RTL Kernel Wizard

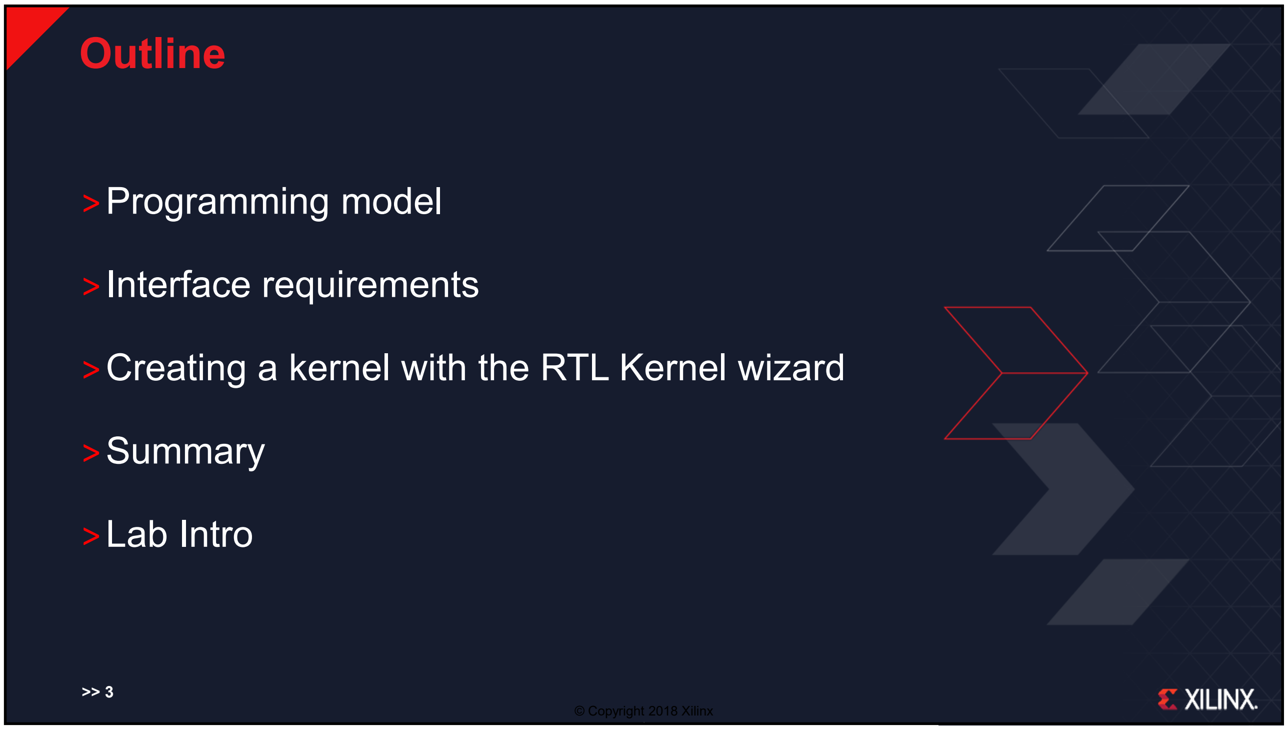
SDx 2018.2

XILINX.

Objectives

> **After completing this module, you will be able to:**
>> List RTL Kernel interface requirements
>> Create a RTL kernel using wizard

XILINX.

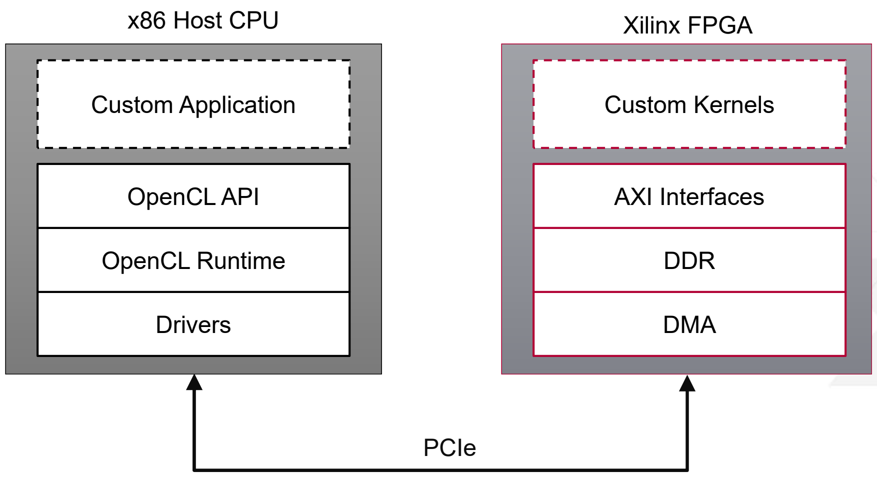


Outline

- > Programming model
- > Interface requirements
- > Creating a kernel with the RTL Kernel wizard
- > Summary
- > Lab Intro

XILINX.

