AMD

XILINX

Creating Processor System

Objectives

▸ After completing this module, you will be able to:

- Describe embedded system development flow in Zynq
- List the steps involved in creating hardware accelerator
- State how hardware accelerator created in Vitis HLS is used in Vivado Design Suite

AMD
XILINX

Outline

▸ Embedded System Design in Zynq using IP Integrator

▸ Creating Hardware Accelerator IP

▸ Integrating the Hardware Accelerator IP in Embedded System

▸ Summary

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

Embedded System Design in Zynq using IP Integrator

AMD
XILINX

Embedded Design Architecture in Zynq

▶ Embedded design in Zynq is based on:

- Processor and peripherals
  - Dual ARM® Cortex™ -A9 processors of Zynq-7000 SoC
  - AXI interconnect
  - AXI component peripherals
  - Reset, clocking, debug ports
- Software platform for processing system
  - Standalone OS
  - C language support
  - Processor services
  - C drivers for hardware
- User application
  - Interrupt service routines (optional)

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

The PS and the PL

▸ The Zynq-7000 SoC architecture consists of two major sections
- PS: Processing system
  - Single/Dual ARM Cortex-A9 processor based (Single core versions available)
  - Multiple peripherals
  - Hard silicon core
- PL: Programmable logic
  - Uses the same 7 series programmable logic

| Features	Zynq-7000S	Zynq-7000	
Devices	Z-7007S, Z-7012S, Z-7014S	Z-7010, Z-7015, Z-7020	Z-7030, Z-7035, Z-7045, Z-7100
Processor Core	Single-core ARM® Cortex™-A9 MPCore™	Dual-core ARM Cortex-A9 MPCore	
Maximum Frequency	Up to 766MHz	Up to 866 MHz	Up to 1GHz
External Memory Support	DDR3, DDR3L, DDR2, LPDDR2		
Key Peripherals	USB 2.0, Gigabit Ethernet, SD/SDIO		
Dedicated Peripheral Pins	Up to 128	Up to 128	128

AMD
XILINX

Vivado

▸ What are Vivado, IP Integrator and Vitis?

- Vivado is the tool suite for Xilinx FPGA design and includes capability for embedded system design

  - IP Integrator, is part of Vivado and allows block level design of the hardware part of an Embedded system

  - Vivado includes all the tools, IP, and documentation that are required for designing systems with the Zynq-7000 SoC hard core and/or Xilinx MicroBlaze soft core processor

  - Vivado + IPI replaces ISE/EDK

- Vitis core development kit includes comprehensive developer tools to design, debug, and deploy software applications targeted towards embedded processors

  - Xilinx Zynq, MPSoC, RFSoC, and ACAP devices.

▸ Vivado is the overall project manager and is used for developing non-embedded hardware and instantiating embedded systems

