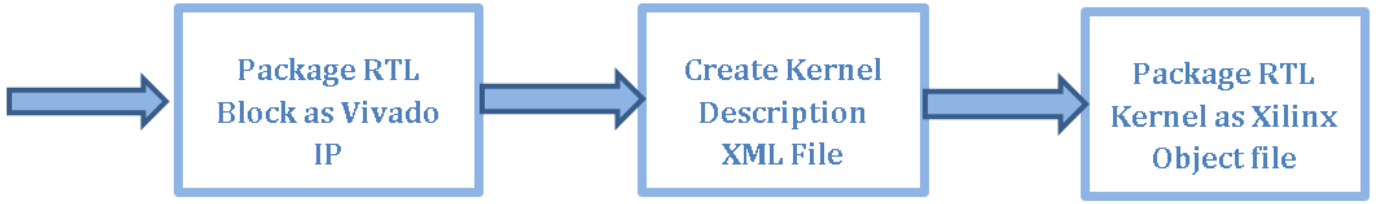
RTL Kernel Wizard

- Dfsdfsdf\asdf \asdkfja;ksdljf.ljsda flkjas asdkfjal;sdjkf alk aksdjf;lakjsd flkjs aladsj fksdj flkjas dlkfj alksdjf lksja lkdjf lkfja lds CATHJASHKJDHAS KDHa ajhdkajhs

Integration of RTL Kernels into the Vitis

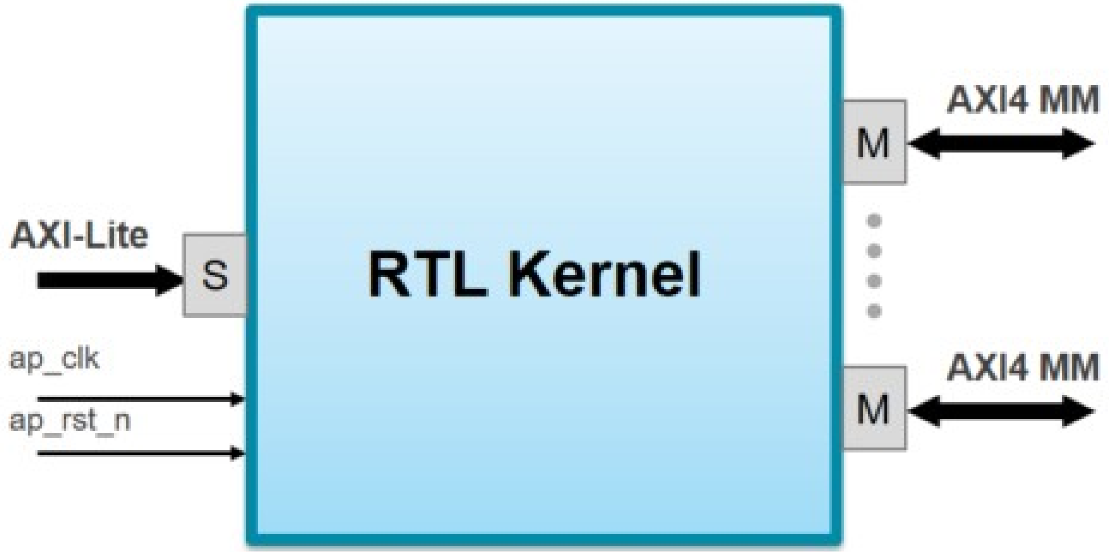
- RTL kernels can give higher performance & QoR than HLS kernels
  - Hardware design expertise and Vivado Design Suite knowledge required
- Vivado RTL Kernel Wizard can be used for packaging to Xilinx object file (.xo)
- IPI packaging define interface names and types
  - Kernel metadata defines the callable software “function” definition of the RTL kernel


Package RTL Block as Vivado IP
Create Kernel Description XML File
Package RTL Kernel as Xilinx Object file

- XO files can be organized into libraries for reuse, and a heterogeneous mix of kernel XOs can be used to construct complex applications

RTL Kernel: Hardware Requirements

- Clock and reset
- AXI-Lite slave interface
- AXI4 memory-mapped master interface(s)
- AXI streaming interface(s)



RTL Kernel Interface Requirements – Clock and Reset

- Primary clock and reset

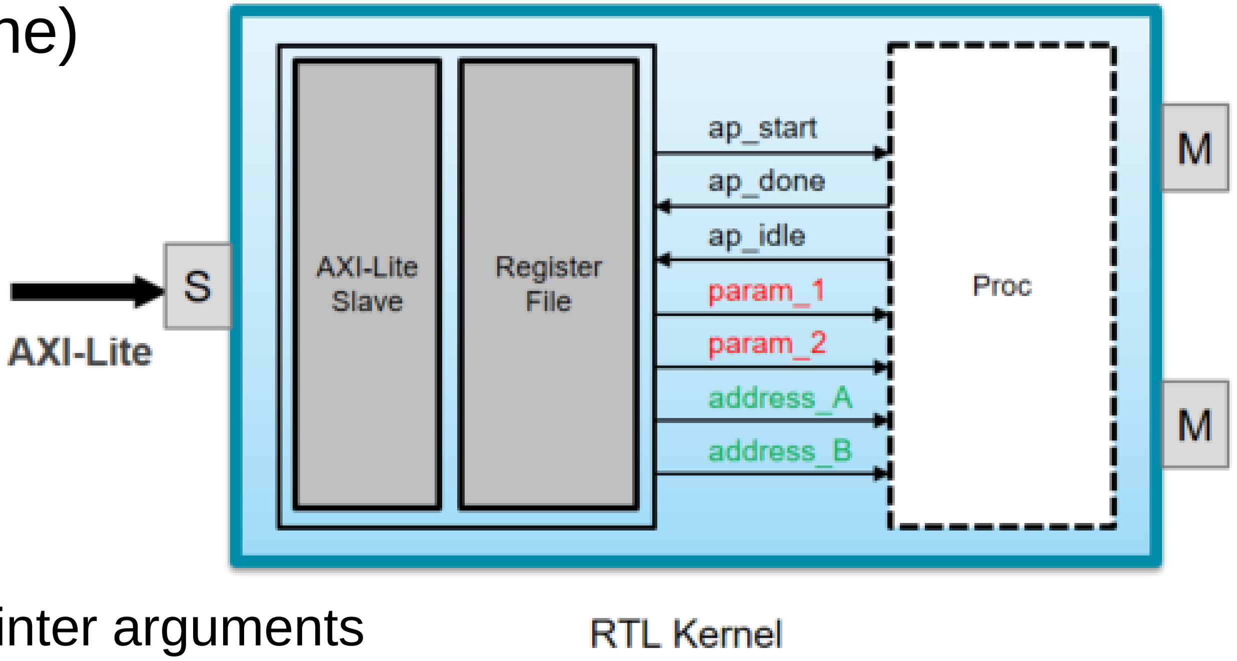
ap_clk	Rising edge	Clocks the AXI interfaces of the kernel.
ap_rst_n	Active low	Synchronous in the *ap_clk* domain.

- Optional secondary clock and reset

ap_clk_2	Rising edge	Independent from the primary clock. Useful if the kernel clock needs to run at a faster/slower rate than the AXI4 interface. When designing with multiple clocks, proper clock domain crossing techniques must be used to ensure data integrity across all clock frequency scenarios.
ap_rst_n_2	Active low	Synchronous in the *ap_clk_2* domain

RTL Kernel Interface Requirements – AXI-Lite Slave

- RTL kernel must have one (and only one) AXI-Lite slave interface
- Kernel control interface
- Use by the host application to
  - Start kernel execution
  - Monitor status
  - Write kernel scalar arguments
  - Write base address in global memory of pointer arguments



AXI-Lite Interface – Control and Status Register

- The kernel control and status register is at 0x00 in the register file

Bit	Name	Description
0	ap_start	The kernel should start processing data when this bit is set.
1	ap_done	The kernel should start asserting this signal when the processing is done. This bit cleared on read.
2	ap_idle	The kernel should assert this signal when it is not processing any data. The transition from low to high should occur synchronously with assertion of done signal.
31:3	Reserved	Reserved.

XILINX.