AWS F1 System Overview



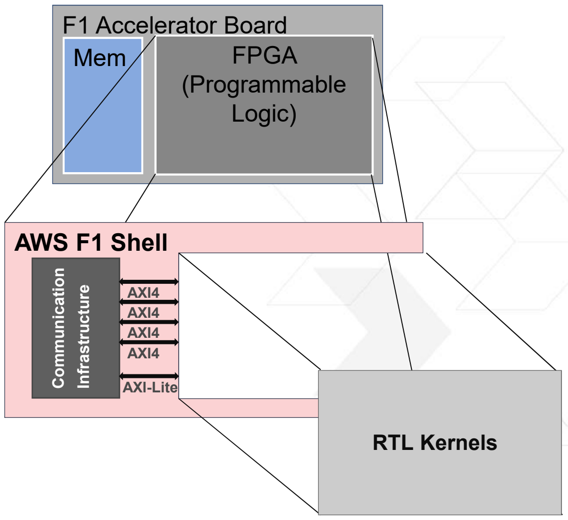
XILINX.

RTL Kernel Interface to AWS F1 Shell

> The AWS F1 Shell provides a thin layer of basic blocks to wrap the acceleration kernels

> The AWS F1 Shell requires specific Accelerator Kernel Interfaces

> The bitsream for the FPGA consists of the AWS F1 Shell combined with accelerator kernels



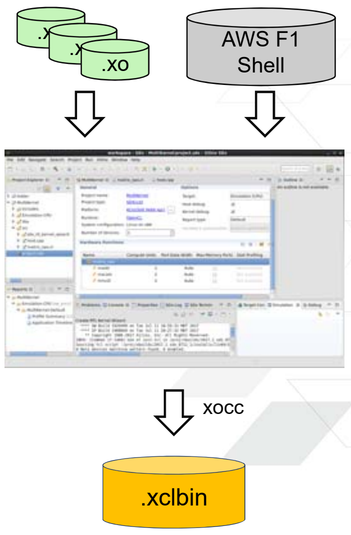
SDAccel Assembles Complete FPGA Platform

> Kernels are packaged into .XO files usable by SDAccel

> SDAccel integrates and connects kernels with the F1 shell

> SDAccel run synthesis and place & route on the assembled design

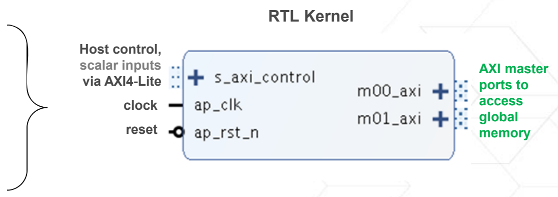
> SDAccel generates .xclbin bitstream for deployment on an FPGA board



RTL Kernel: Programming Paradigm (1/2)

> SDAccel associates specific C function argument types (host-code) with specific HW ports types (RTL kernel)

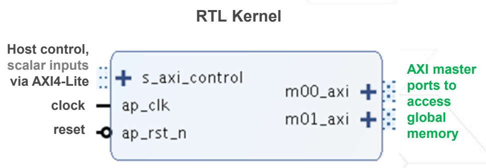
```
  void func( int length,
             int *a,
             int *b,
             int *output ) { … }
```


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 needs a AXI-Lite Slave port for scalars arguments

> RTL kernel needs a AXI MM Master port for pointer arguments

RTL Kernel: Programming Paradigm (2/2)


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

> Scalar arguments:
- >> Inputs only
- >> Written to the kernel via AXI4-lite interface

> Pointer arguments:
- >> Inputs or outputs
- >> Data resides in the global memory
- >> Kernel is responsible for accessing the data through the AXI4 master interface
- >> The base address of the memory is passed via the AXI4-lite interface

> The kernel is started and polled for completion status via AXI4-Lite

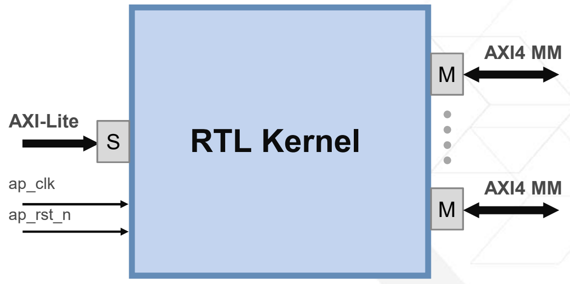
Interface Requirements

RTL Kernel Interface Requirements

> Clock and Reset

> AXI-Lite Slave interface

> AXI4 MM Master interface(s)



RTL 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

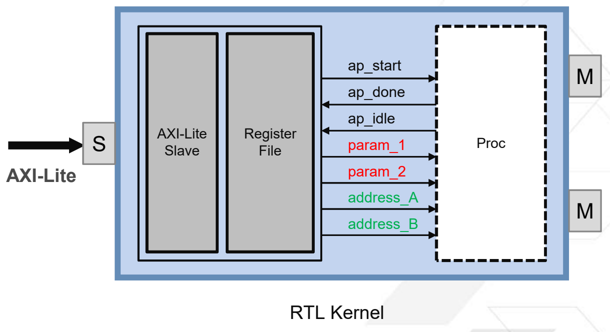
RTL Interface Requirements – AXI-Lite Slave

