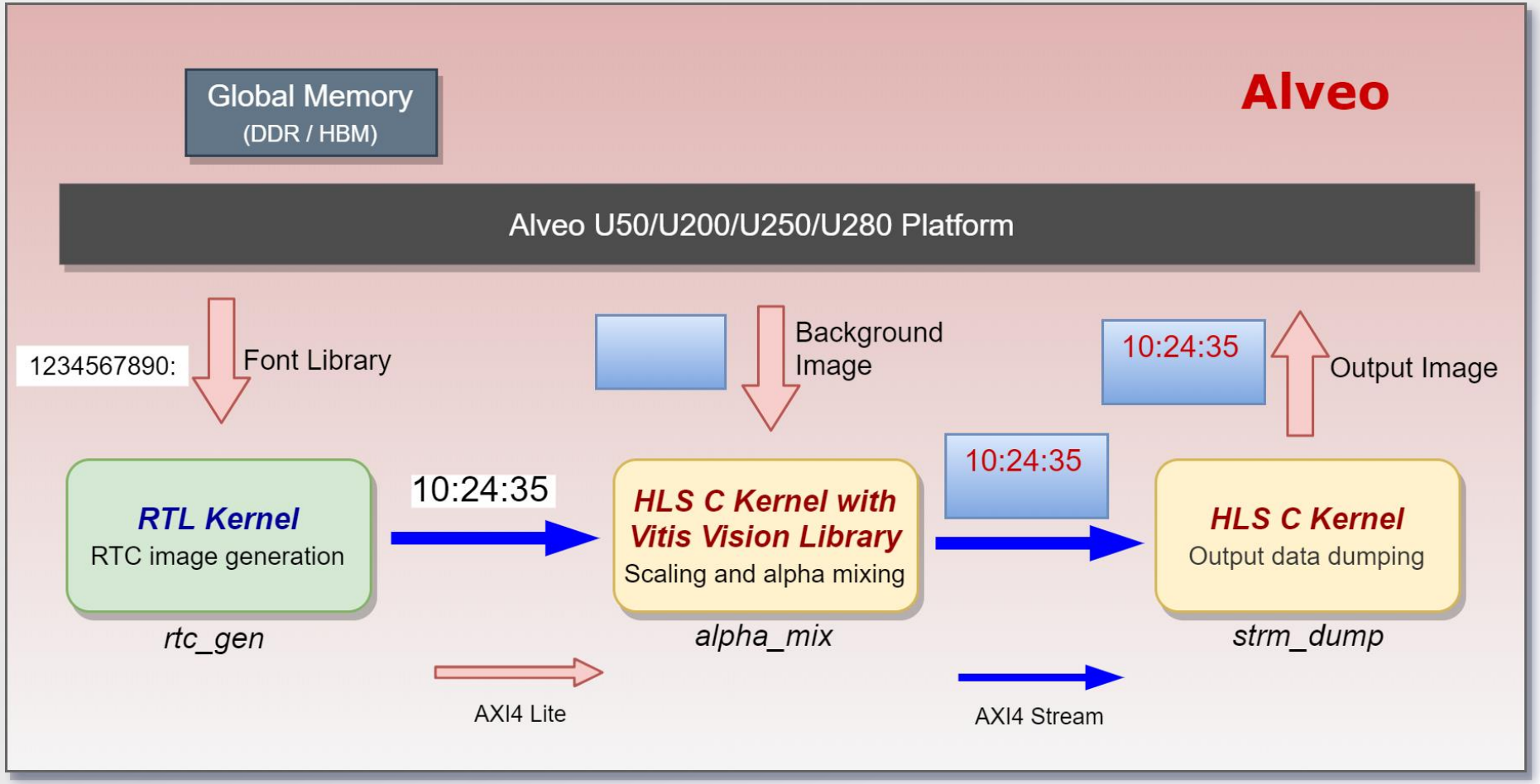
```
[connectivity]
stream_connect=rtc_gen_1.dataout_axis_m:alpha_mix_1.time_img_input
stream_connect=alpha_mix_1.mix_img_output:strm_dump_1.data_input
```

