



Vitis Tutorials

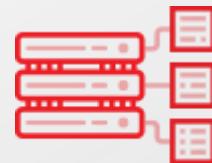
Advanced RTL Kernel Integration

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

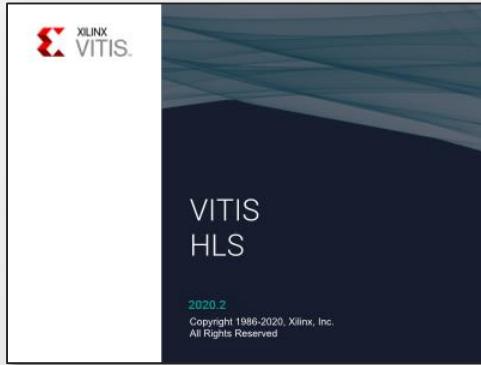


Alveo U280



Versal

Inclusive Hardware Design Methodology



High Level Synthesis (HLS) Modules

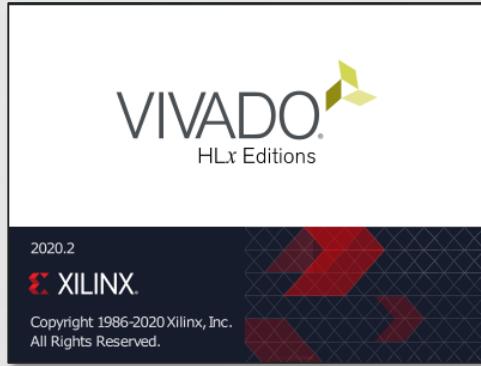
Image
Processing

Matrix
Operation

Encryption

Data
Compression

Others..



RTL Modules

Video
CODEC

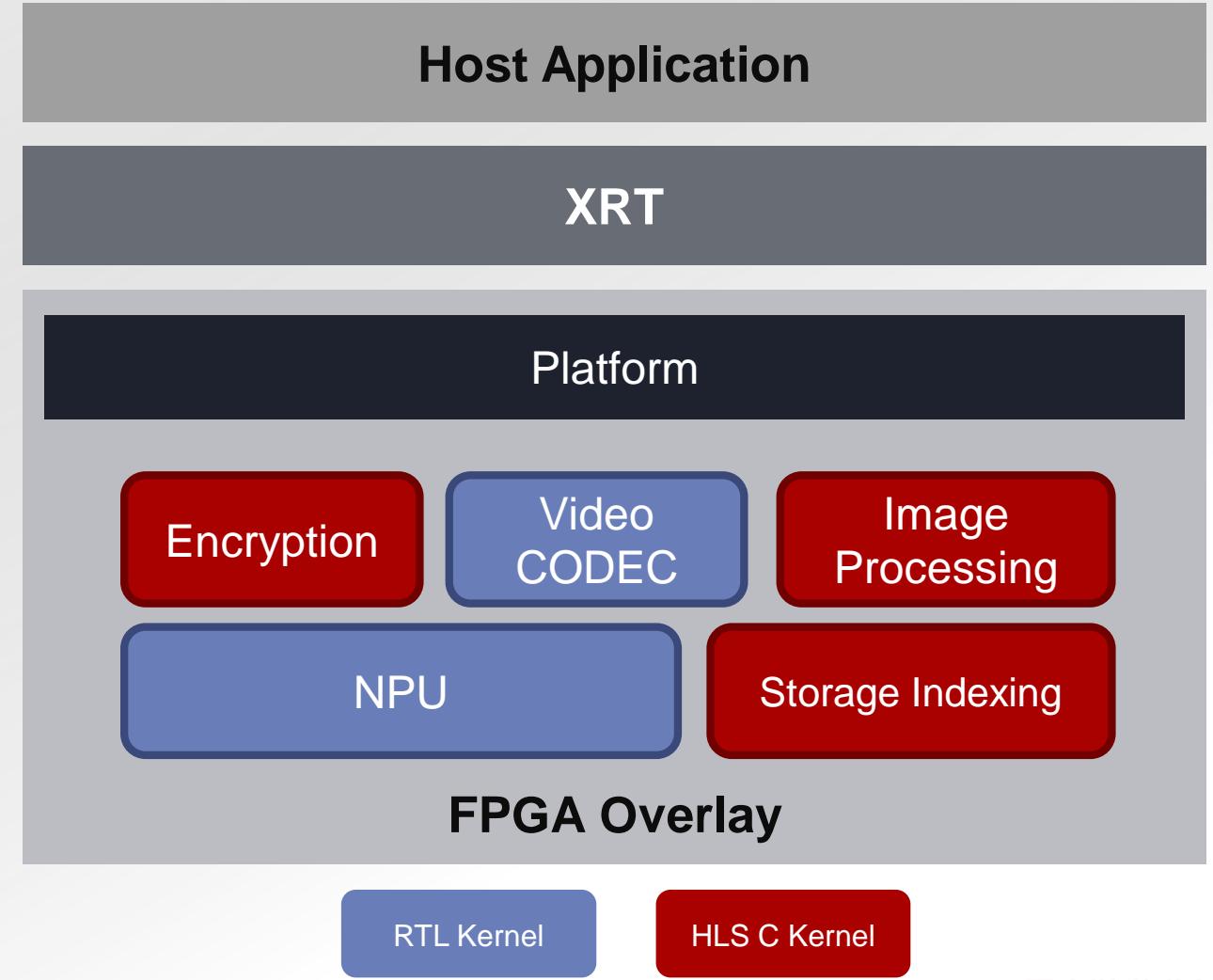
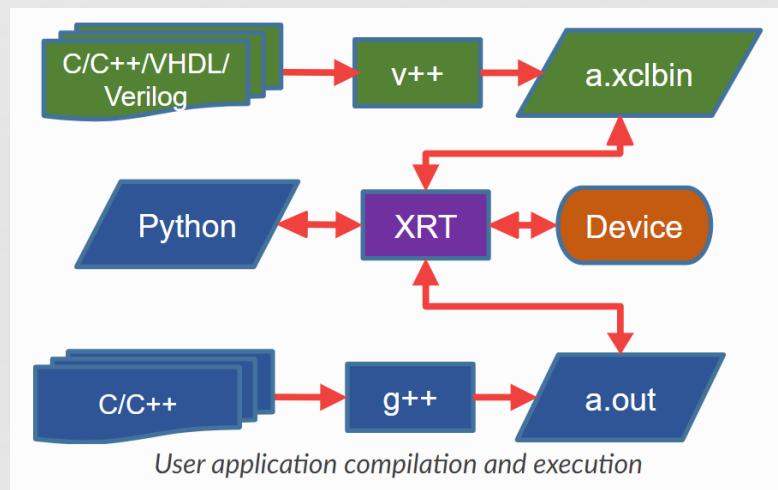
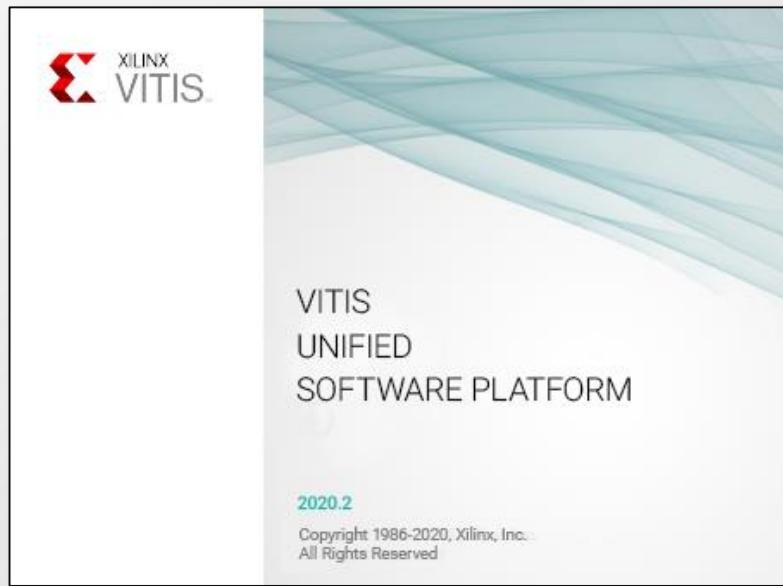
Image
CODEC