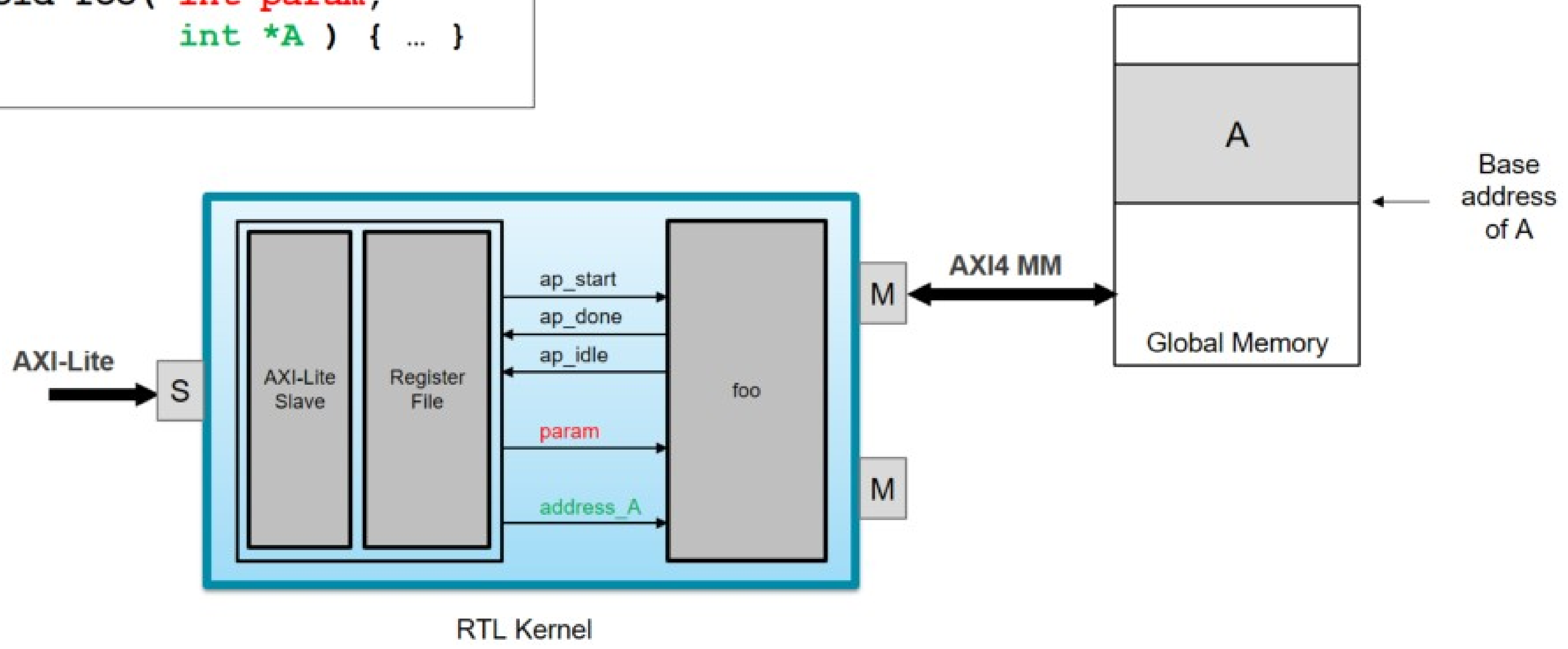
RTL Interface Requirements – AXI4 MM Master

- 1 to 16 AXI4 memory-mapped master interfaces to read and write data from global memory
- Base address of data in global memory is provided by the host application through the AXI-Lite slave interface
- All AXI4 master interfaces must have 64-bit addresses
- Global memory management using the AXI4 master ports should be based on the performance and bandwidth requirements of the design
  - One master interface per required DDR channel is recommended

AXI4 MM Master – Data and Base Address Example

Host code

```
void foo( int param,
          int *A ) { … }
```

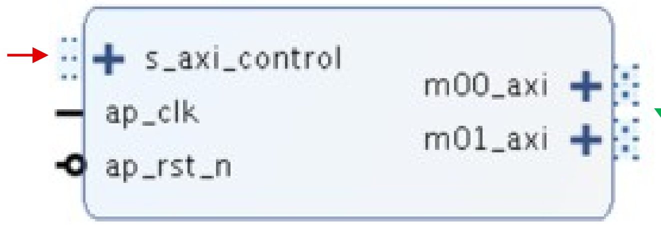


RTL Kernel: Software Requirements (1)

```
void func(    int length,
              int *a,
              int *b,
              int *output);
```

}

AXI Lite:
Scalar inputs



AXI
Master/Stream
Port(s)

- Vitis associates specific C function argument types (host code) with specific hardware port types (RTL kernel)
- AXI-Lite slave port for *scalar* arguments
  - Control data can share this interface
- AXI memory-mapped master or AXI Stream port can be used for ***pointer/array*** arguments

RTL Interface Requirements – AXI-Stream Interfaces

- 1 to 32 AXI-Stream interfaces are available

- Supports the direct streaming of data from host to kernel and kernel to host without the need to migrate data through global memory as an intermediate step

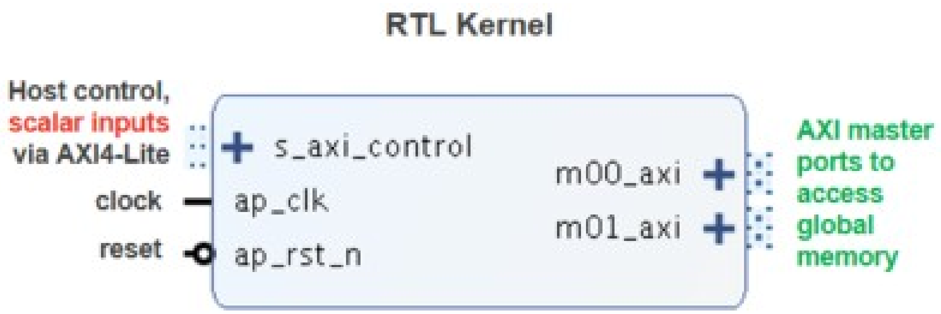
- Streaming interface could be a master or slave interface
  - Master is write only; only host can retrieve data from it
  - Slave is read only; only host can send data to it

- Interface width is limited to 1 to 64 bytes in powers of 2

- Uses the TDATA/TKEEP/TLAST signals of the AXI4-Stream protocol

RTL Kernel: Software Requirements (2)