> **RTL kernel must have one (and only one) AXI-Lite Slave interface**

> **The kernel control interface**

> **Used 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 address 0x00 in the register file

Bit	Name	Description
0	Start	The kernel should start processing data when this bit is set
1	Done	The kernel should assert this signal when the processing is done. This bit is cleared on read
2	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
Others	N/A	Reserved

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

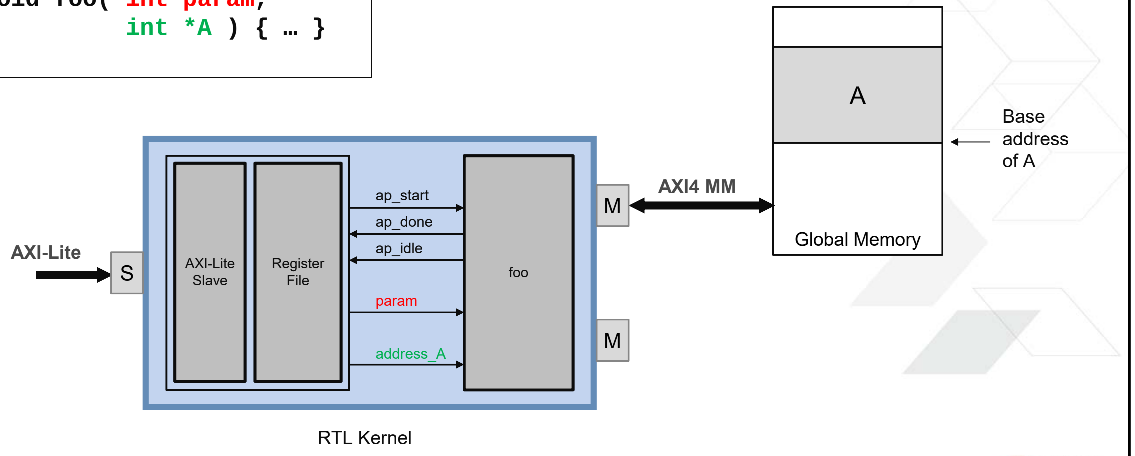
> All AXI4 master interfaces must have 64-bit addresses

> Global memory management using the AXI4 master ports should be based on the performance and bandwidth requirements of the design
>> Recommend one master interface per required DDR channel

AXI4 MM Master – Data and Base Address Example

Host code

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



Creating a kernel with the RTL Kernel wizard

RTL Kernel Wizard Overview

> **Provides an easy means of packaging an RTL IP into an SDAccel Kernel**

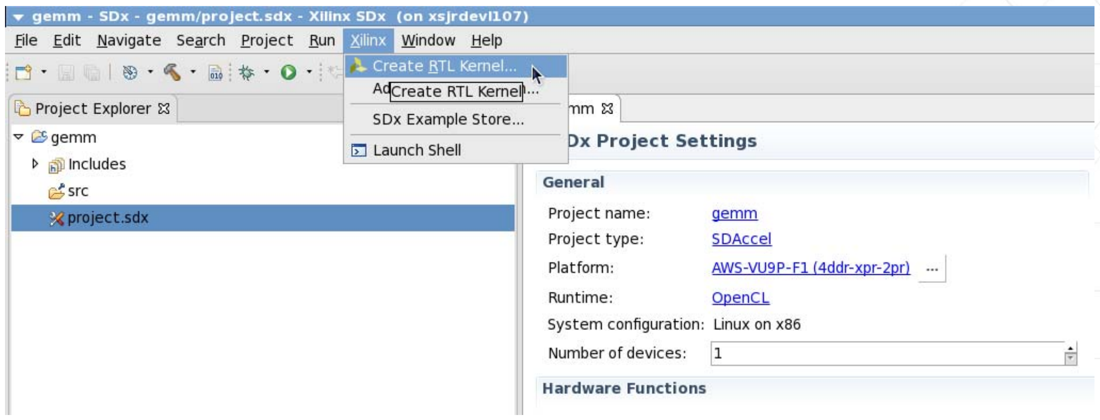
> **Creates a top level RTL Kernel wrapper that contains:**
- AXI-lite interface module including control logic and register file
- Example kernel IP module to be replaced with the actual RTL IP design
- One or more AXI-master interfaces

> **Creates a Vivado project for the RTL Kernel wrapper and generated files**

> **Also provides a simple test infrastructure for the wrapper IP**
- RTL testbench for the RTL kernel wrapper only
- Sample host code to exercise the packaged RTL kernel