Processor

NPU

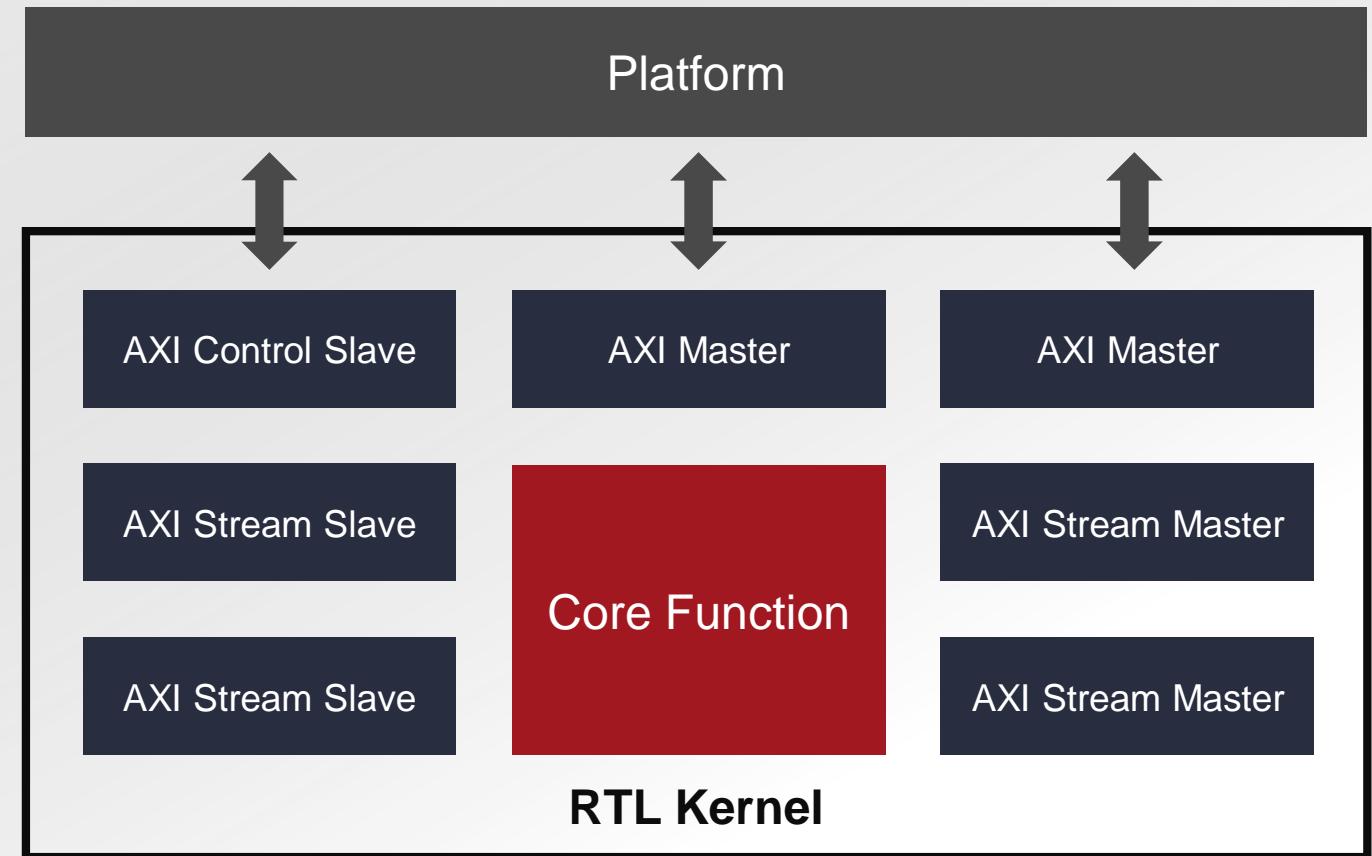
Others..

Fast Mixed Kernel Integration Flow with Vitis



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)

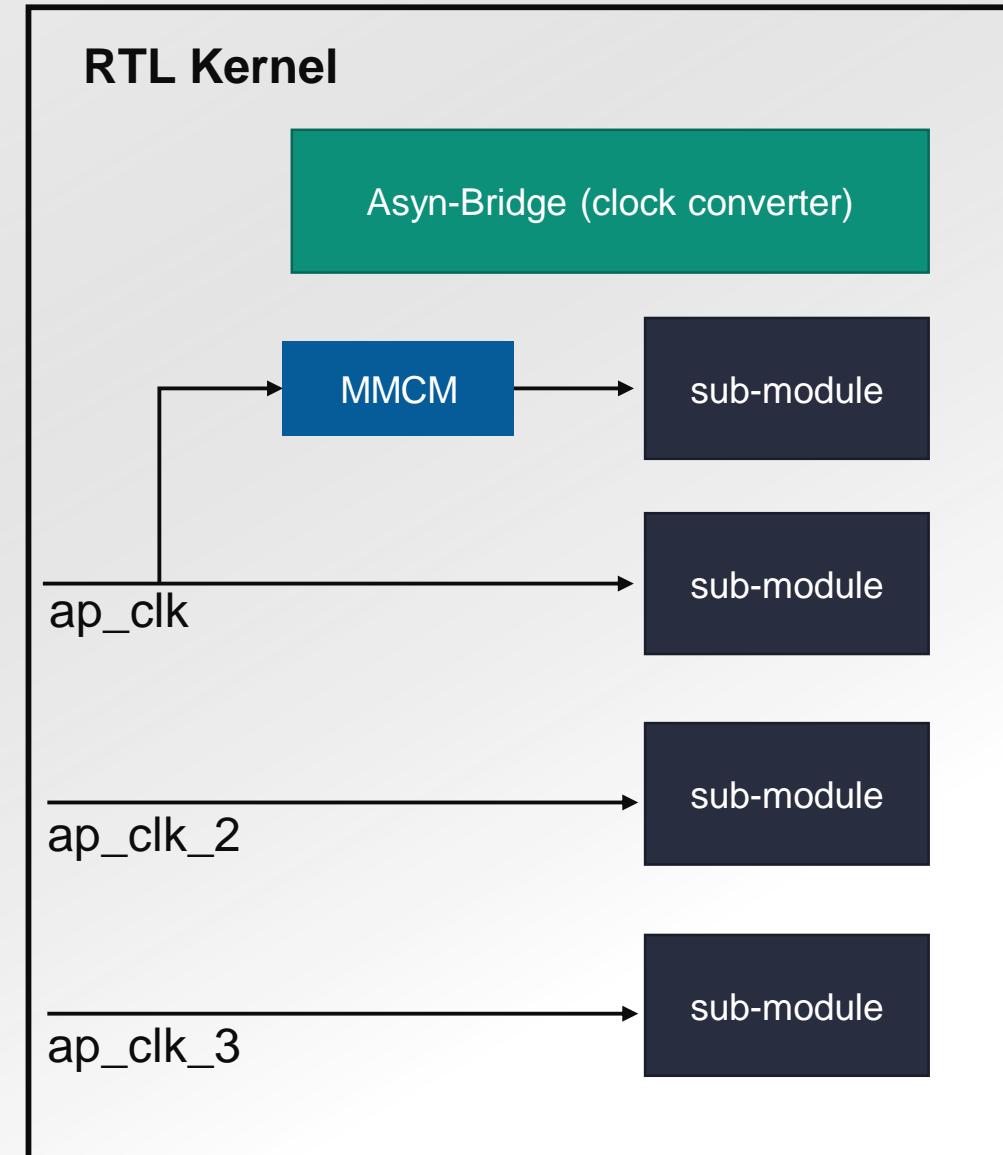


Clocking for RTL Kernel

- 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

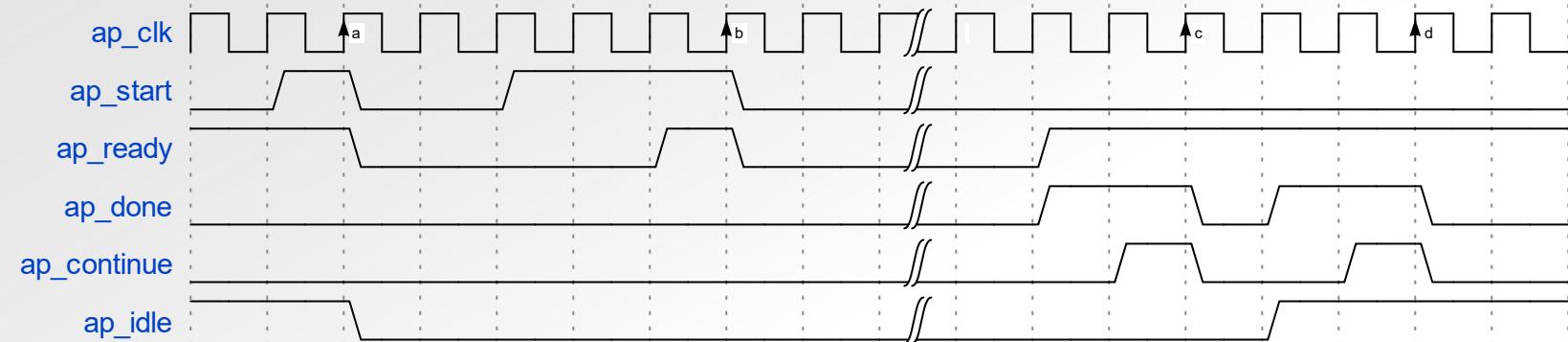
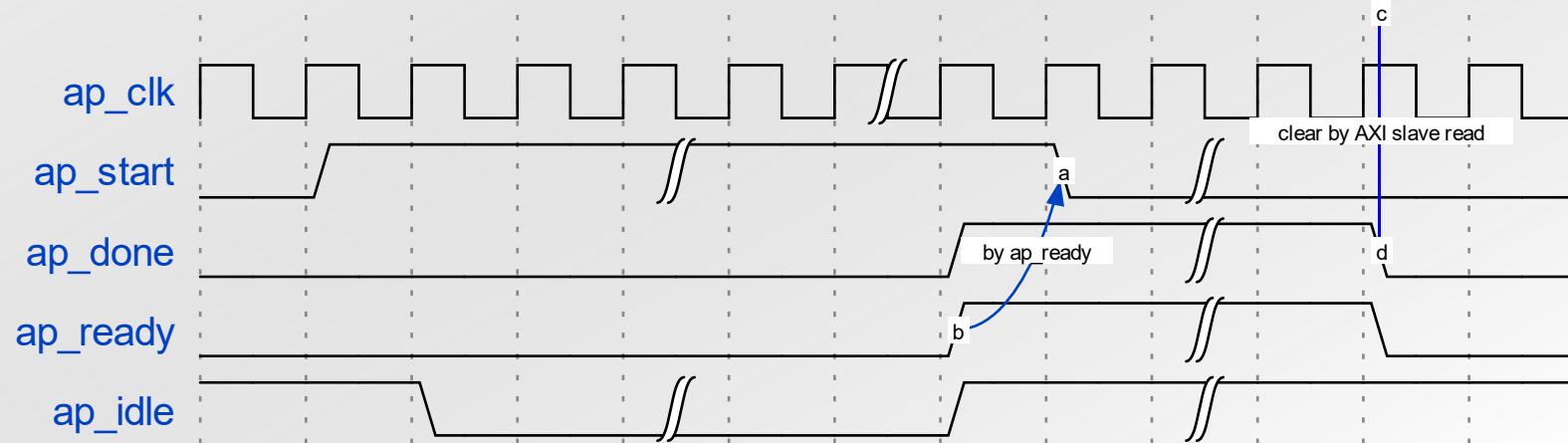
- Controlled via AXI slave
 - Hand-shake protocol
 - Sequential kernel
 - Input/output coupled