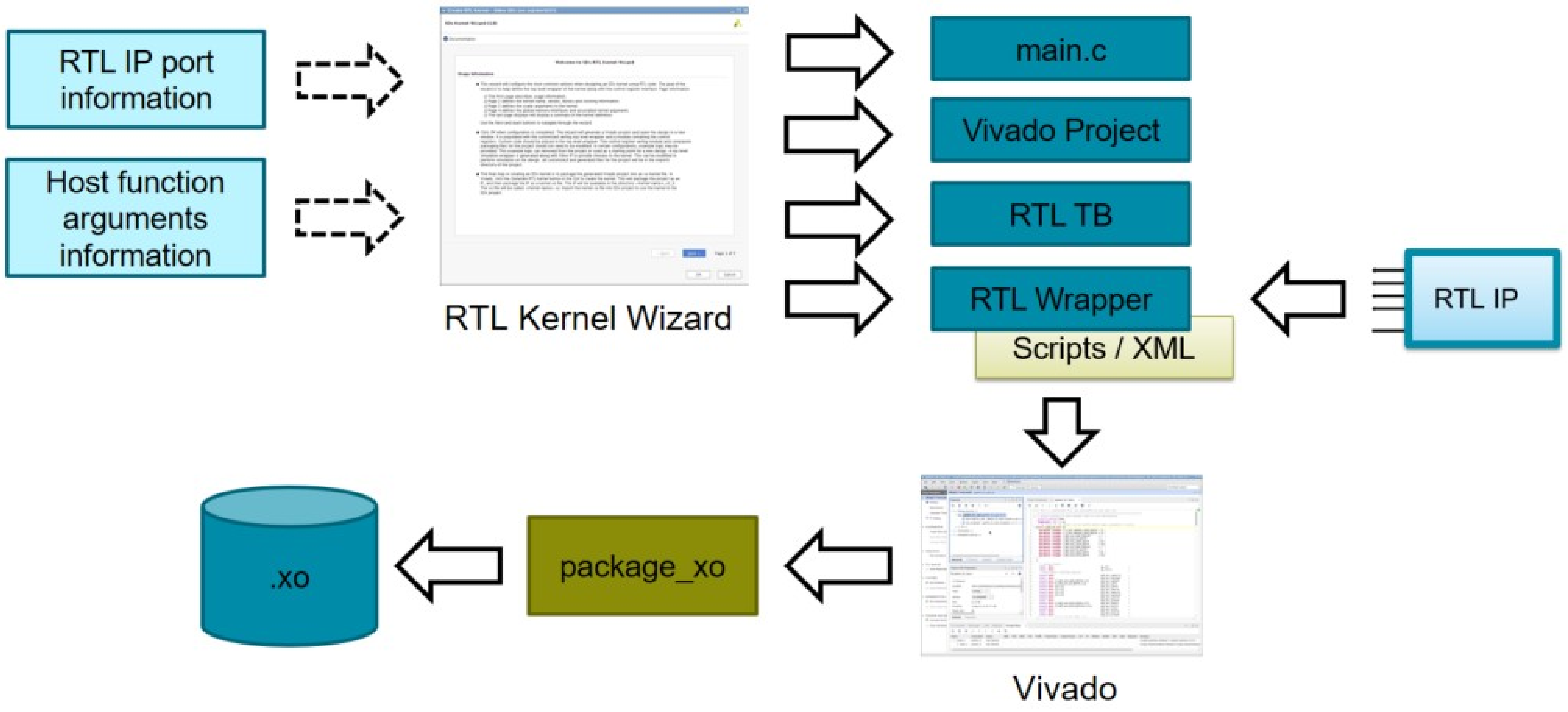
- ***Scalar*** arguments
  - Inputs only
  - Written to the kernel via AXI4-Lite interface
- ***Pointer*** arguments
  - Inputs or outputs
  - Kernel is responsible for accessing the data through the AXI4 master interface
  - Base address of the memory is passed via the AXI4-Lite interface
  - Kernel is started and polled for completion status via AXI4-Lite


RTL Kernel
Host control, scalar inputs via AXI4-Lite
clock
reset
s_axi_control
ap_clk
ap_rst_n
m00_axi
m01_axi
AXI master ports to access global memory

RTL Kernel Wizard Flow Overview

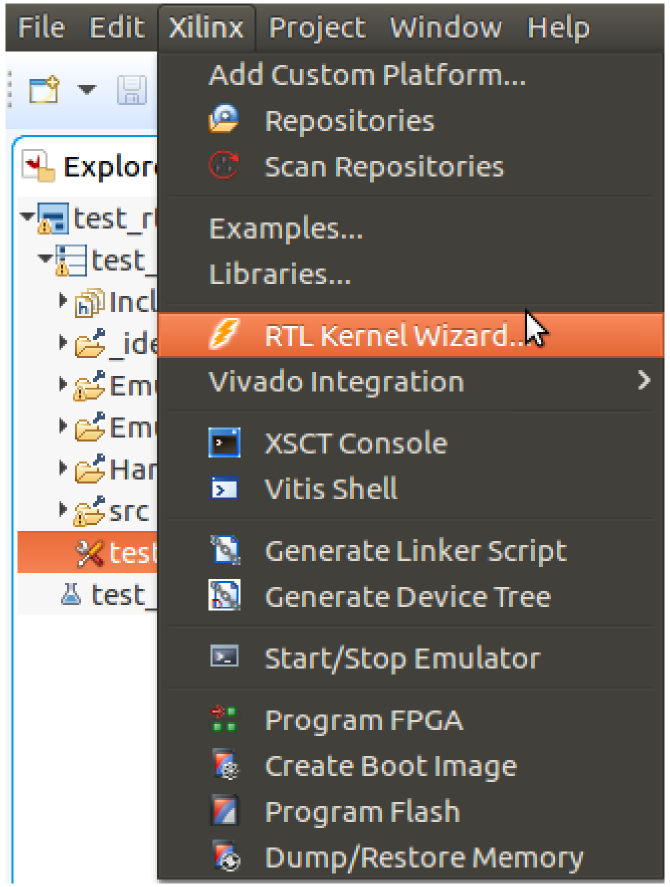
- Provides an easy means of packaging an RTL IP into a Vitis platform kernel
- Creates a top-level RTL kernel wrapper that contains
  - AXI-Lite interface module, including control logic and register file
  - Example kernel IP module to be replaced with the actual RTL IP design
  - One or more AXI-master interfaces
- Creates a Vivado Design Suite project for the RTL kernel wrapper and generated files
- Also provides a simple test infrastructure for the wrapper IP
  - RTL test bench for the RTL kernel wrapper only
  - Sample host code to exercise the packaged RTL kernel