Invoking the RTL Kernel Wizard

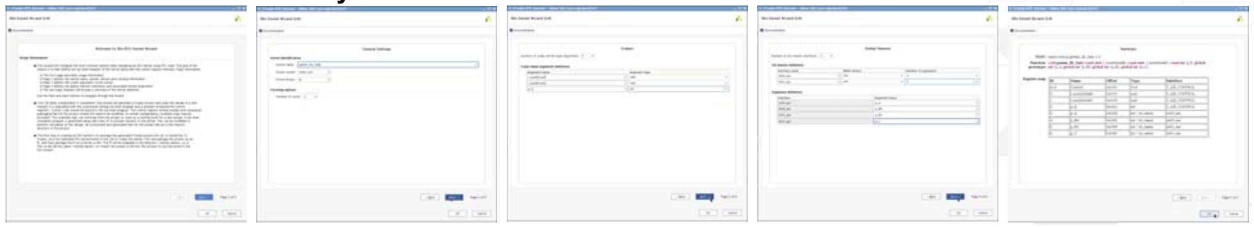
> **RTL kernel wizard is invoked from the SDAccel GUI**



Project name: gemm
Project type: SDAccel
Platform: AWS-VU9P-F1 (4ddr-xpr-2pr)
Runtime: OpenCL
System configuration: Linux on x86
Number of devices: 1

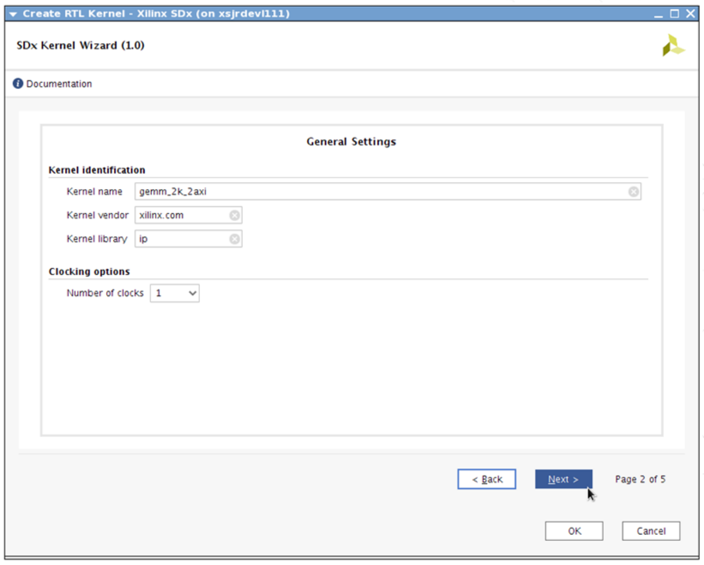
The 5 Stages of the Kernel Wizard

- Introduction and Usage
- Naming the RTL Kernel and specifying clock information
- Scalar input argument identification
- Specification of memory mapped AXI masters
- RTL Kernel summary



RTL Kernel Wizard – General Settings

- Kernel name is used with function `clCreateKernel` in the host code to reference the kernel
- Number of clocks determines if the RTL IP includes a secondary clock and reset
- Clock frequency is specified during final assembly of the Kernel with the F1 Shell



Create RTL Kernel - Xilinx SDx (on xsjrdevl111)

SDx Kernel Wizard (1.0)