ap_ctrl_chain

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

AXI slave register signals:

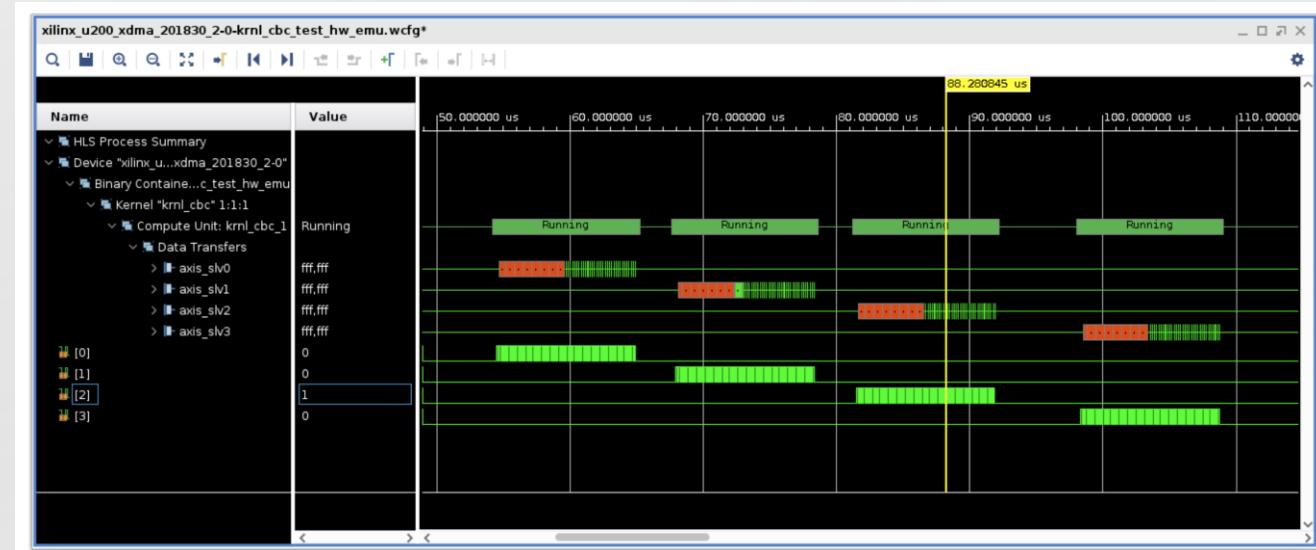
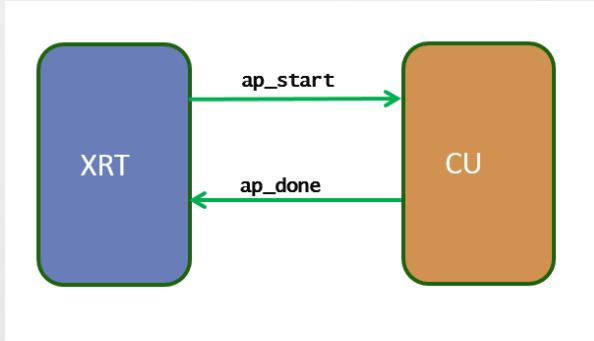
`ap_start, ap_done, ap_ready, ap_idle, ap_continue, ap_idles`



Execution Model: ap_ctrl_hs v.s. ap_ctrl_chain

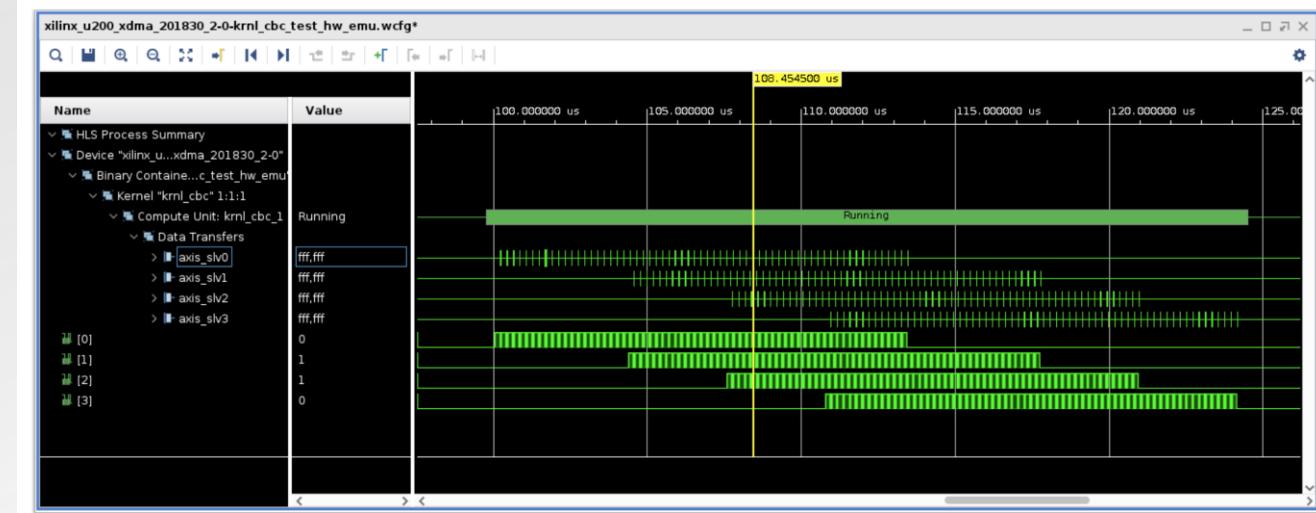
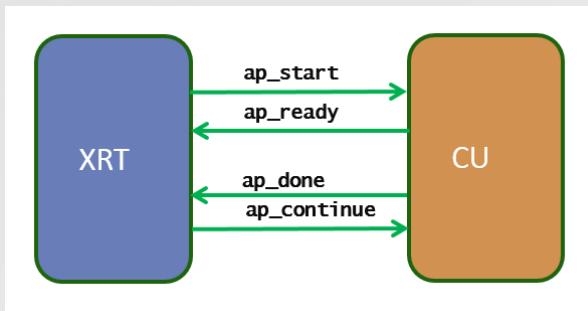
ap_ctrl_hs