XILINX

Mixed Kernel Design Example in Vitis Tutorials

Mixed HDL/HLS Kernel Integration Example

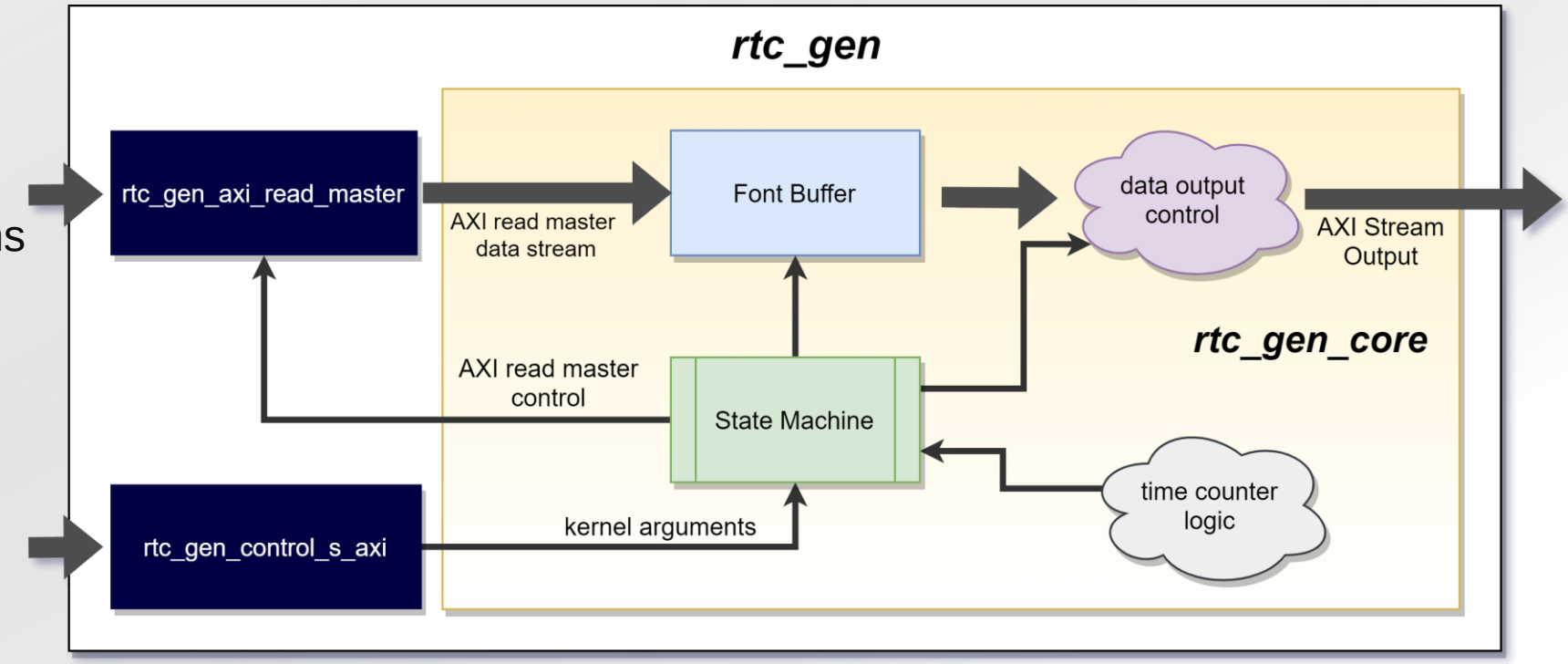
- ***rtc_gen***
  - ✓ Verilog Kernel
- ***alpha_mix***
  - ✓ HLS C Kernel with Vitis Vision Library
- ***strm_dump***
  - ✓ HLS C Kernel



Example RTL Kernel Design

RTL Kernel *rtc_gen*

- *One AXI control slave port*
  - 6 kernel arguments
  - Control interface: ap_ctrl_hs
- *One AXI master port*
  - Data width: 32-bit
- *One AXI stream master port*
  - Data width: 64-bit