- Vivado/IP Integrator flow is recommended for developing Zynq embedded systems

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

Introduction to Embedded System Development
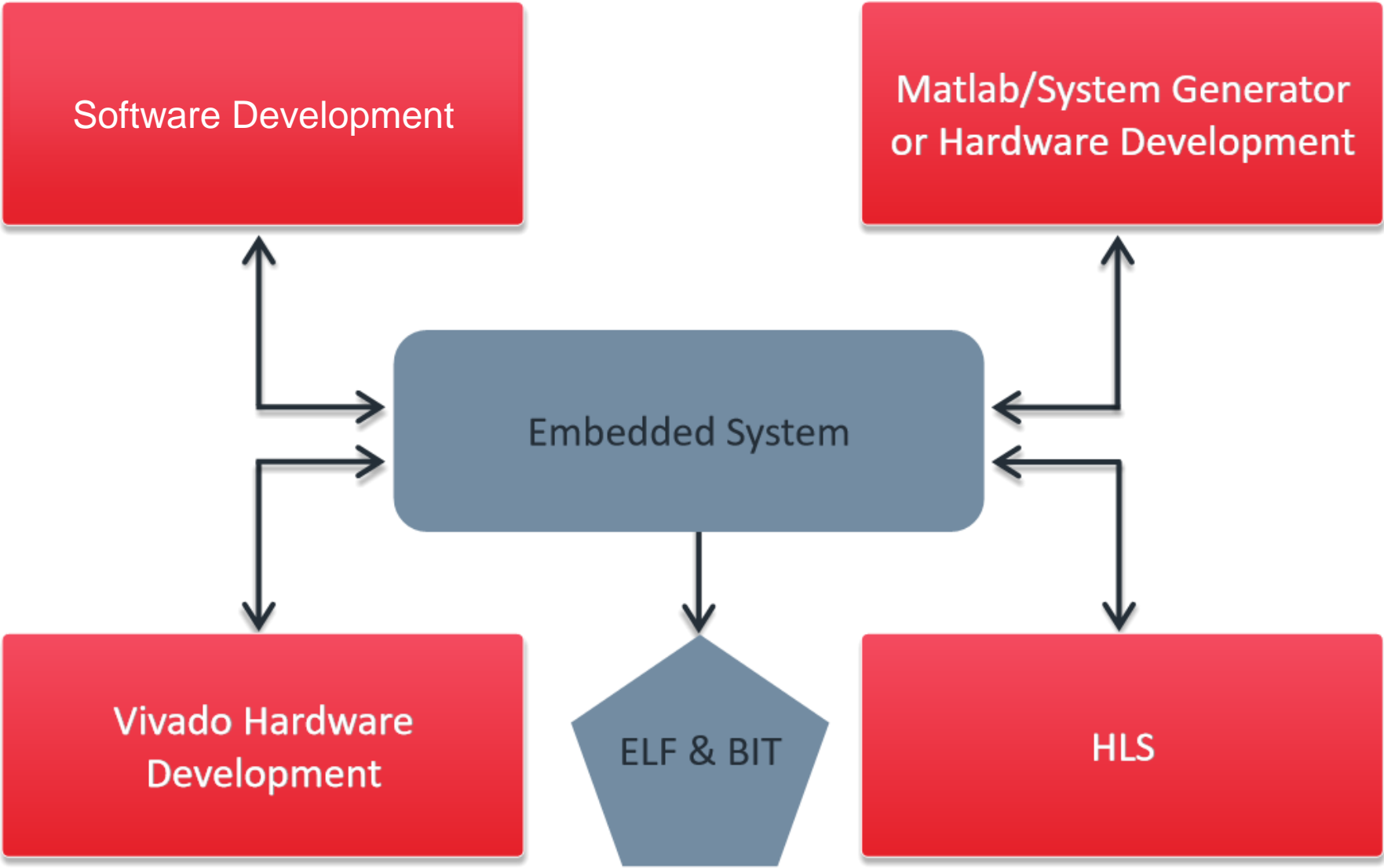
Embedded System Design Flow for Zynq-7000 SoC

AMD XILINX
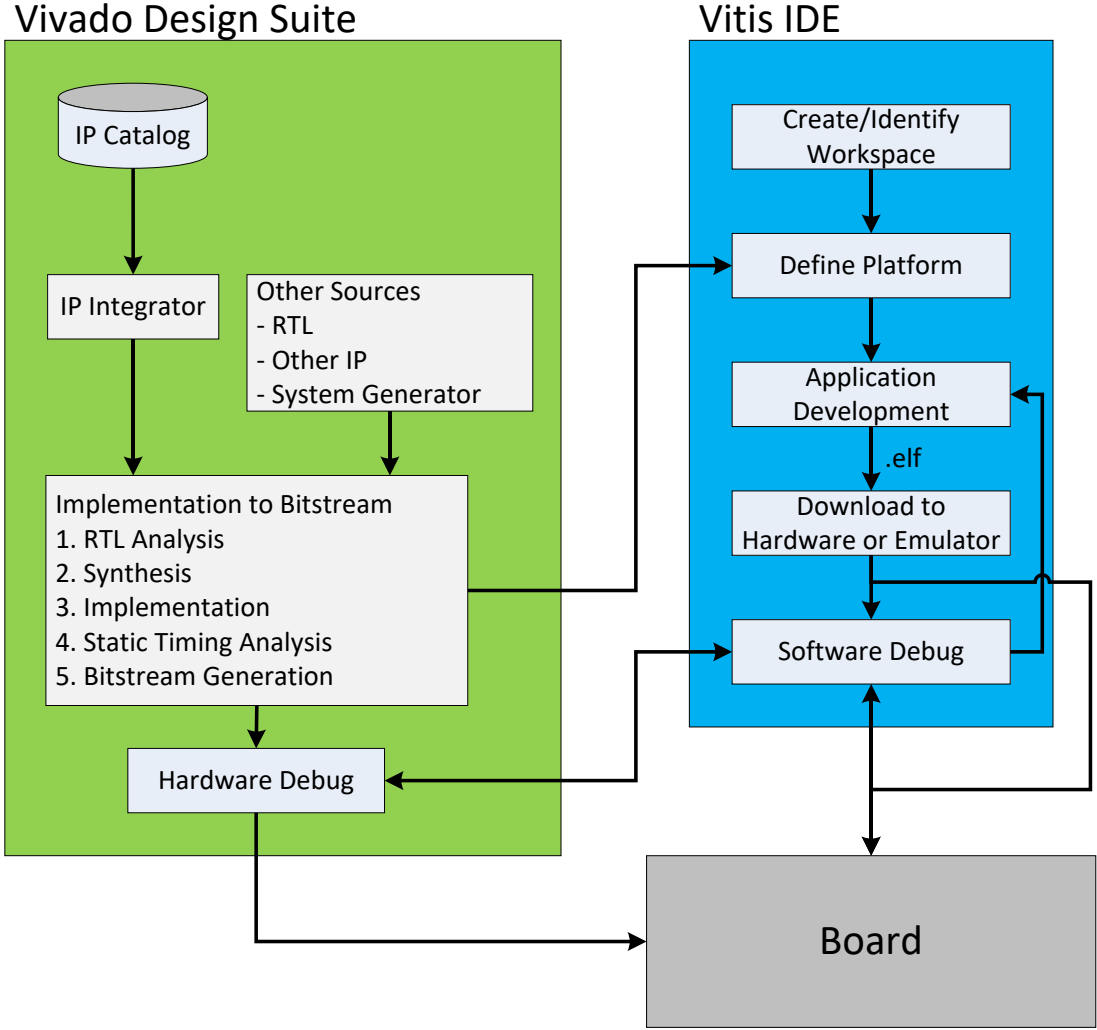
Embedded System Tools: Hardware