Documentation

General Settings

Kernel identification

Kernel name	gemm_2k_2axi
Kernel vendor	xilinx.com
Kernel library	ip

Clocking options

Number of clocks	1

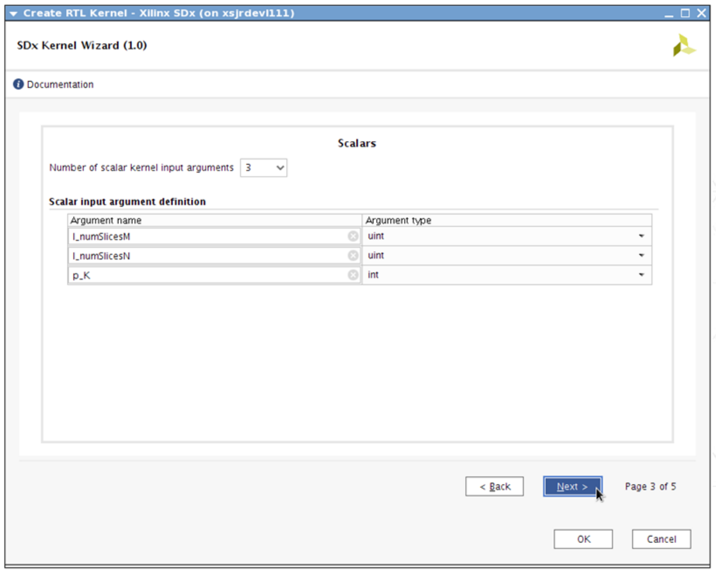
RTL Kernel Wizard – Scalar inputs

> Used to pass 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



Scalars

Number of scalar kernel input arguments: 3

Scalar input argument definition

Argument name	Argument type
l_numSlicesM	uint
l_numSlicesN	uint
p_K	int

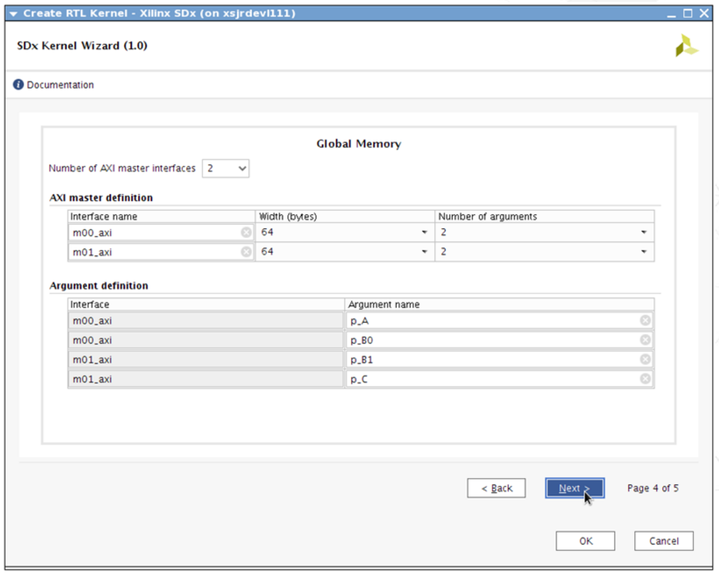
RTL Kernel Wizard – Memory Mapped Interface

> Specify the number of AXI master ports

> Assign arguments for each AXI master port

> Multiple arguments can share the same AXI master port

> The host provides the base address for each argument through the AXI-lite interface during runtime



Global Memory

Number of AXI master interfaces: 2

AXI master definition

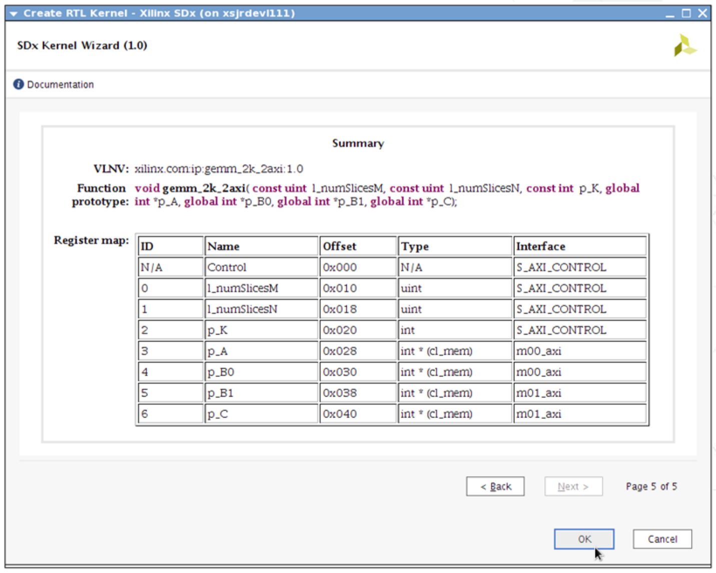
Interface name	Width (bytes)	Number of arguments
m00_axi	64	2
m01_axi	64	2

Argument definition

Interface	Argument name
m00_axi	p_A
m00_axi	p_B0
m01_axi	p_B1
m01_axi	p_C

RTL Kernel Wizard – Summary

- Gives a summary of what was created from options selected in the previous pages
- The function prototype conveys what a kernel call would like if it was a C function
- The register map shows the relationship between host software ID, argument name, hardware register offset, type, and associated interface
- The ID is used with `clSetKernelArg` function in the host code