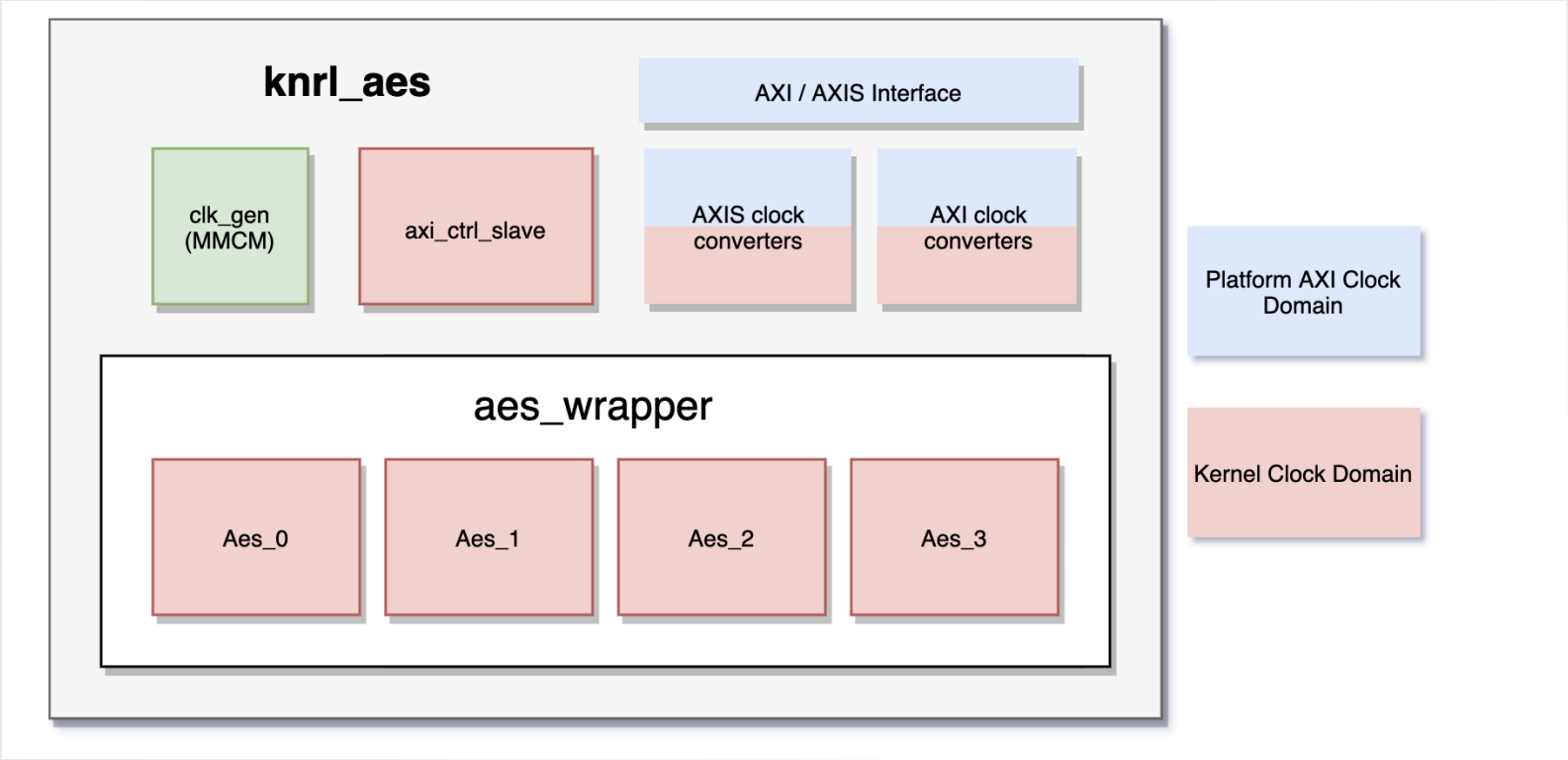


- Generated AXI master and AXI control slave modules are used.
- ***rtc_gen_core*** module is hand coded Verilog modules.
- Generated top level wrapper is modified to instantiate ***rtc_gen_core*** module.