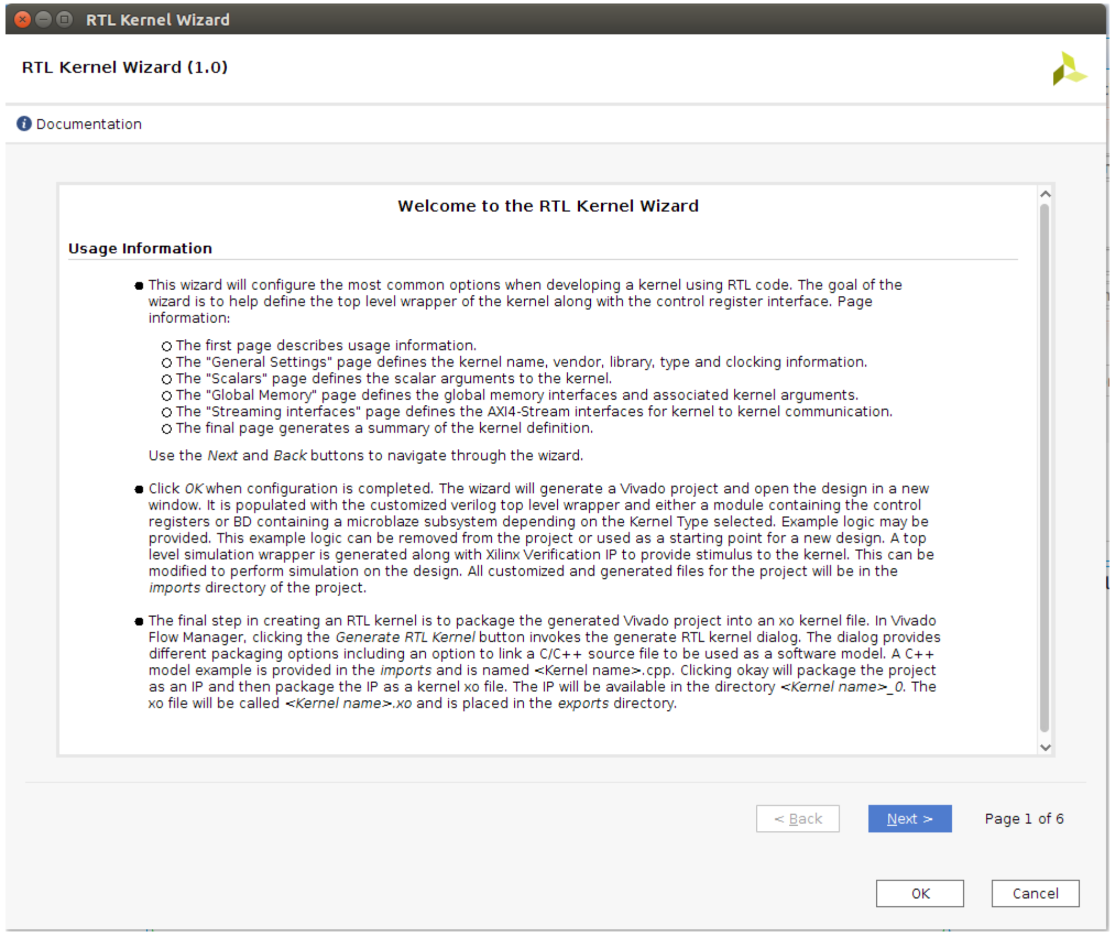
RTL Kernel Wizard Flow Overview



Invoking the RTL Kernel Wizard

- RTL Kernel Wizard is invoked from the Vitis IDE GUI





Welcome to the RTL Kernel Wizard

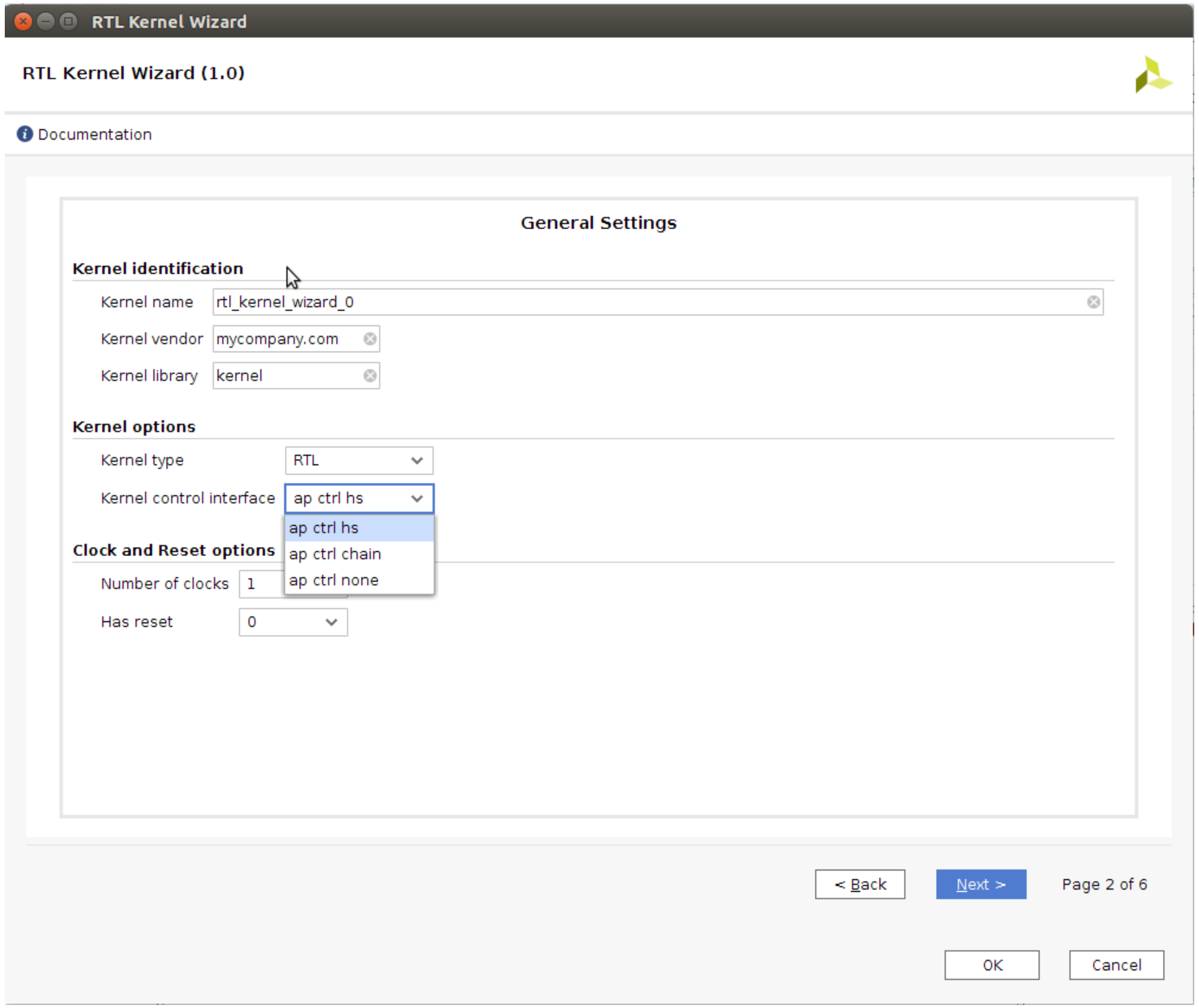
Usage Information

- This wizard will configure the most common options when developing a kernel using RTL code. The goal of the wizard is to help define the top level wrapper of the kernel along with the control register interface. Page information:
  - The first page describes usage information.
  - The "General Settings" page defines the kernel name, vendor, library, type and clocking information.
  - The "Scalars" page defines the scalar arguments to the kernel.
  - The "Global Memory" page defines the global memory interfaces and associated kernel arguments.
  - The "Streaming interfaces" page defines the AXI4-Stream interfaces for kernel to kernel communication.
  - The final page generates a summary of the kernel definition.

  Use the *Next* and *Back* buttons to navigate through the wizard.
- Click *OK* when configuration is completed. The wizard will generate a Vivado project and open the design in a new window. It is populated with the customized verilog top level wrapper and either a module containing the control registers or BD containing a microblaze subsystem depending on the Kernel Type selected. Example logic may be provided. This example logic can be removed from the project or used as a starting point for a new design. A top level simulation wrapper is generated along with Xilinx Verification IP to provide stimulus to the kernel. This can be modified to perform simulation on the design. All customized and generated files for the project will be in the *imports* directory of the project.
- The final step in creating an RTL kernel is to package the generated Vivado project into an xo kernel file. In Vivado Flow Manager, clicking the *Generate RTL Kernel* button invokes the generate RTL kernel dialog. The dialog provides different packaging options including an option to link a C/C++ source file to be used as a software model. A C++ model example is provided in the *imports* and is named <Kernel name>.cpp. Clicking okay will package the project as an IP and then package the IP as a kernel xo file. The IP will be available in the directory *<Kernel name>_0*. The xo file will be called *<Kernel name>.xo* and is placed in the *exports* directory.

RTL Kernel Wizard – General Settings

- Kernel name is used with function `clCreateKernel` in the host code to reference the kernel
- Kernel control interface determines the RTL kernel interface generated
- Number of clocks determines if the RTL IP includes a secondary clock
- ‘Has reset’ determines if global reset is created during kernel generation


RTL Kernel Wizard

RTL Kernel Wizard (1.0)

Documentation

General Settings

Kernel identification

Kernel name	rtl_kernel_wizard_0
Kernel vendor	mycompany.com
Kernel library	kernel