Summary

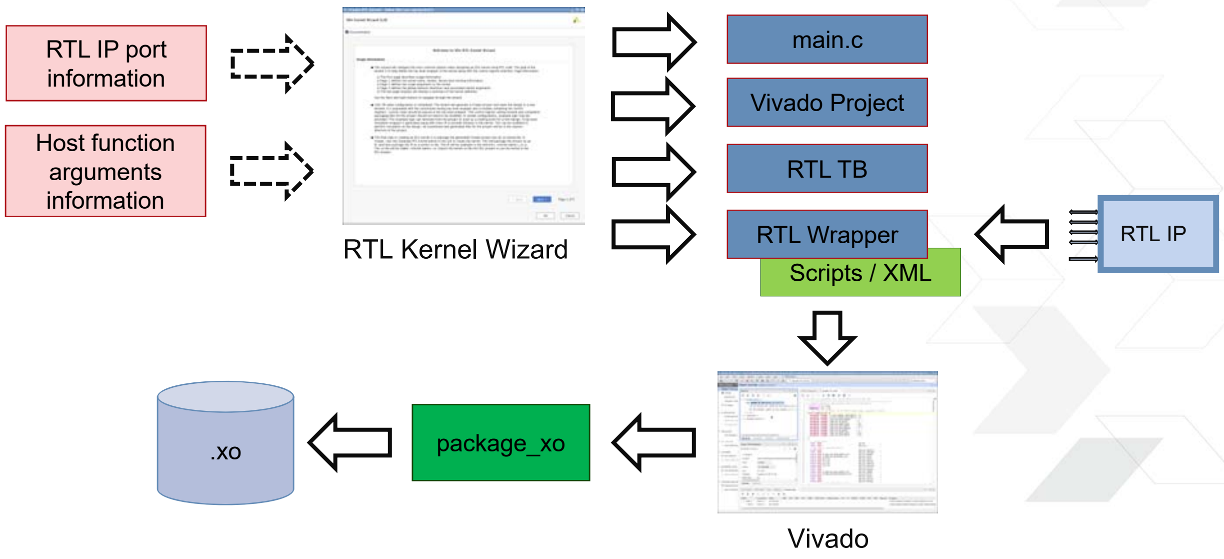
VLNV: xilinx.com:ip:gemm_2k_2axi:1.0

Function prototype: `void gemm_2k_2axi( const uint l_numSlicesM, const uint l_numSlicesN, const int p_K, global int *p_A, global int *p_B0, global int *p_B1, global int *p_C);`

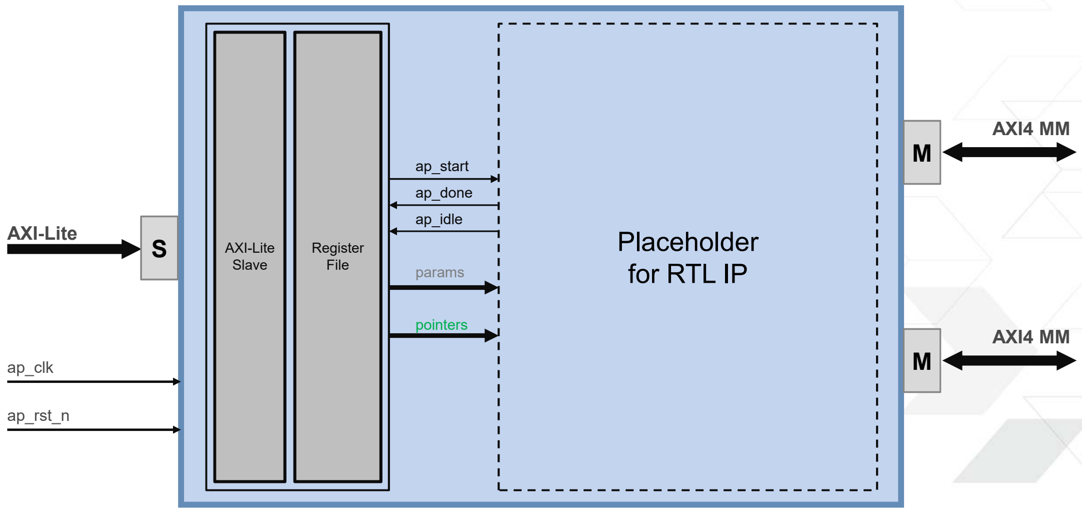
Register map:

ID	Name	Offset	Type	Interface
N/A	Control	0x000	N/A	S_AXI_CONTROL
0	l_numSlicesM	0x010	uint	S_AXI_CONTROL
1	l_numSlicesN	0x018	uint	S_AXI_CONTROL
2	p_K	0x020	int	S_AXI_CONTROL
3	p_A	0x028	int * (cl_mem)	m00_axi
4	p_B0	0x030	int * (cl_mem)	m00_axi
5	p_B1	0x038	int * (cl_mem)	m01_axi
6	p_C	0x040	int * (cl_mem)	m01_axi