▶ Hardware development tools

- IP Integrator

- IP Packager

- Hardware netlist generation

- Simulation model generation

- Hardware debugging using Vivado analyzer cores
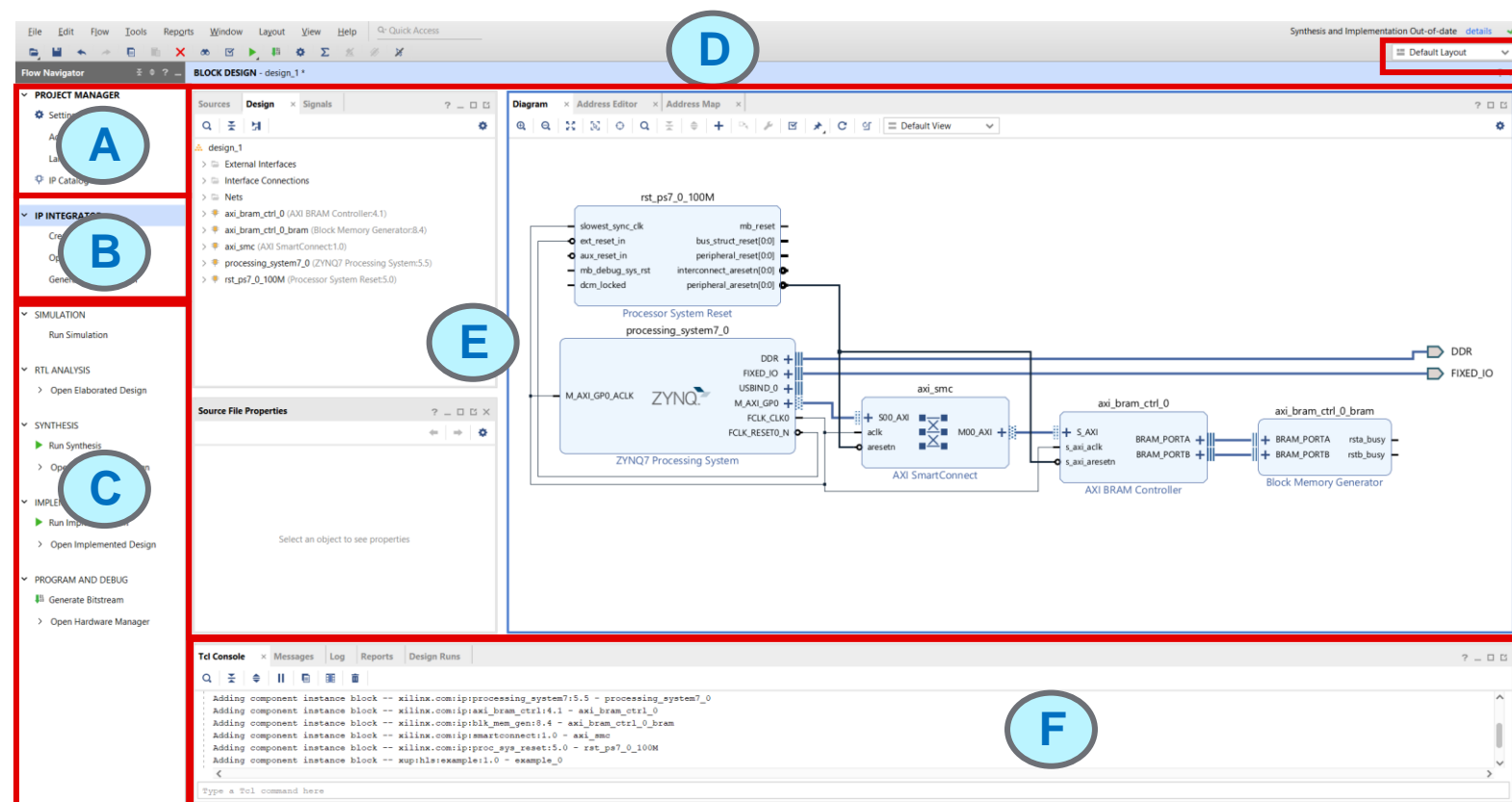
AMD
XILINX