Kernel options

Kernel type	RTL
Kernel control interface	ap ctrl hs

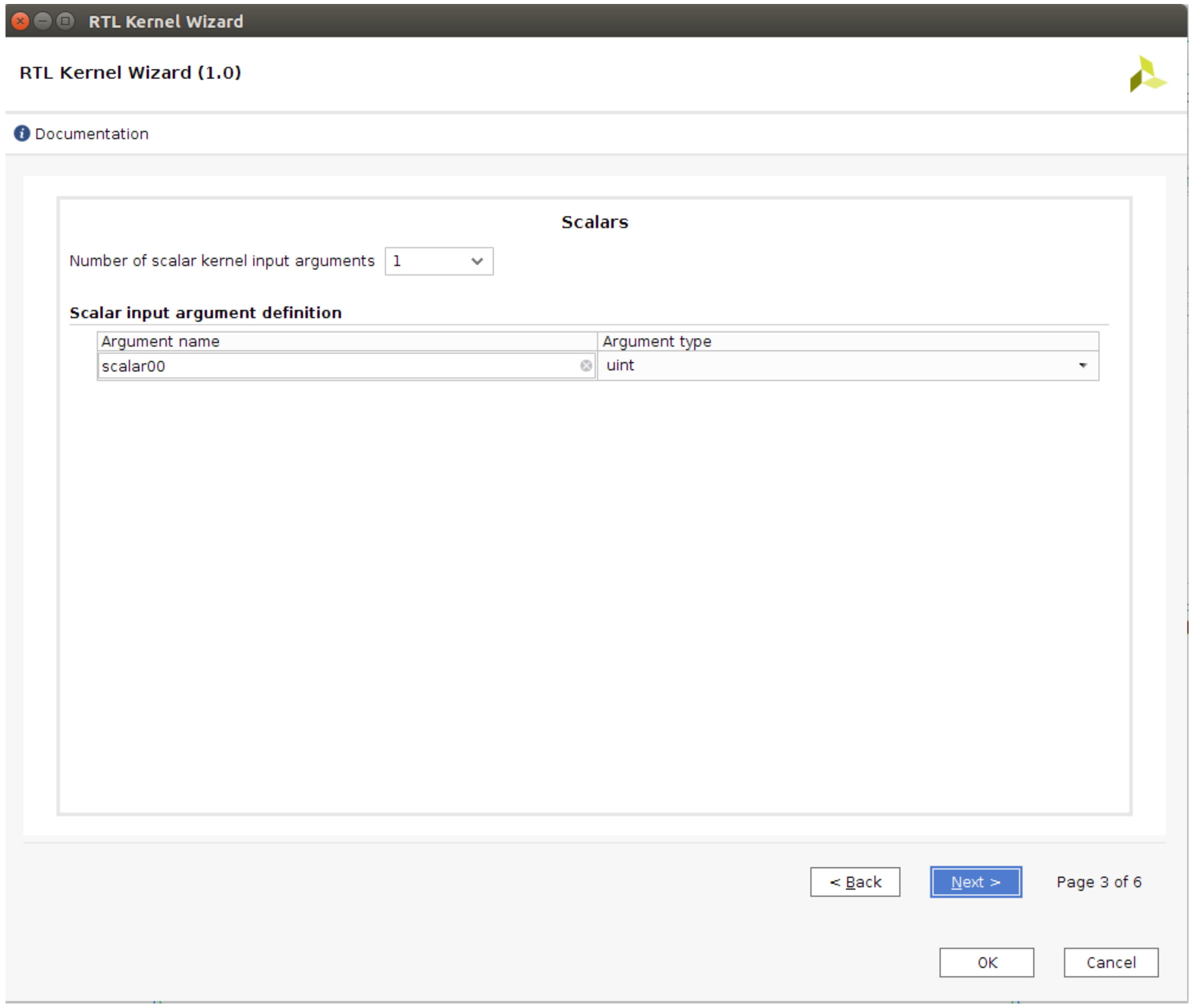
ap ctrl hs
ap ctrl chain
ap ctrl none

Clock and Reset options

Number of clocks	1
Has reset	0

RTL Kernel Wizard – Scalar Inputs

- Used to pass the control type of information to the kernels through the AXI4-Lite slave interface
- Scalar arguments cannot be read back from the host
- For each argument a corresponding control register is created to facilitate passing the argument from software to hardware
- Argument types affect the width of the control register in the generated Verilog module


RTL Kernel Wizard

RTL Kernel Wizard (1.0)

Documentation

Scalars

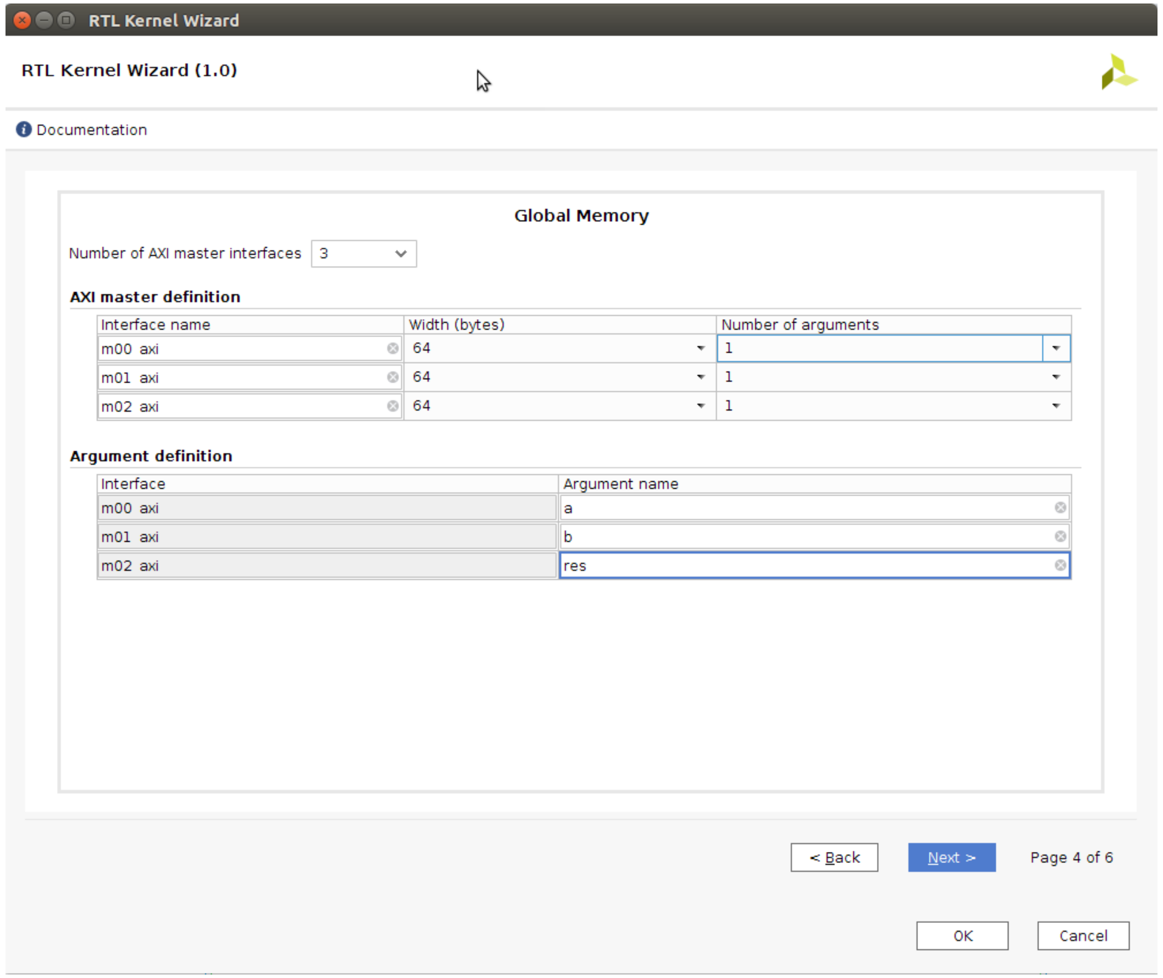
Number of scalar kernel input arguments: 1

Scalar input argument definition

Argument name	Argument type
scalar00	uint

RTL Kernel Wizard – Global Memory

- Specify the number of AXI master ports
- Assign arguments for each AXI master port
- Multiple arguments can share the same AXI master port
- Host provides the base address for each argument through the AXI-Lite interface during runtime



Global Memory

Number of AXI master interfaces: 3

AXI master definition

Interface name	Width (bytes)	Number of arguments
m00_axi	64	1
m01_axi	64	1
m02_axi	64	1