- handshake signals: **ap_start, ap_done**



ap_ctrl_chain

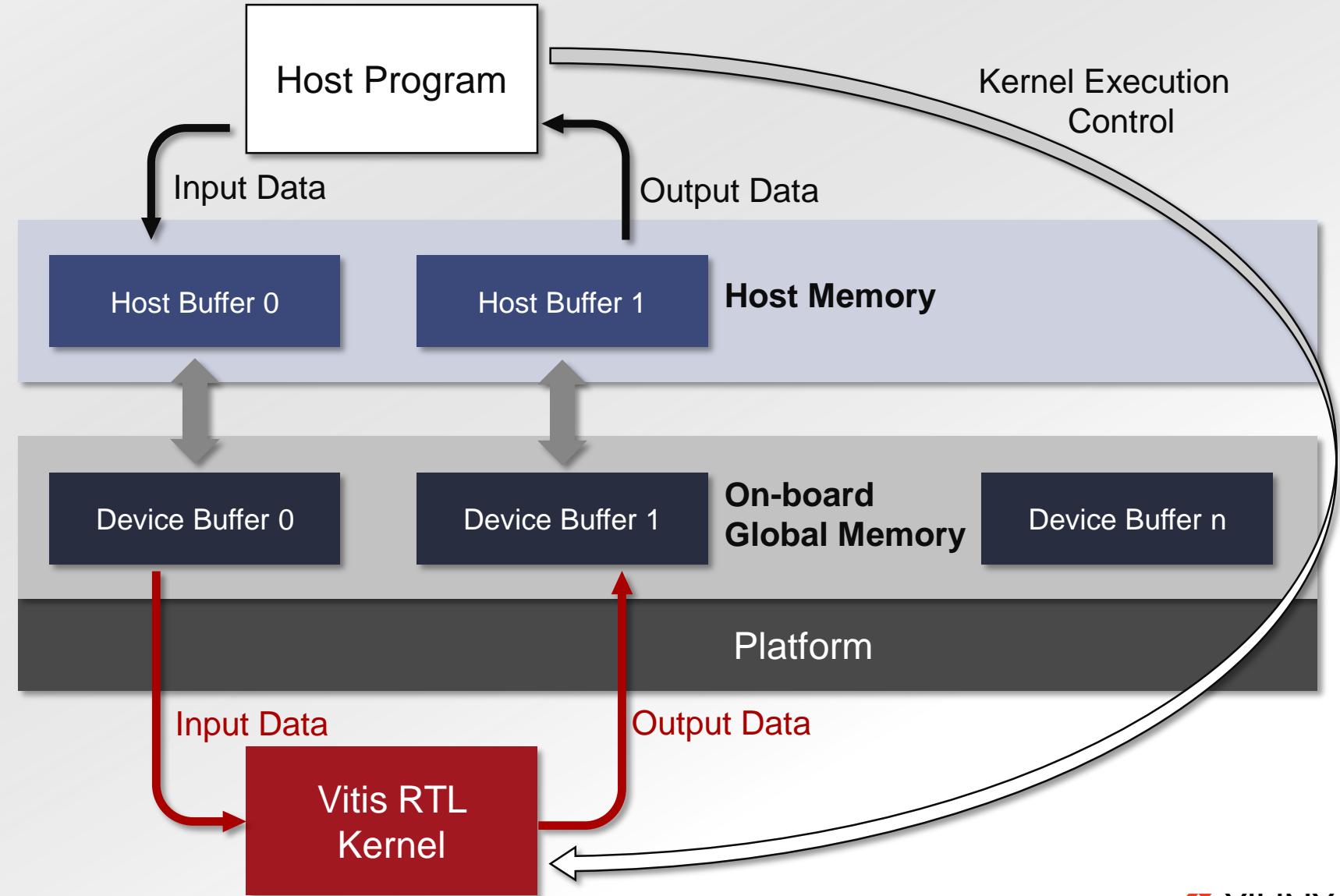
- input handshake signals:
ap_start, ap_ready
- output handshake signals:
ap_continue, ap_done



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

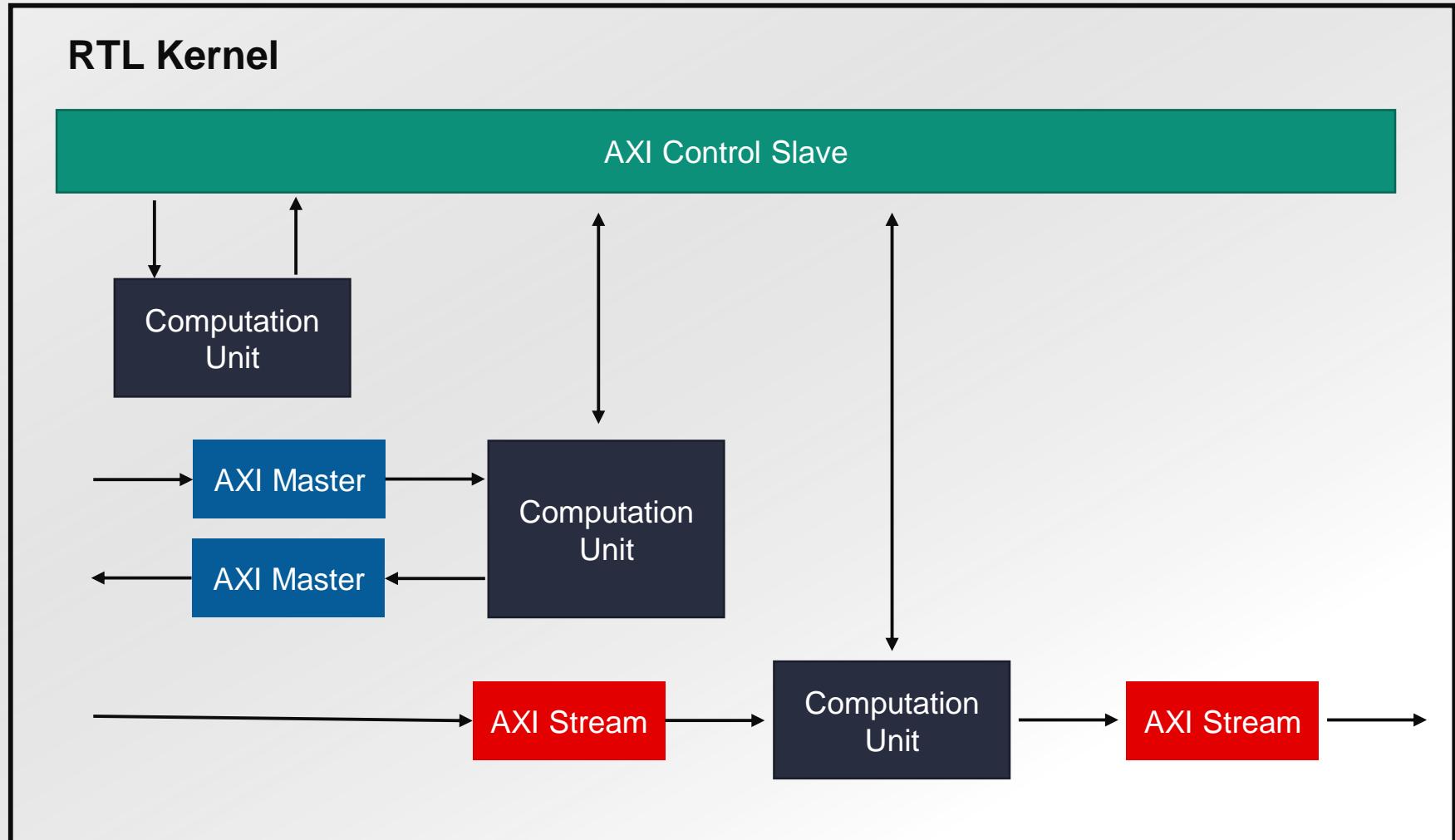
XRT Native API

- *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

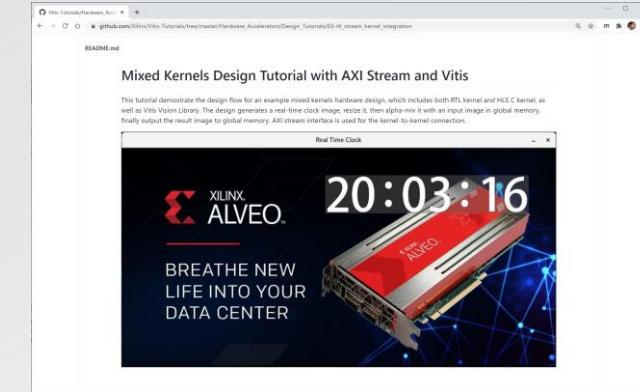


RTL Kernel Flow Example in Vitis Tutorials



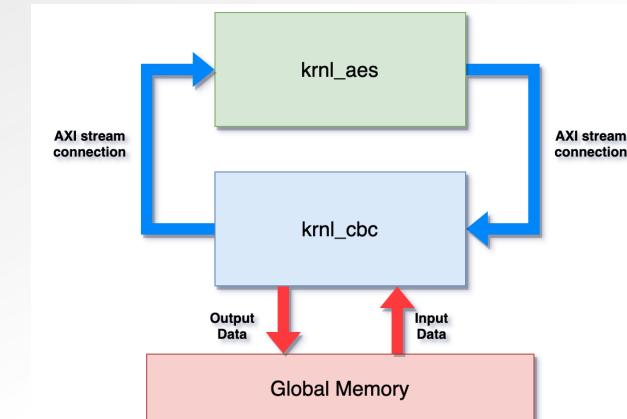
https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Hardware_Accelerators/Design_Tutorials/03-rtl_stream_kernel_integration

RTL Kernel Wizard Based Flow

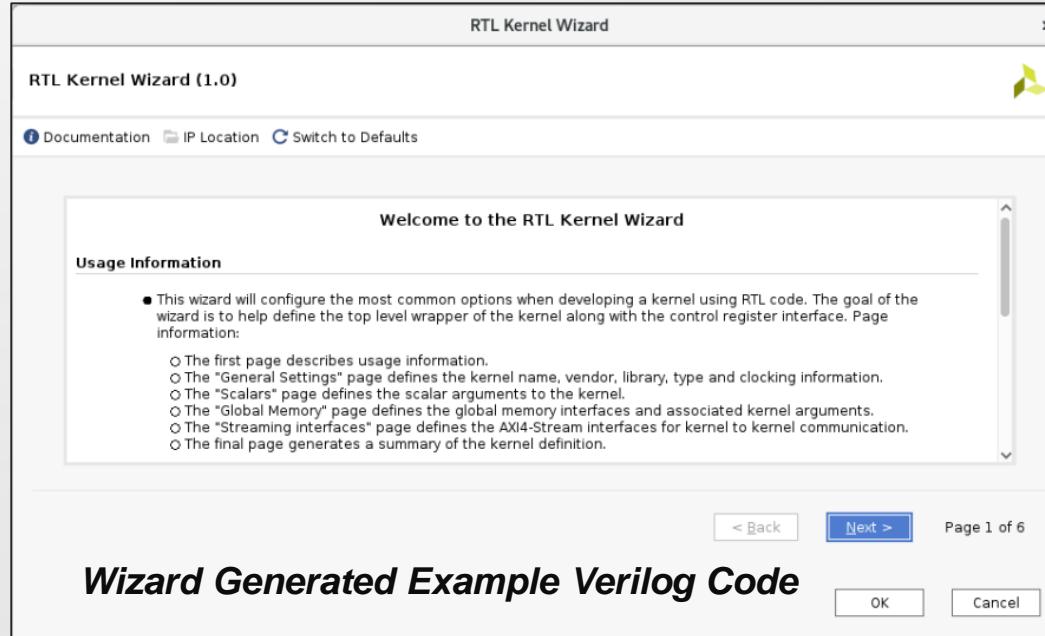


https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Hardware_Accelerators/Design_Tutorials/05-bottom_up_rtl_kernel