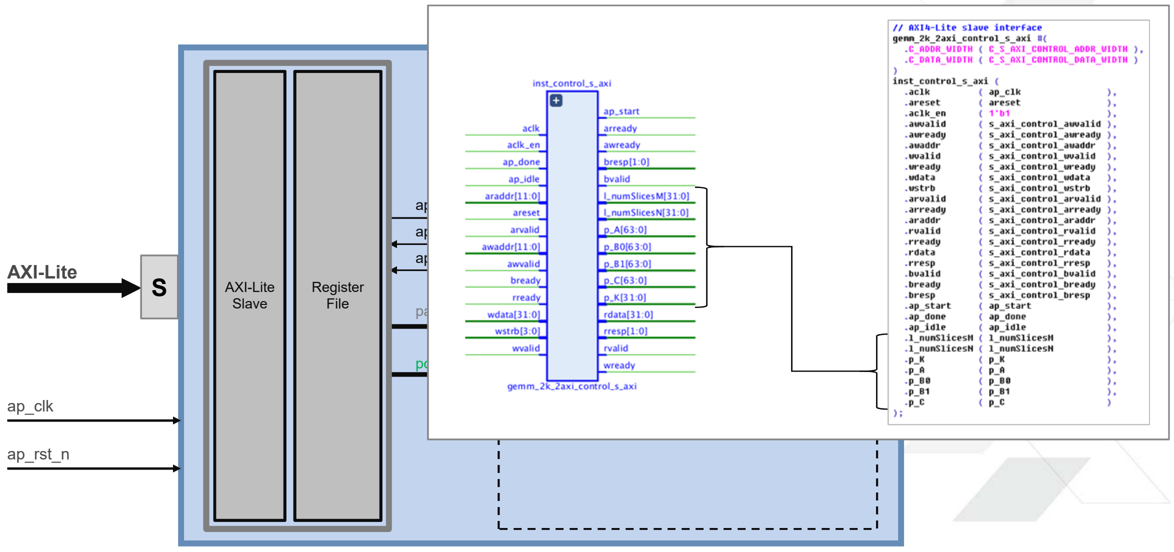
RTL Kernel Wizard – Generated Outputs



Generated RTL Kernel Wrapper

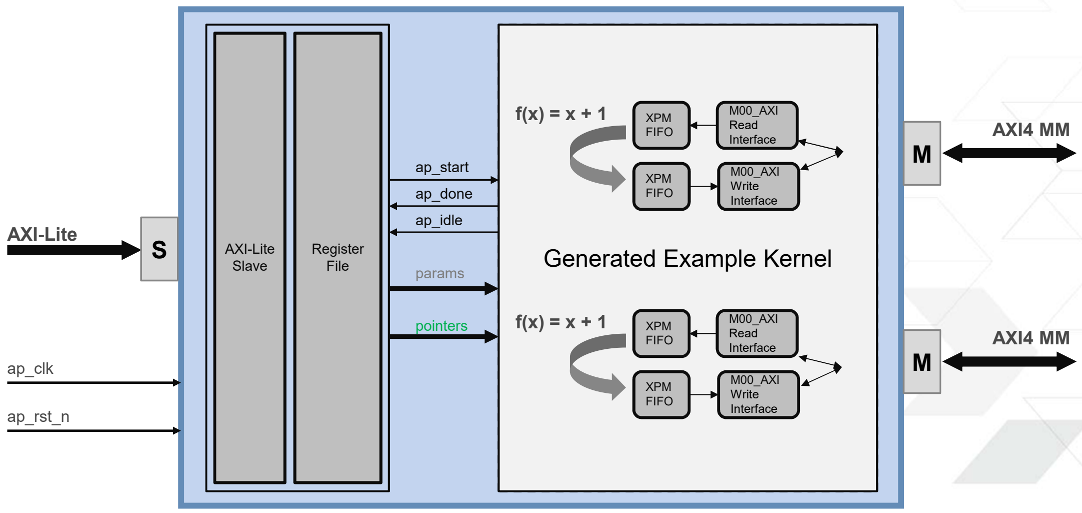


AXI-Lite Slave and Register File Auto-Generated



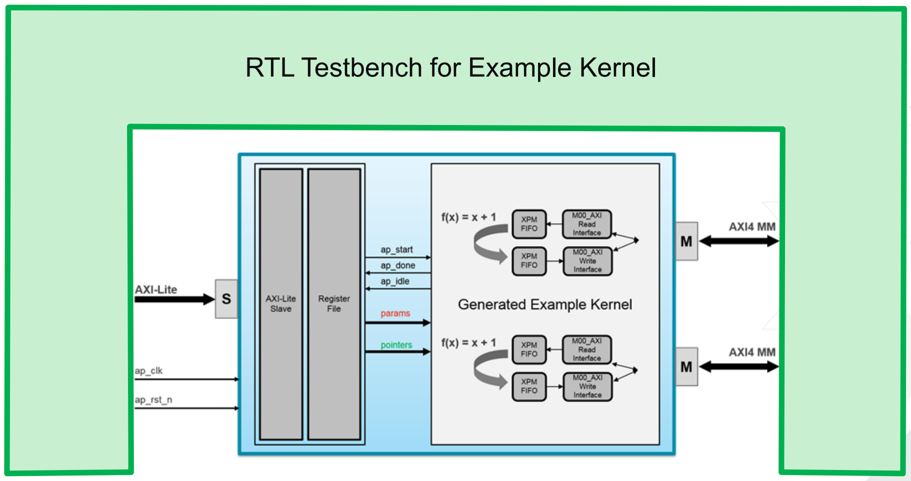
```
// AXI4-Lite slave interface
gemm_2k_2axi_control_s_axi #(
  .C_ADDR_WIDTH ( C_S_AXI_CONTROL_ADDR_WIDTH ),
  .C_DATA_WIDTH ( C_S_AXI_CONTROL_DATA_WIDTH )
)
inst_control_s_axi (
  .aclk         ( ap_clk                  ),
  .areset       ( areset                  ),
  .aclk_en      ( 1'b1                    ),
  .awvalid      ( s_axi_control_awvalid ),
  .awready      ( s_axi_control_awready ),
  .awaddr       ( s_axi_control_awaddr  ),
  .wvalid       ( s_axi_control_wvalid  ),
  .wready       ( s_axi_control_wready  ),
  .wdata        ( s_axi_control_wdata   ),
  .wstrb        ( s_axi_control_wstrb   ),
  .arvalid      ( s_axi_control_arvalid ),
  .arready      ( s_axi_control_arready ),
  .araddr       ( s_axi_control_araddr  ),
  .rvalid       ( s_axi_control_rvalid  ),
  .rready       ( s_axi_control_rready  ),
  .rdata        ( s_axi_control_rdata   ),
  .rresp        ( s_axi_control_rresp   ),
  .bvalid       ( s_axi_control_bvalid  ),
  .bready       ( s_axi_control_bready  ),
  .bresp        ( s_axi_control_bresp   ),
  .ap_start     ( ap_start                ),
  .ap_done      ( ap_done                 ),
  .ap_idle      ( ap_idle                 ),
  .l_numSlicesM ( l_numSlicesM            ),
  .l_numSlicesN ( l_numSlicesN            ),
  .p_K          ( p_K                     ),
  .p_A          ( p_A                     ),
  .p_B0         ( p_B0                    ),
  .p_B1         ( p_B1                    ),
  .p_C          ( p_C                     )
);
```