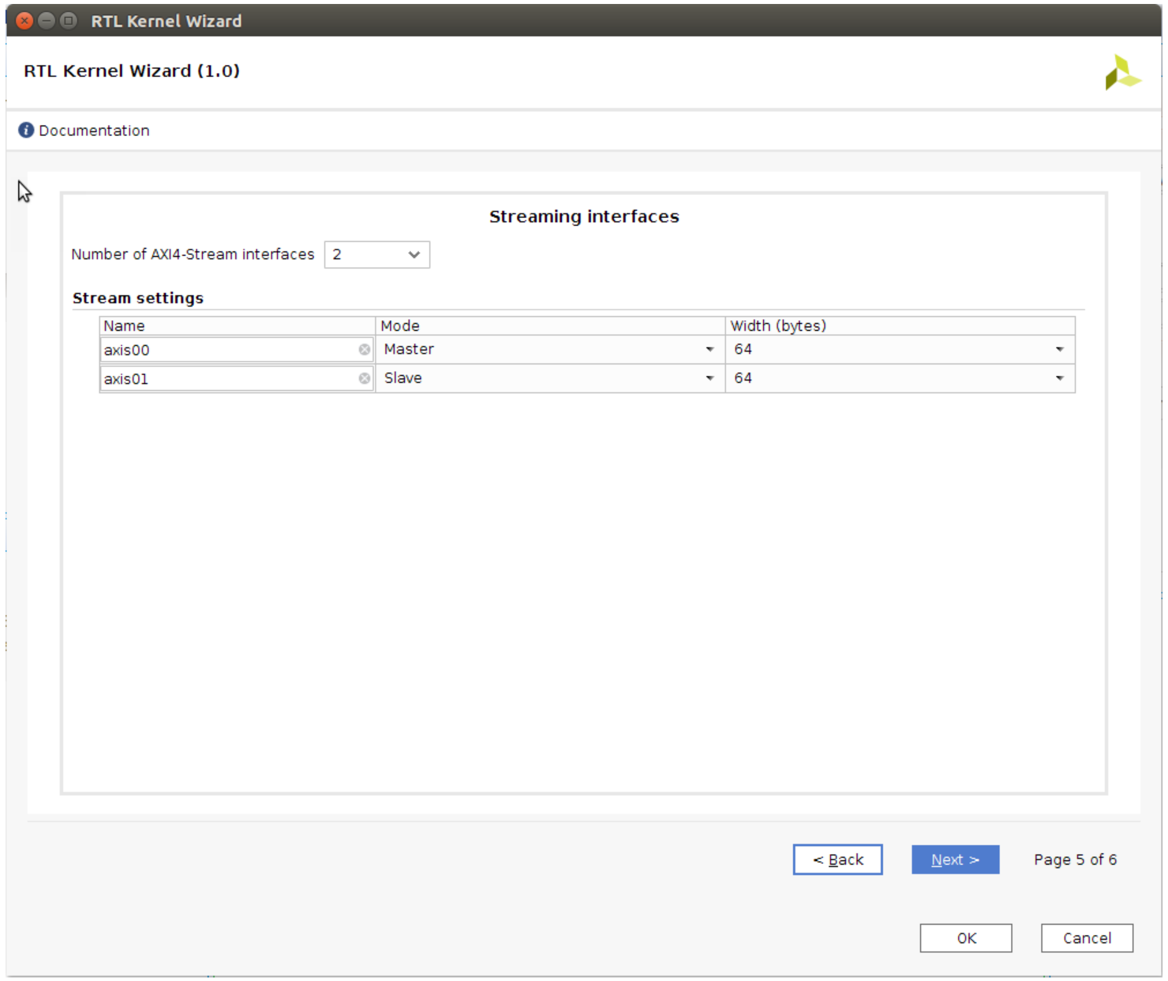
Argument definition

Interface	Argument name
m00_axi	a
m01_axi	b
m02_axi	res

Page 4 of 6

RTL Kernel Wizard – Streaming Interfaces

- Specifies the number of AXI4-Stream interfaces
- Specifies the direction of the interface
  - Master is write only; only host can retrieve data from it
  - Slave is read only; only host can send data to it



Streaming interfaces

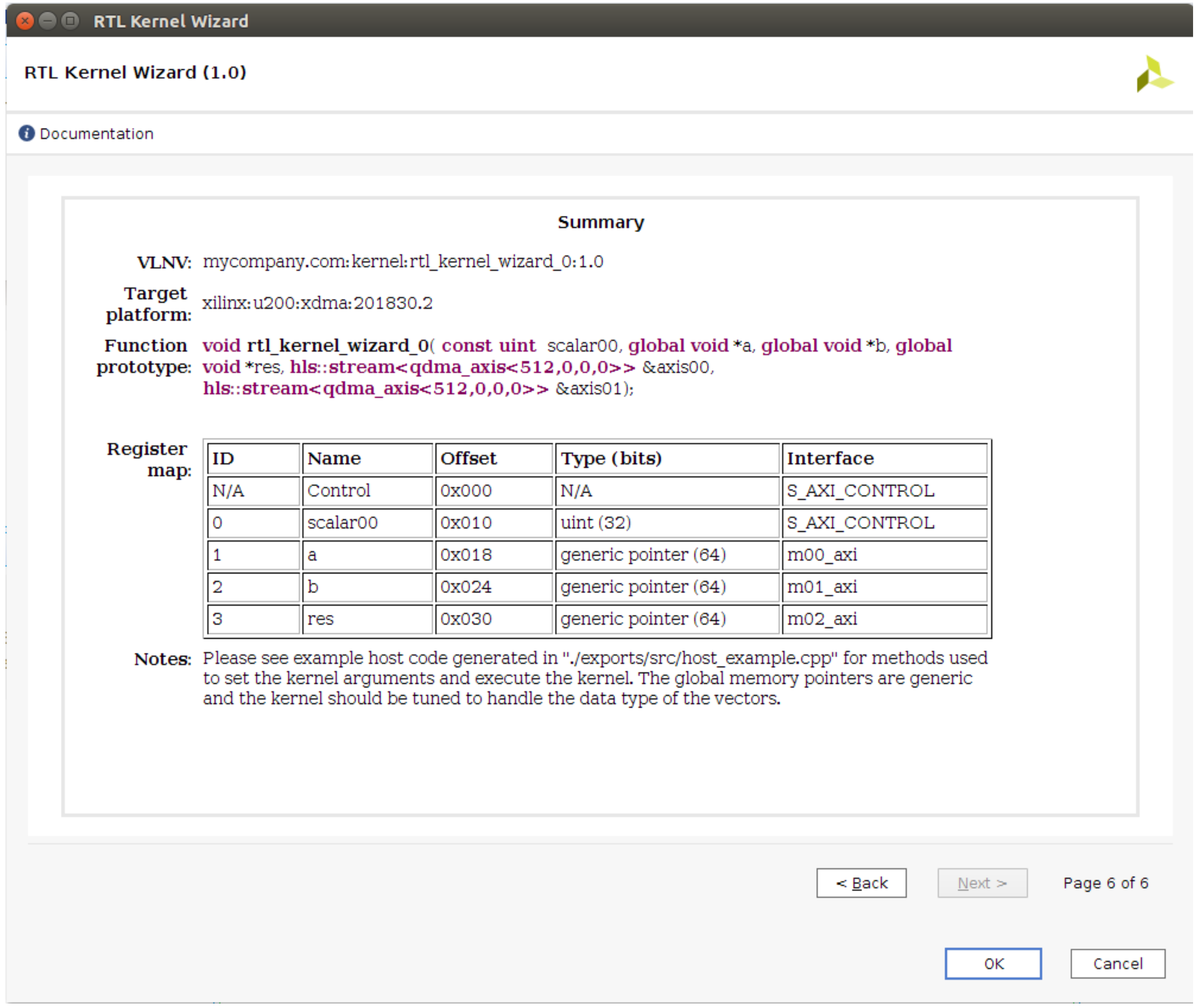
Number of AXI4-Stream interfaces: 2

Stream settings

Name	Mode	Width (bytes)
axis00	Master	64
axis01	Slave	64

RTL Kernel Wizard – Summary

- Gives a summary of what was created from options selected in the previous stages
- Function prototype conveys what a kernel call would like if it was a C function
- Register map shows the relationship between host software ID, argument name, hardware register offset, type, and associated interface



Summary

VLNV: mycompany.com:kernel:rtl_kernel_wizard_0:1.0

Target platform: xilinx:u200:xdma:201830.2

Function prototype: void rtl_kernel_wizard_0(const uint scalar00, global void *a, global void *b, global void *res, hls::stream<qdma_axis<512,0,0,0>> &axis00, hls::stream<qdma_axis<512,0,0,0>> &axis01);

Register map:

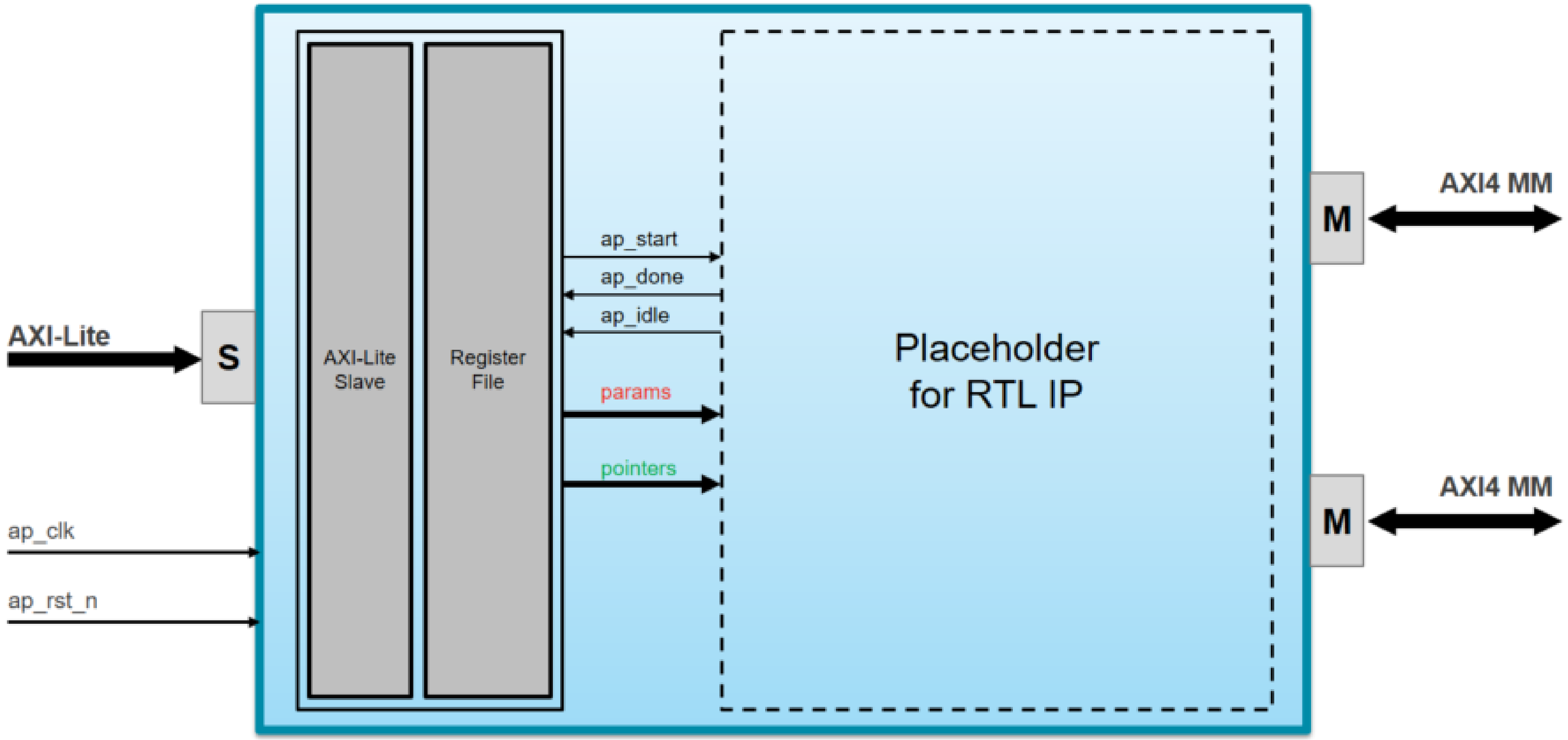
ID	Name	Offset	Type (bits)	Interface
N/A	Control	0x000	N/A	S_AXI_CONTROL
0	scalar00	0x010	uint (32)	S_AXI_CONTROL
1	a	0x018	generic pointer (64)	m00_axi
2	b	0x024	generic pointer (64)	m01_axi
3	res	0x030	generic pointer (64)	m02_axi

Notes: Please see example host code generated in "./exports/src/host_example.cpp" for methods used to set the kernel arguments and execute the kernel. The global memory pointers are generic and the kernel should be tuned to handle the data type of the vectors.

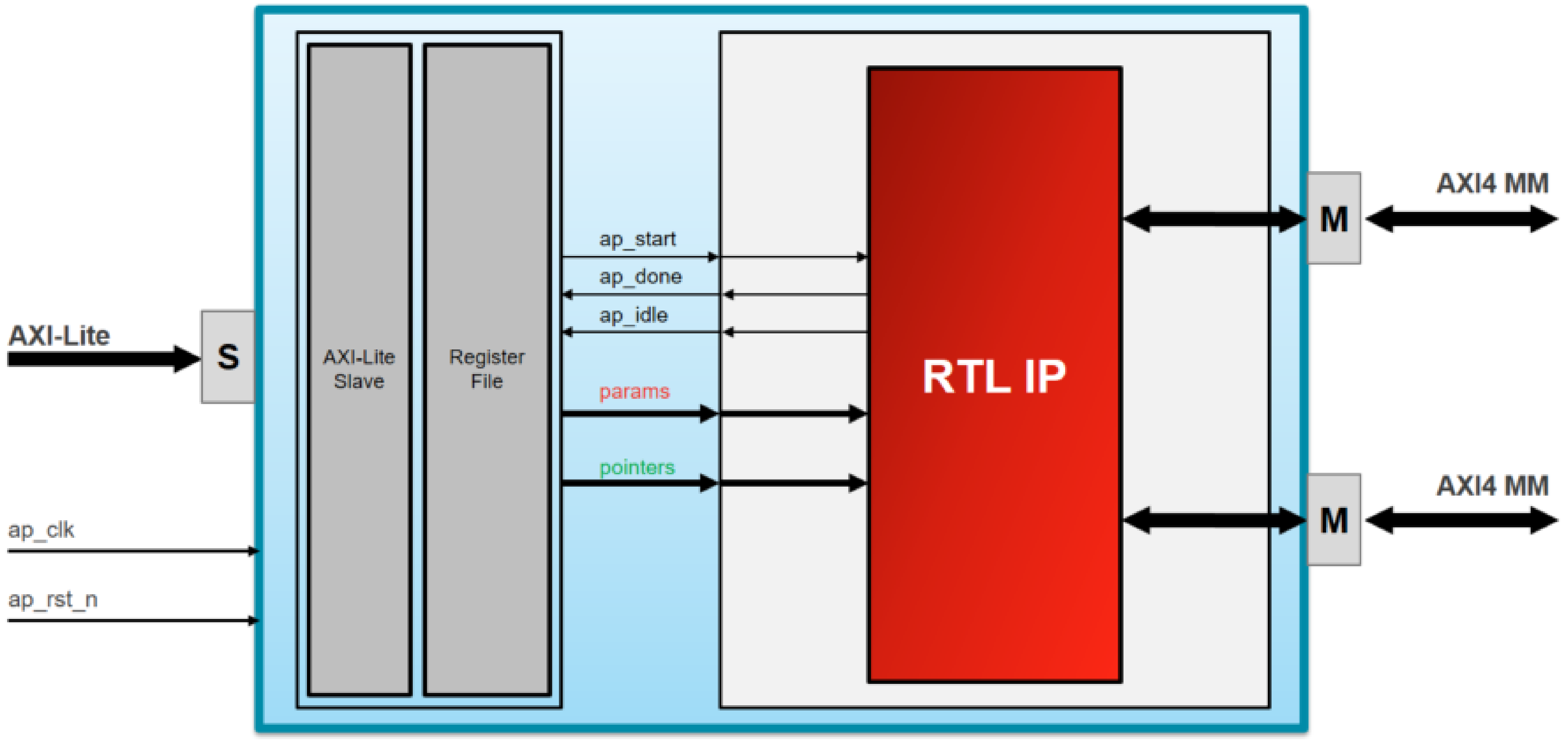
Kernel Operation Modes

- ap_ctrl_hs
  - Kernel is sequentially executed. Asserts ap_start and waits for ap_done
  - Restart the kernel when it is done. Does not support pipeline
  - This is the mode supported by older version of the tools
- ap_ctrl_chain
  - Kernel is pipelined. Asserts ap_start and waits for ap_done. Use ap_continue to allow the kernel to continue
  - Recommended for pipeline kernel execution
- ap_ctrl_none
  - Kernel is free running. Starts as soon as design is out of reset and never stops running