Bottom-up Flow with GUI or CLI



RTL Kernel Wizard Flow for Kernel Creation



Wizard Generated Example Verilog Code

```
Open Save File rtc_gen_core.v ~xilinx_work/rtl_stream_kernel_reference/rtc_gen/src
253 // =====
254 // real time clock accumulator
255 // =====
256 always @ (posedge clk or negedge reset_n) begin : cs_timer_control
257   if (!reset_n) begin
258     cs_timer <= 'h0;
259   end else if (cs_timer == (cs_count[21:0] - 1'b1)) begin
260     cs_timer <= 'h0;
261   end else begin
262     cs_timer <= cs_timer + 1'b1;
263   end
264 end // cs_timer_control
265
266 always @ (posedge clk or negedge reset_n) begin : time_set_control
267   if (!reset_n) begin
268     time_set_en_reg <= 1'b0;
269   end else begin
270     time_set_en_reg <= time_set_en[0];
271   end
272 end // time_set_control
273
274 always @ (posedge clk or negedge reset_n) begin : time_accumulator_control_seq
275   if (!reset_n) begin
276     cs_acc <= 'h0;
277     sec_acc <= 'h0;
278     min_acc <= 'h0;
279     hour_acc <= 'h0;
280   end else if (!time_set_en_reg && time_set_en[0]) begin
281     cs_acc <= time_set_centisecs;
282     sec_acc <= time_set_seconds;
283     min_acc <= time_set_minutes;
284     hour_acc <= time_set_hours;
285   end else begin
286     cs_acc <= next_cs_acc;
287     sec_acc <= next_sec_acc;
288     min_acc <= next_min_acc;
289     hour_acc <= next_hour_acc;
290   end
291 end // time_accumulator_control_seq
292
```

Customer Core Function Verilog Code



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

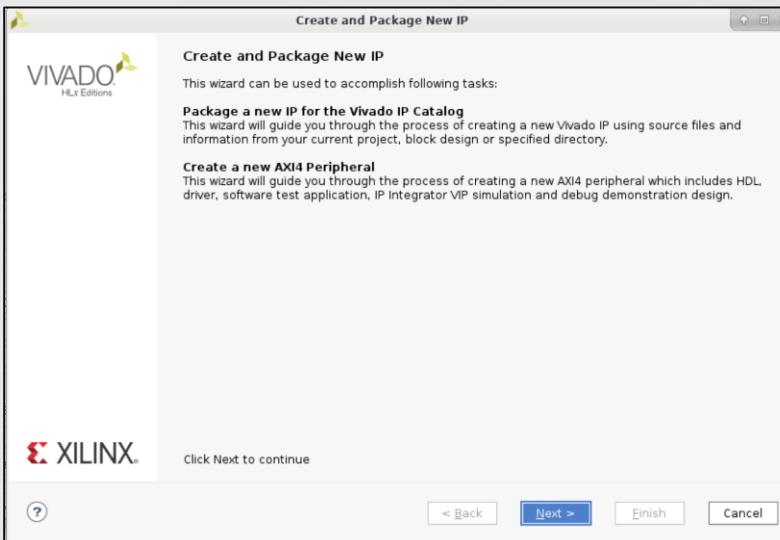
Vitis RTL Kernel

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

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 inferred 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*

The screenshot shows a text editor window titled 'pack.kernel.tcl' containing a Tcl script. The script defines various AXI interfaces and their associated clock and reset signals. It also sets up memory maps and registers for the kernel. Key parts of the script include:

```
47 ipx::infer_bus_interface ap_clk xilinx.com:signal:clock_rtl:1.0 [ipx::current_core]
48 ipx::infer_bus_interface ap_rst_n xilinx.com:signal:reset_rtl:1.0 [ipx::current_core]
49
50 # associate AXI/AXIS interface with clock
51 ipx::associate_bus_interfaces -busif s_axi_control -clock ap_clk [ipx::current_core]
52 ipx::associate_bus_interfaces -busif axi_rmst -clock ap_clk [ipx::current_core]
53 ipx::associate_bus_interfaces -busif axi_wmst -clock ap_clk [ipx::current_core]
54 ipx::associate_bus_interfaces -busif axis_mst0 -clock ap_clk [ipx::current_core]
55 ipx::associate_bus_interfaces -busif axis_mst1 -clock ap_clk [ipx::current_core]
56 ipx::associate_bus_interfaces -busif axis_mst2 -clock ap_clk [ipx::current_core]
57 ipx::associate_bus_interfaces -busif axis_mst3 -clock ap_clk [ipx::current_core]
58 ipx::associate_bus_interfaces -busif axis_slv0 -clock ap_clk [ipx::current_core]
59 ipx::associate_bus_interfaces -busif axis_slv1 -clock ap_clk [ipx::current_core]
60 ipx::associate_bus_interfaces -busif axis_slv2 -clock ap_clk [ipx::current_core]
61 ipx::associate_bus_interfaces -busif axis_slv3 -clock ap_clk [ipx::current_core]
62
63 # associate reset signal with clock
64 ipx::associate_bus_interfaces -clock ap_clk -reset ap_rst_n [ipx::current_core]
65
66
67 ##### Step 3: Set the definition of AXI control slave registers, including CTRL and user kernel arguments
68
69 # Add RTL kernel registers
70 ipx::add_register CTRL [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
71 ipx::add_register MODE [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
72 ipx::add_register WORDS_MODE [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
73 ipx::add_register IV_HD [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
74 ipx::add_register IV_W2 [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
75 ipx::add_register IV_W1 [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
76 ipx::add_register IV_W0 [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
77 ipx::add_register WORDS_NUM [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
78 ipx::add_register SRC_ADDR [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
79 ipx::add_register DEST_ADDR [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
80
81 # Set RTL kernel registers property
82 set_property description {Control Signals} [ipx::get_registers CTRL] -of_objects [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
83 set_property address_offset {0x000} [ipx::get_registers CTRL] -of_objects [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
84 set_property size {32} [ipx::get_registers CTRL] -of_objects [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axi_control -of_objects [ipx::current_core]]]
85
```

*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`



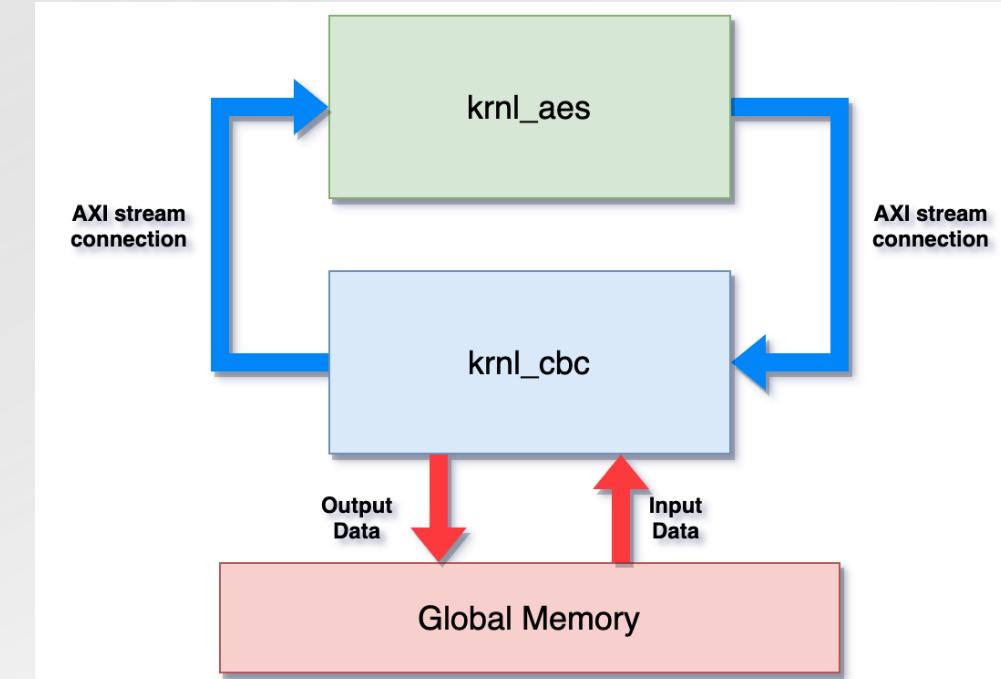
These `Tcl` commands can run in Vivado `Tcl` mode, batch mode, or `GUI` mode `Tcl` console.