Generated Example Kernel



XILINX.

RTL Testbench for Example Kernel



XILINX.

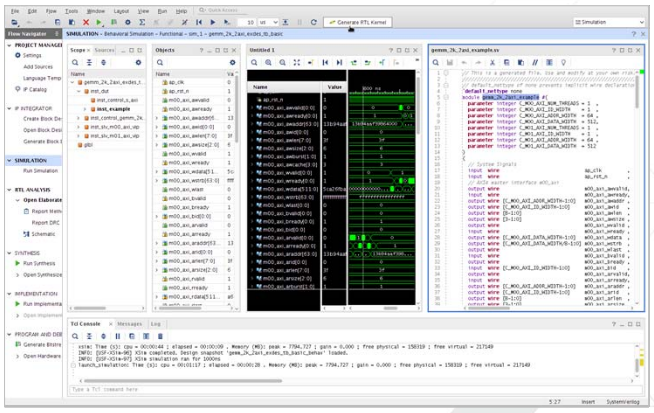
Vivado Project for the Example Kernel

> Vivado project created with RTL Kernel wrapper top level

> Vivado provides a complete RTL design and verification environment

> Build your RTL kernel

> Verify your design using integrated simulation tools



XILINX.

Host Code Template

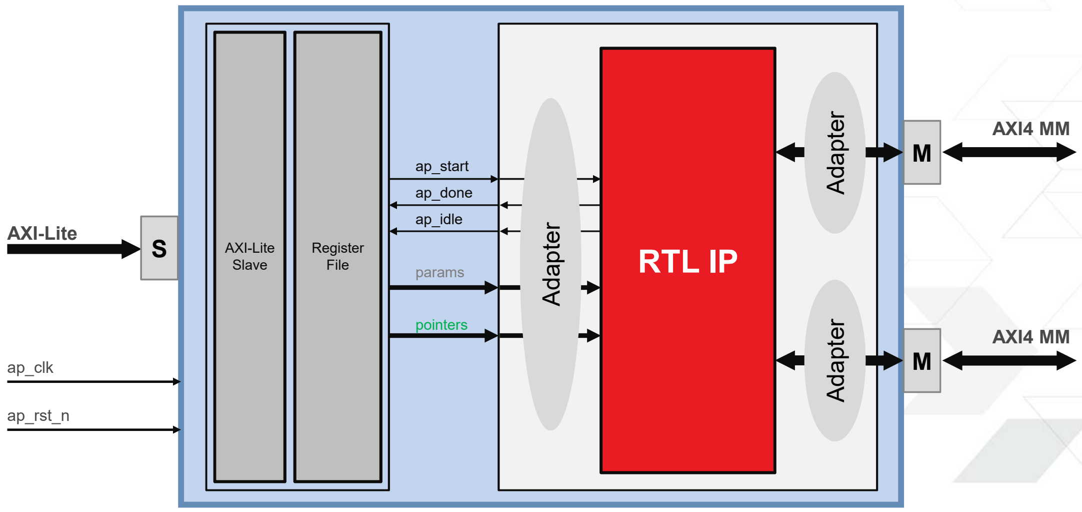
> Sample host code to exercise the example kernel (main.c)

> Performs to 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

XILINX.

Instantiating your RTL IP into the Kernel Wrapper



Packaging the RTL Kernel for SDAccel

> When the RTL Kernel is ready to be packaged, use `Flow > Generate RTL Kernel` in Vivado to generate a kernel container file (XO file)

> The XO file is a container file that includes the RTL files for the Kernel along with all the necessary information for integration of the kernel in SDAccel

> The XO file can be compiled into the platform and run in hardware, or hardware emulation flows in SDAccel

Summary

XILINX

Summary

> SDAccel provides RTL Kernel developers with a framework to integrate their hardware functions into an application running on a host PC connected to FPGA via PCIe

> RTL kernels need to have correct interfaces and packaging to be recognized by SDAccel tool flow

> RTL Kernel Wizard helps guide the user to adapt existing IP or create new IP for SDAccel

XILINX

Lab Intro

XILINX

Lab Intro

> In this lab you will use a RTL Kernel wizard to create an example template so a user can include their RTL code and generate an IP that can be used in a SDAccel project

XILINX

Adaptable.
Intelligent.

XILINX.