Generated RTL Kernel Wrapper

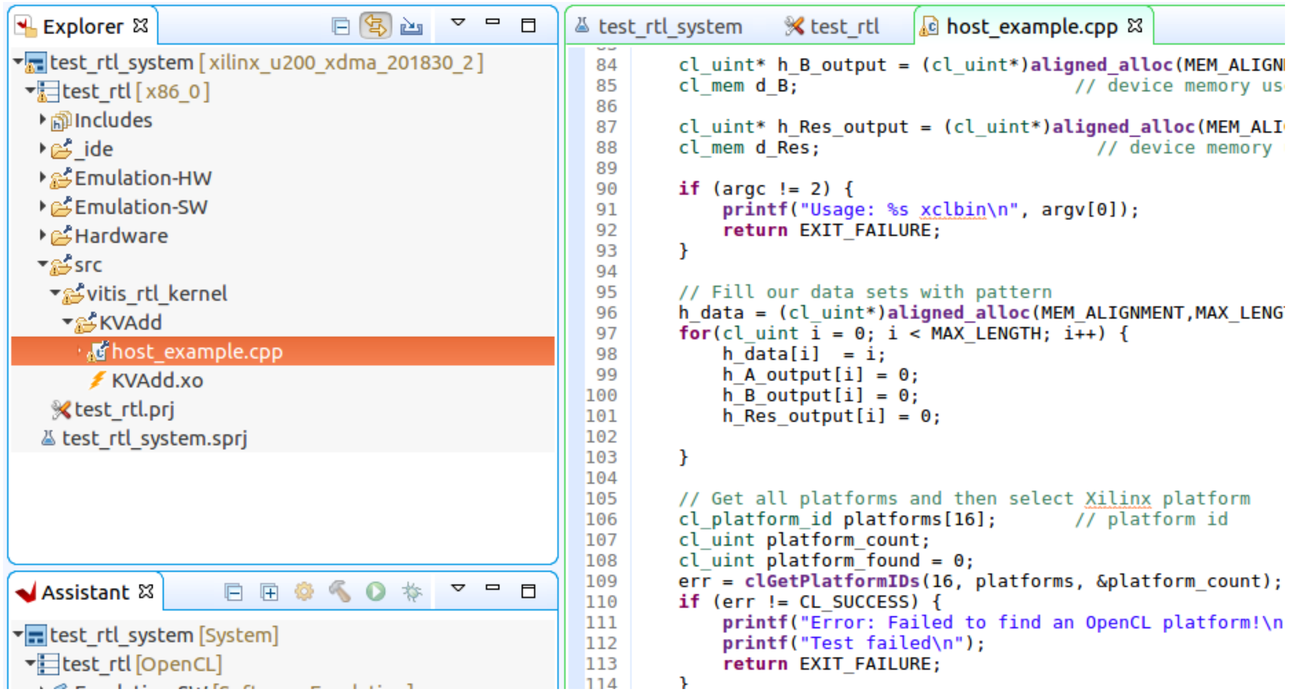


Instantiating Your RTL IP into the Kernel Wrapper



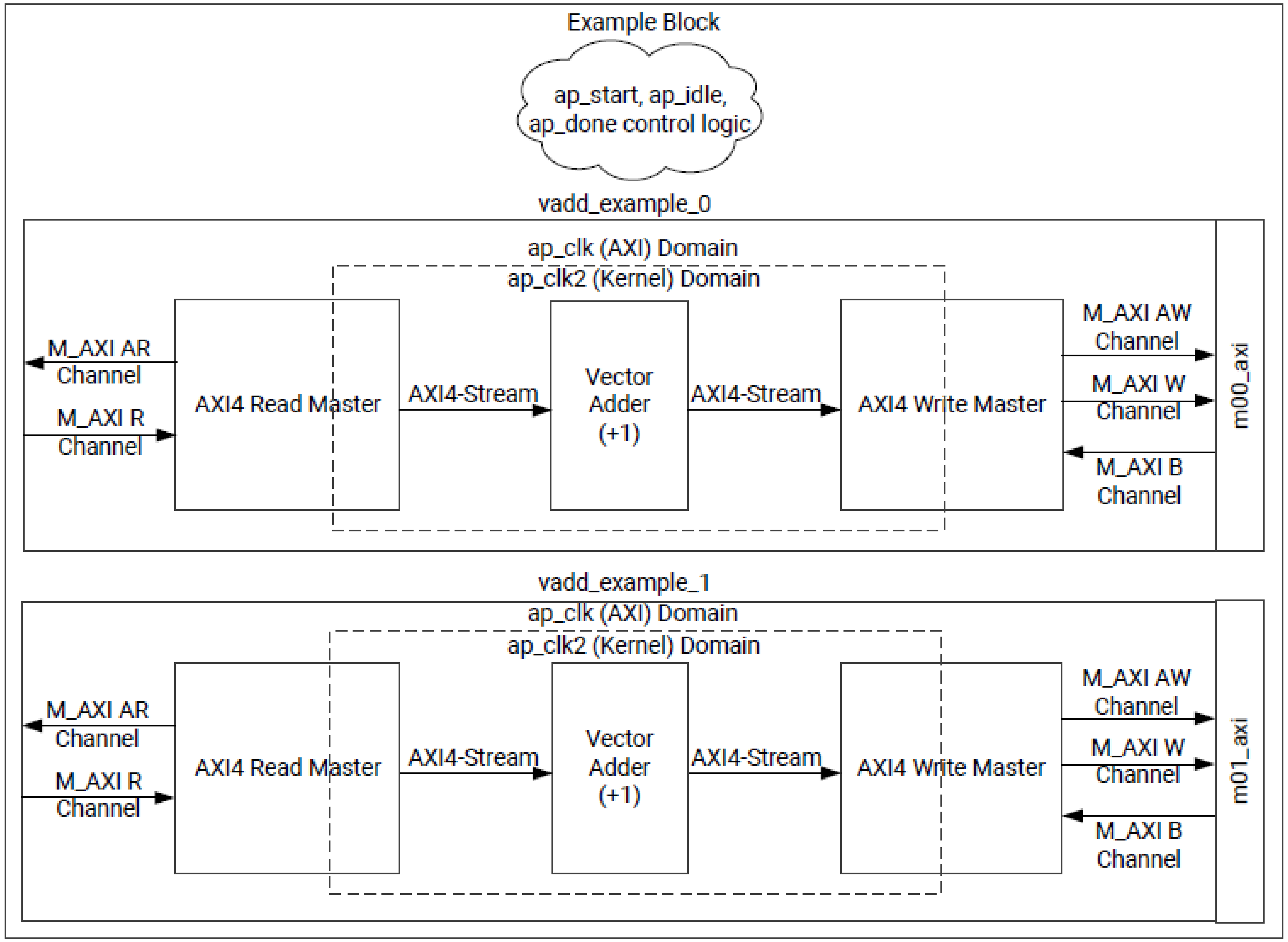
Host Code Template

- Sample host code to exercise the example kernel (host_example.cpp)
- Performs the following tasks
  - FPGA accelerator platform setup
  - Execution of accelerator
  - Post processing / FPGA accelerator cleanup
- Can be reused and adapted to exercise the custom RTL kernel



```
    cl_uint* h_B_output = (cl_uint*)aligned_alloc(MEM_ALIGN
    cl_mem d_B;                                     // device memory us

    cl_uint* h_Res_output = (cl_uint*)aligned_alloc(MEM_ALI
    cl_mem d_Res;                                     // device memory

    if (argc != 2) {
        printf("Usage: %s xclbin\n", argv[0]);
        return EXIT_FAILURE;
    }

    // Fill our data sets with pattern
    h_data = (cl_uint*)aligned_alloc(MEM_ALIGNMENT,MAX_LENG
    for(cl_uint i = 0; i < MAX_LENGTH; i++) {
        h_data[i]  = i;
        h_A_output[i] = 0;
        h_B_output[i] = 0;
        h_Res_output[i] = 0;

    }

    // Get all platforms and then select Xilinx platform
    cl_platform_id platforms[16];       // platform id
    cl_uint platform_count;
    cl_uint platform_found = 0;
    err = clGetPlatformIDs(16, platforms, &platform_count);
    if (err != CL_SUCCESS) {
        printf("Error: Failed to find an OpenCL platform!\n
        printf("Test failed\n");
        return EXIT_FAILURE;
    }
```

Generated RTL Kernel Example Block



Thank You