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

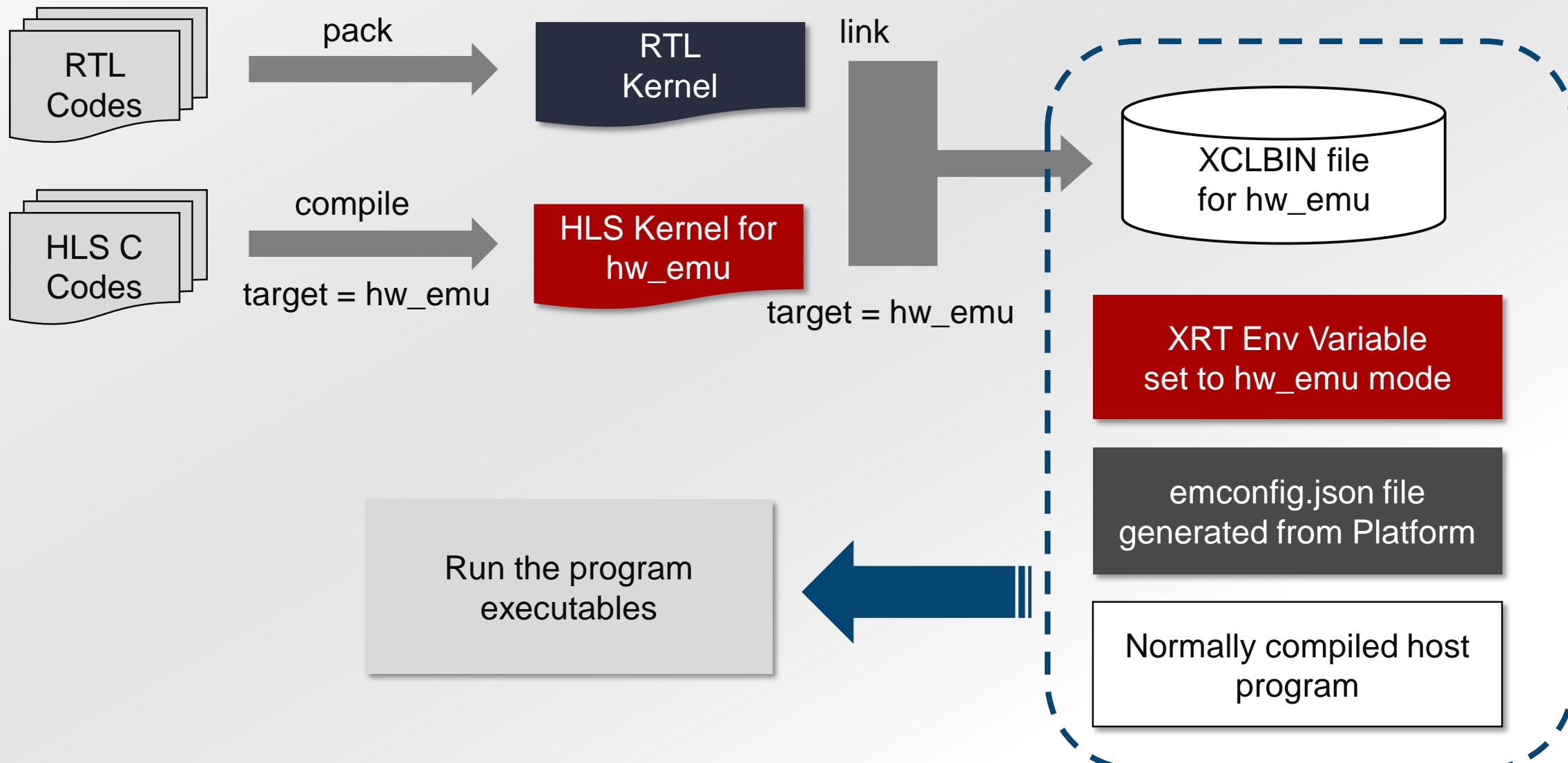
- ▶ Realize an Vitis RTL kernel for AES-256 and CBC mode acceleration
- ▶ Full CLI flow from RTL kernel creation to final system integration
- ▶ Demonstrate the implementation of *ap_ctrl_hs* and *ap_ctrl_chain* mode in RTL kernel



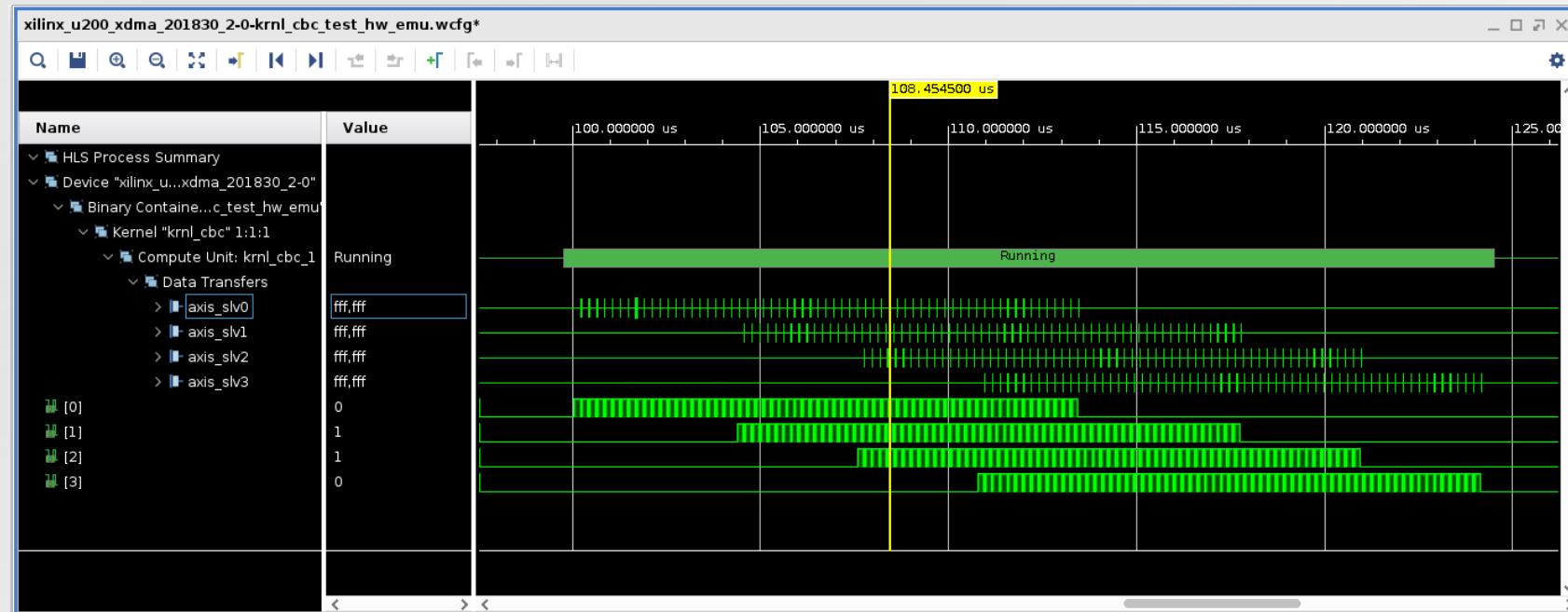
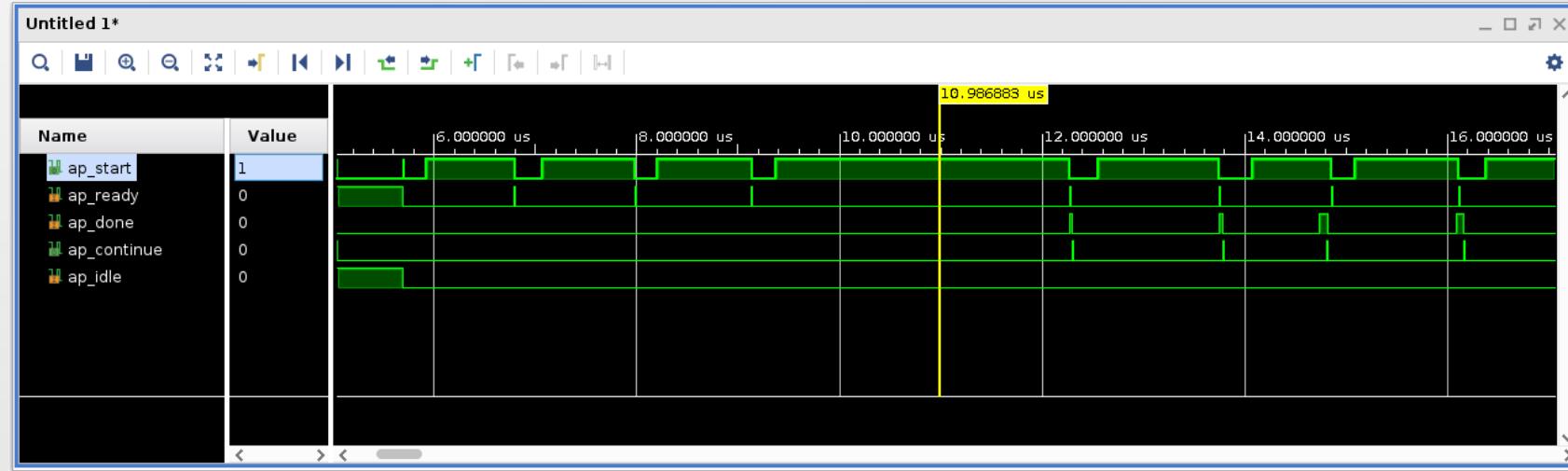
Use Hardware Emulation for Vitis RTL Kernel Designs