Example RTL Kernel Design

RTL Kernel *krnl_aes*

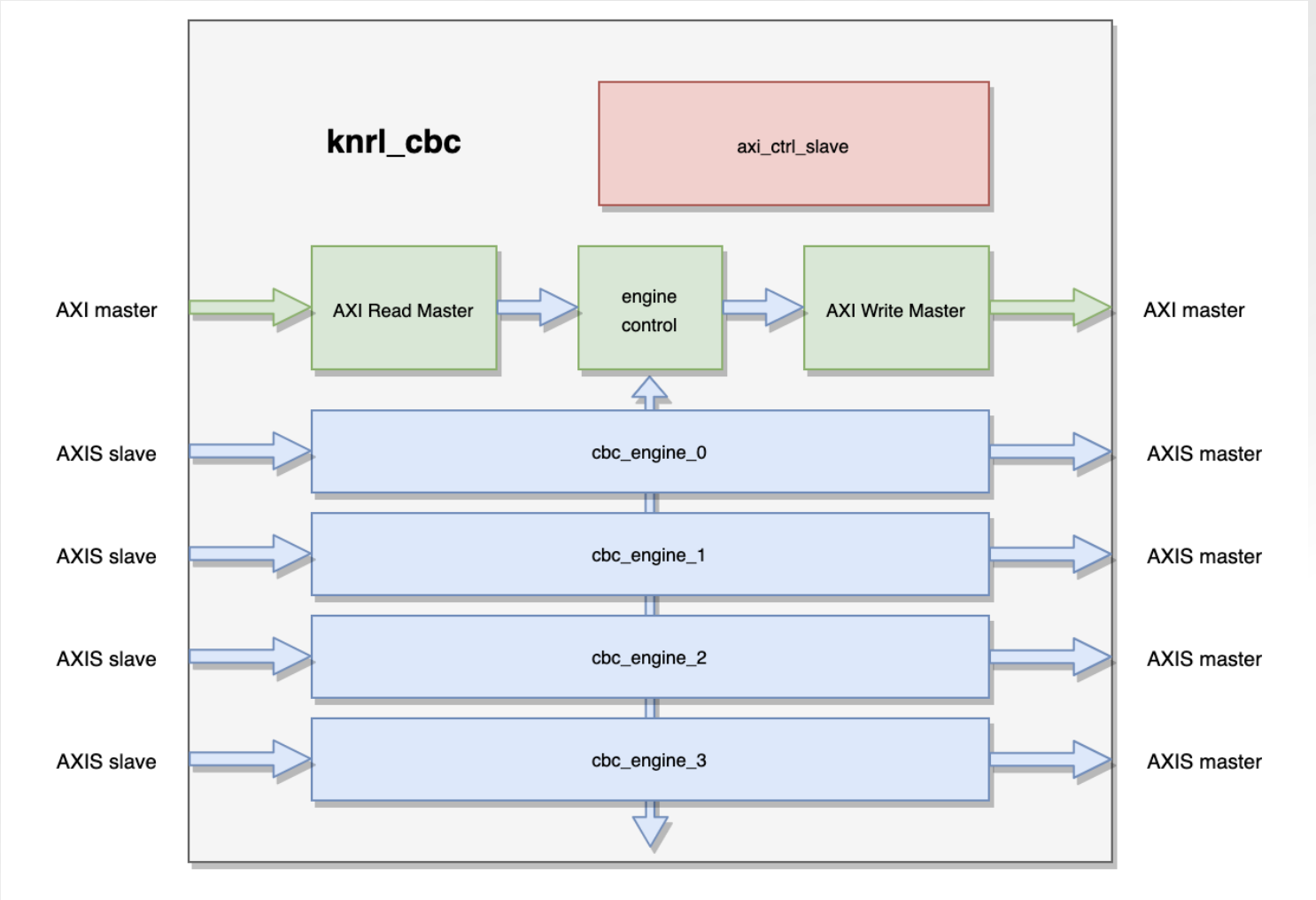
- *One AXI control slave port*
- *Execution Model: ap_ctrl_none, ap_ctrl_hs*
- *One AXI stream slave port*
- *One AXI stream master port*



Example RTL Kernel Design

RTL Kernel *krnl_cbc*

- *One AXI control slave port*
  - Execution Model: ap_ctrl_chain
- *Two AXI master ports*
- *Four AXI stream slave port*
- *Four AXI stream master port*



Summary for RTL Kernel for Acceleration

1. The RTL kernel for Vitis can use AXI slave port for control and use AXI master or AXI stream ports for data transfer. All of these ports are optional according to your design specification.

2. You can use RTL Kernel Wizard to start the RTL kernel design easily if you are new to Vitis.

3. You can use Tcl scripts based command line flow to pack RTL kernel faster and more flexible.

4. You can use any of the three execution model (ap_ctrl_none, ap_ctrl_hs and ap_ctrl_chain), even combination of them in your RTL kernel.

5. Additional MMCM/PLL can be instantiated in the RTL kernel to generated specific clock

Thank you!