- 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



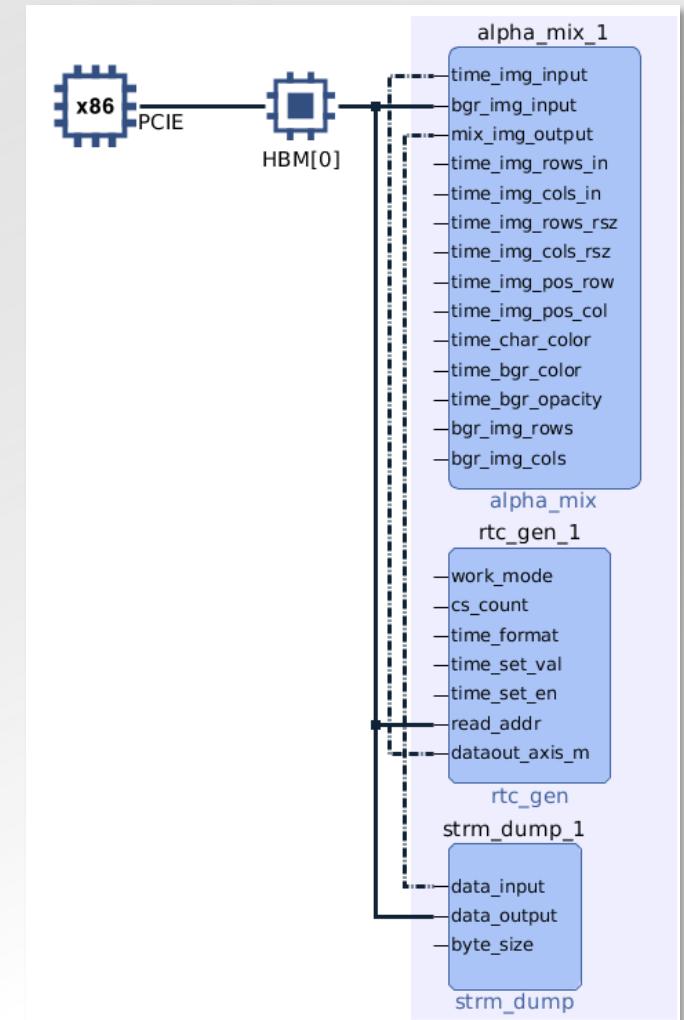
Use Vitis to Build Hardware Overlay (XCLBIN)



```
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

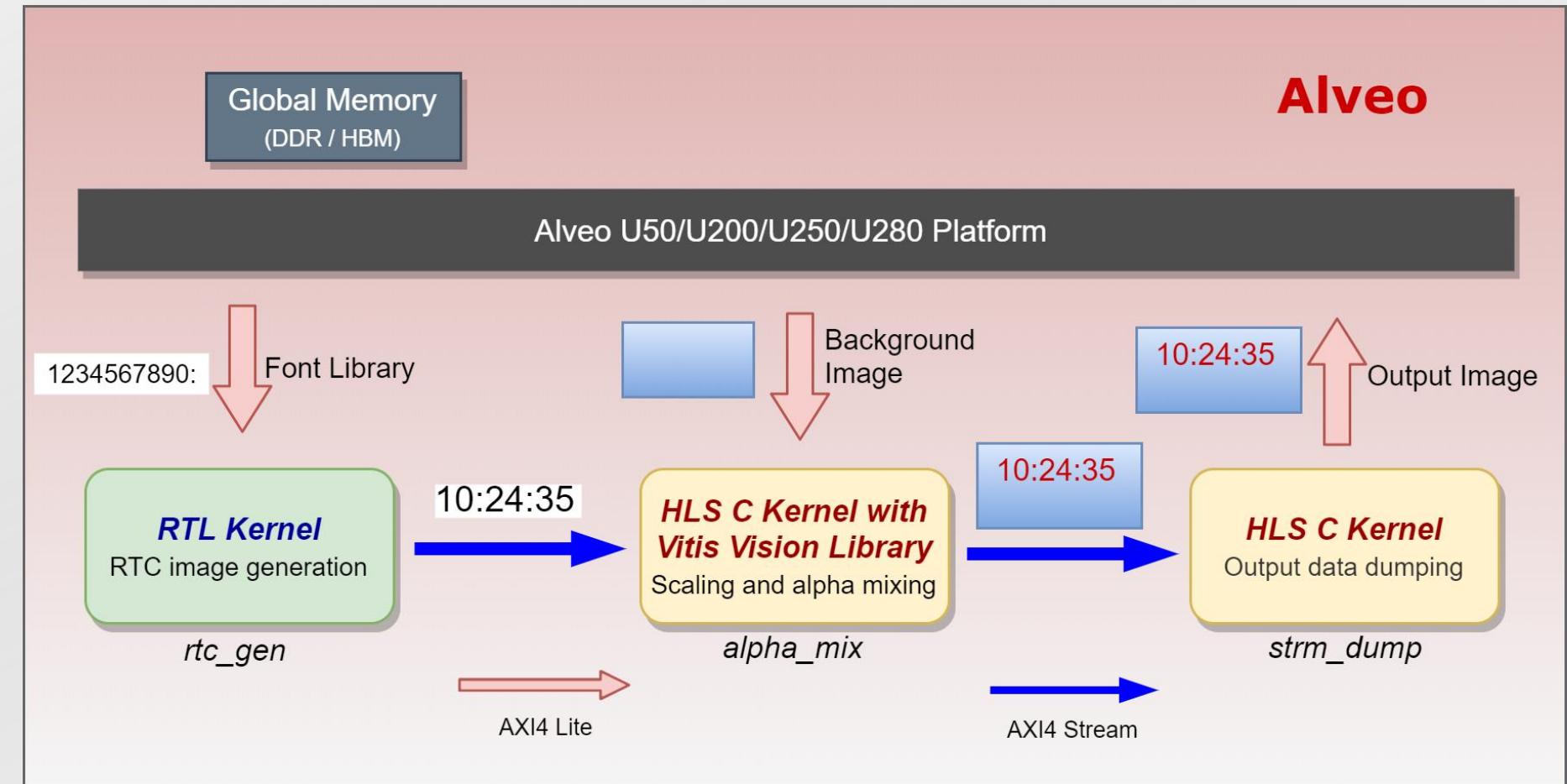
```
[connectivity]
stream_connect=rtc_gen_1.dataout_axis_m:alpha_mix_1.time_img_input
stream_connect=alpha_mix_1.mix_img_output:strm_dump_1.data_input
```



Mixed Kernel Design Example in Vitis Tutorials

Mixed HDL/HLS Kernel Integration Example

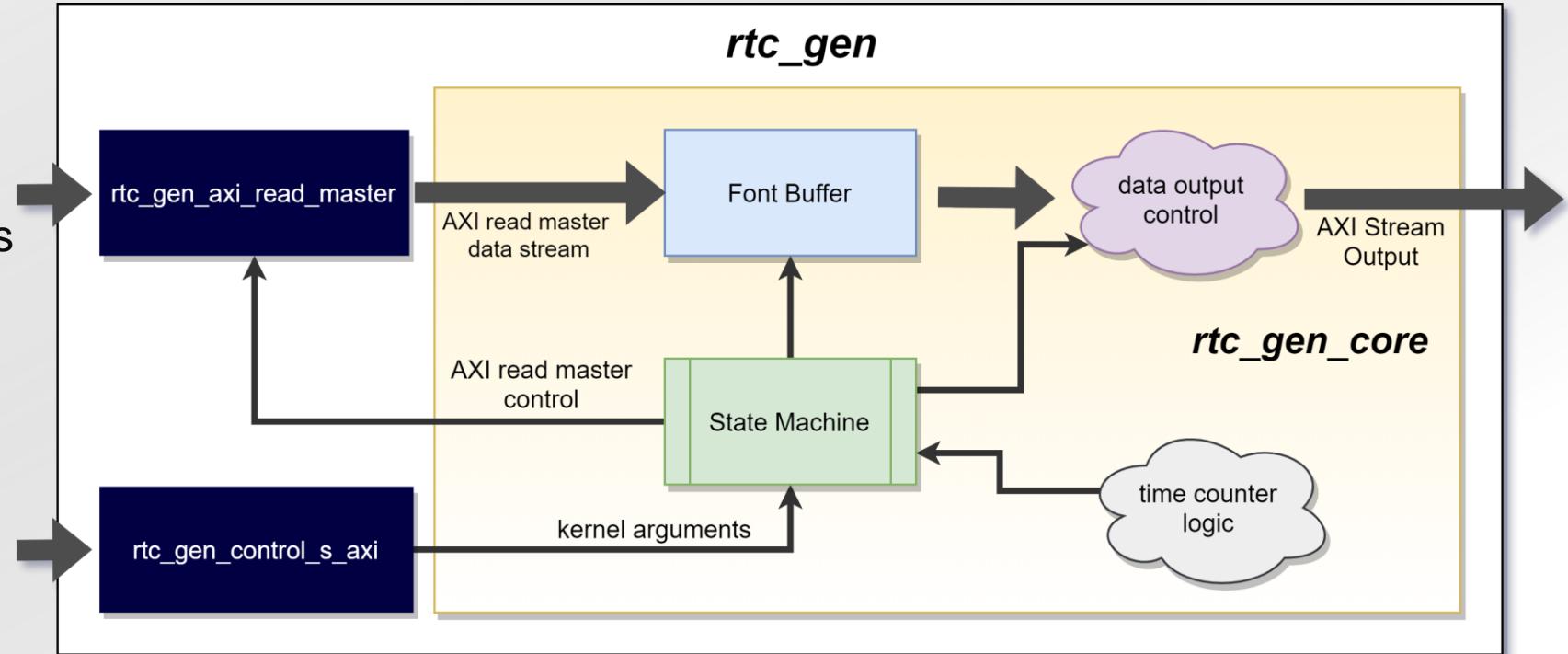
- rtc_gen**
 - ✓ Verilog Kernel
- alpha_mix**
 - ✓ HLS C Kernel with Vitis Vision Library
- strm_dump**
 - ✓ HLS C Kernel



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

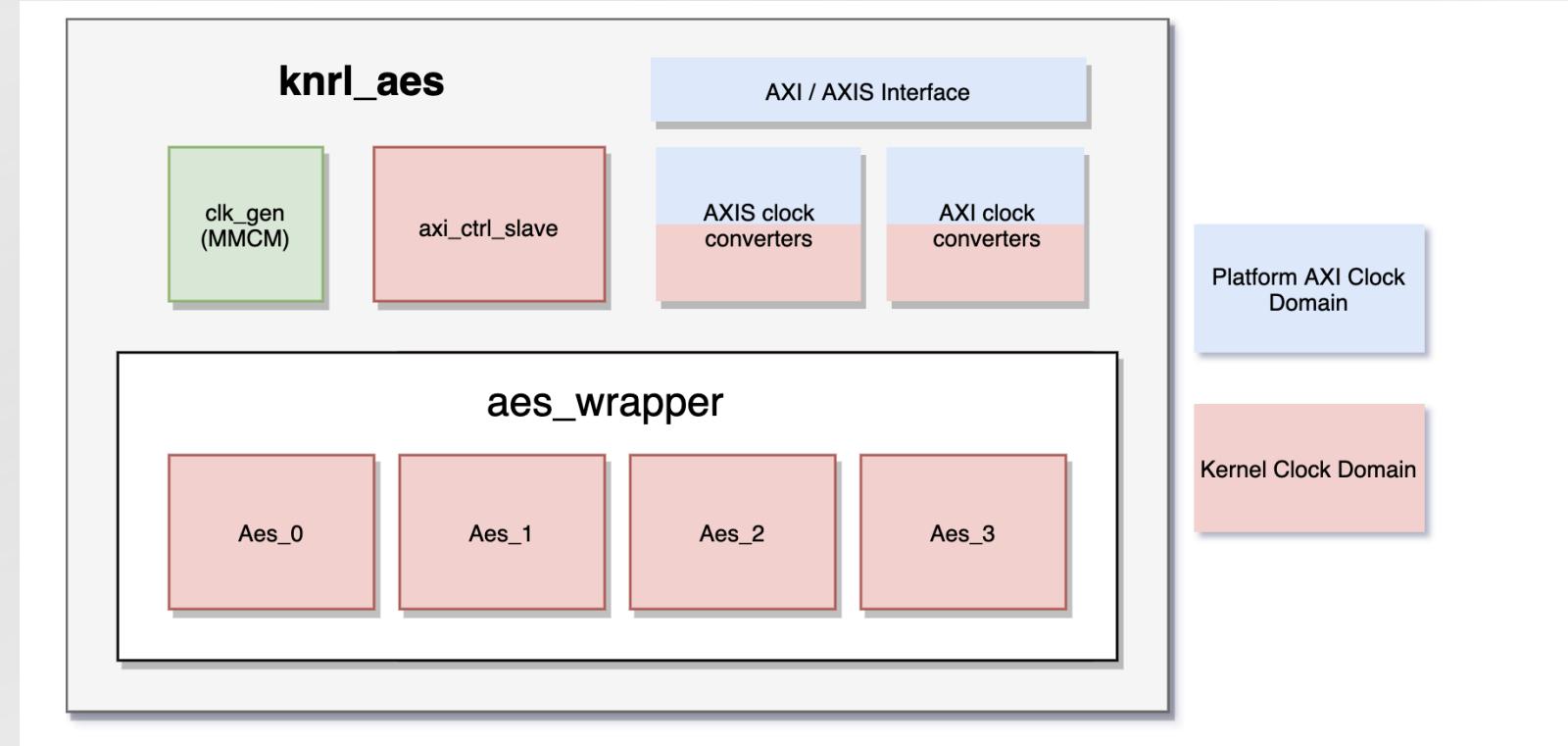


- Generated AXI master and AXI control slave modules are used.
- *rtc_gen_core* module is hand coded Verilog modules.
- Generated top level wrapper is modified to instantiate *rtc_gen_core* module.

Example RTL Kernel Design

RTL Kernel *knrl_aes*

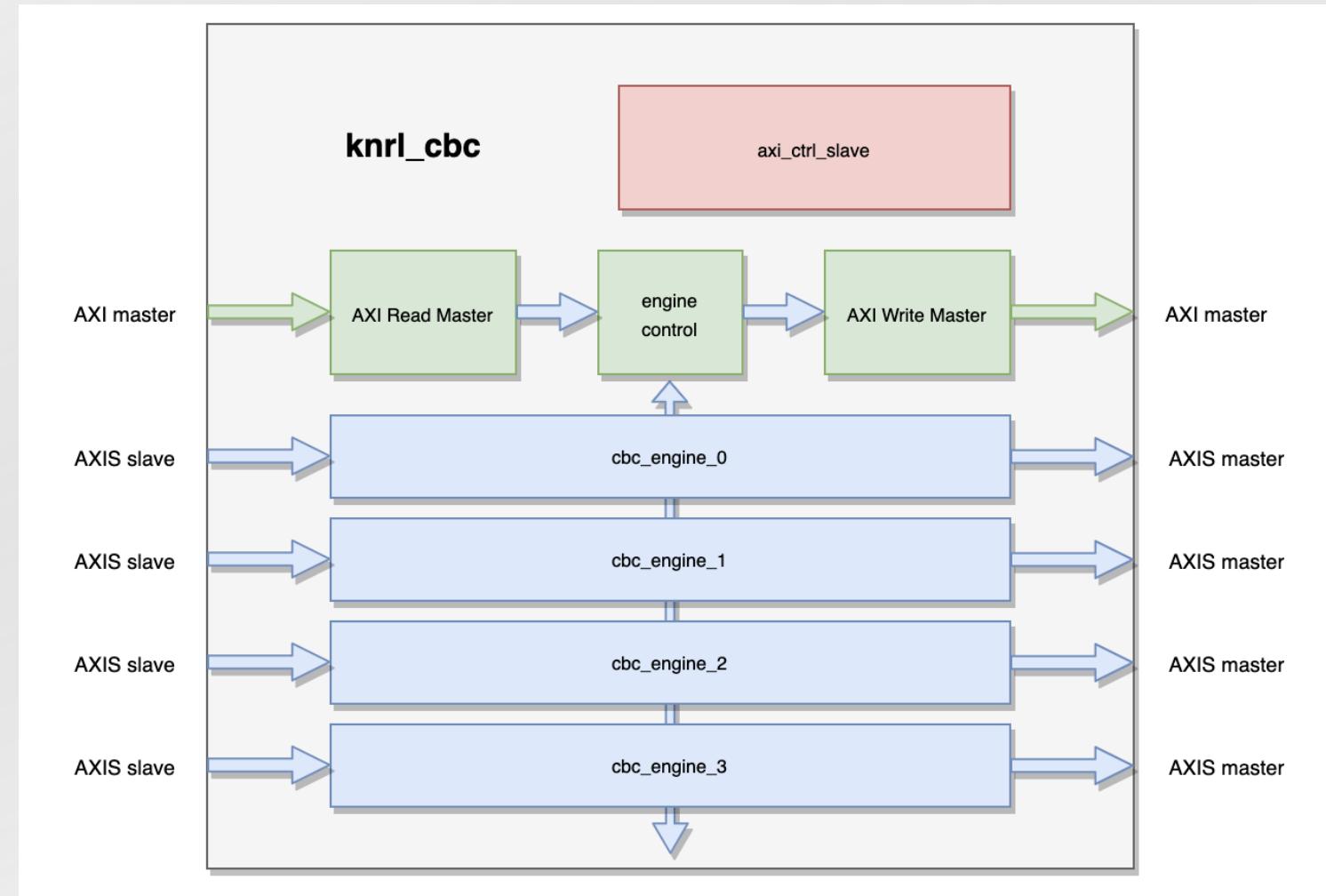
- One AXI control slave port
- Execution Model: *ap_ctrl_none*, *ap_ctrl_hs*
- One AXI stream slave port
- One AXI stream master port



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 generate specific clock



Thank you!