Embedded System Tools: Software

▸ Vitis Unified Software Platform

- Board support package creation

- GNU software development tools

- C/C++ compiler for the MicroBlaze and ARM Cortex-A9 processors (gcc)

- Debugger for the MicroBlaze and ARM Cortex-A9 processors (system debugger)

- TCF framework – multicore debug

- Feature

  - Importing target platform definition created using Vivado® Design Suite

  - Application development for single, multi-processor and heterogenous processor systems

  - Ability to create and configure board support packages (BSPs) for third-party OS

  - Board bring-up and Firmware development

  - System-level performance analysis and benchmarking

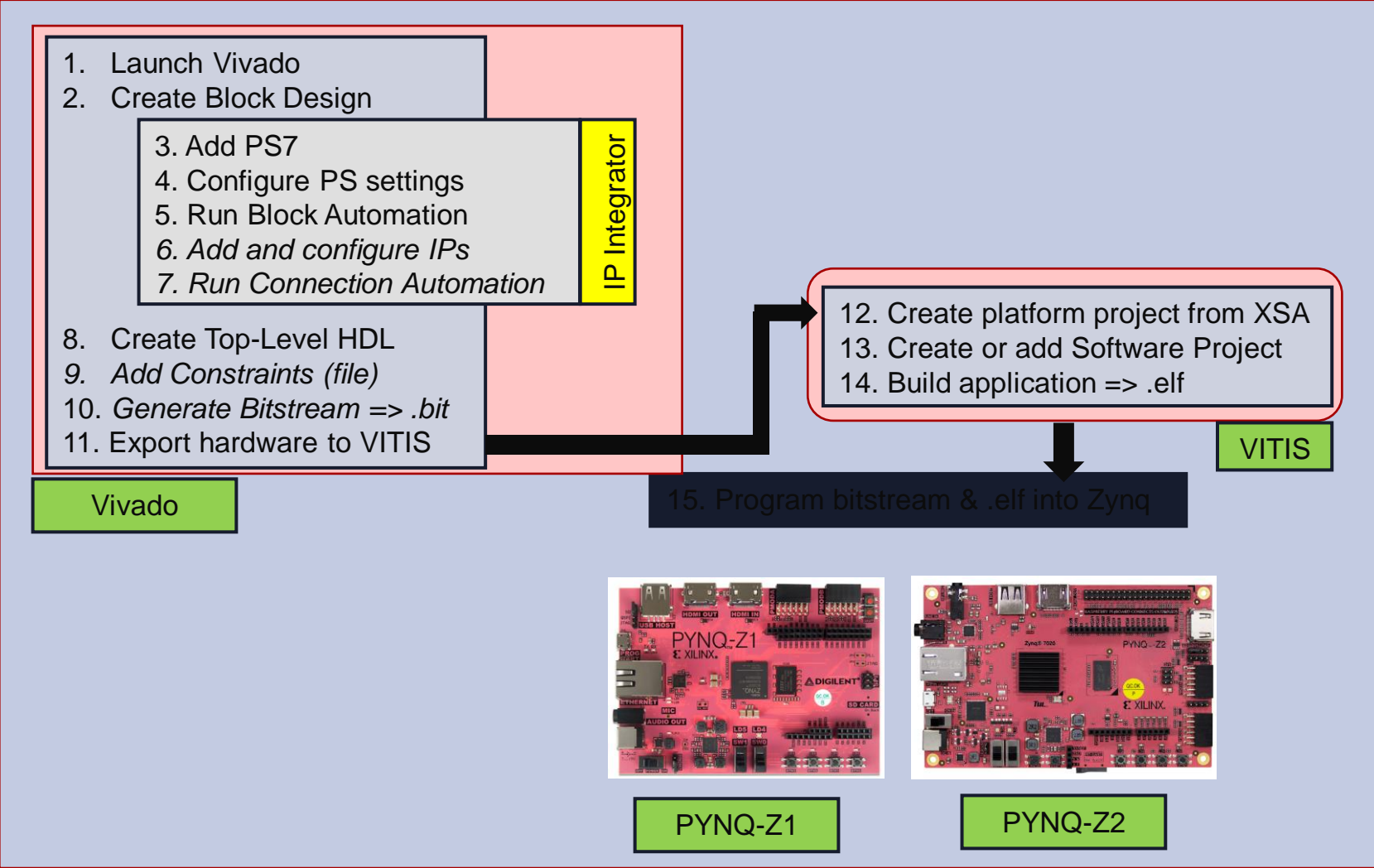 - Real-time debug and trace of heterogeneous embedded systems
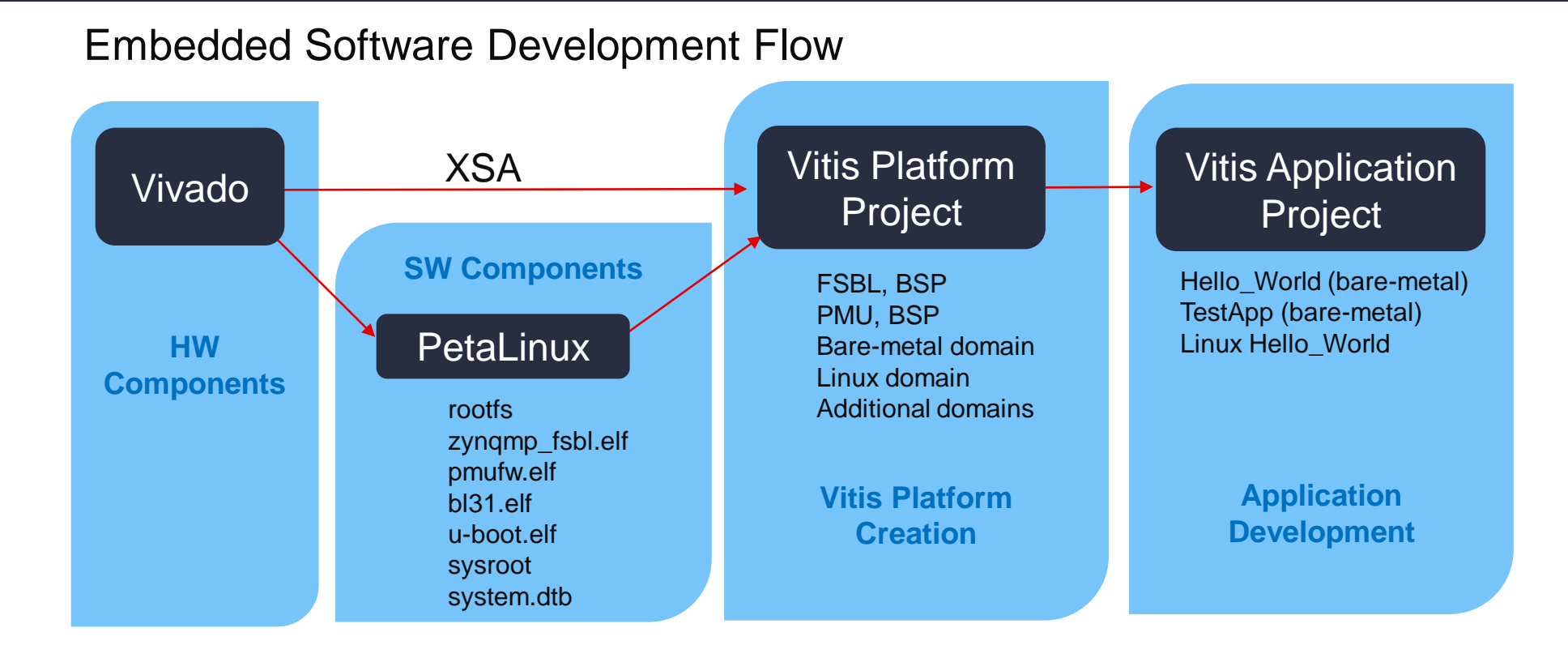
AMD
XILINX

Vivado View

▸ Customizable panels

- A: Project Management
- B: IP Integrator
- C: FPGA Flow
- D: Layout Selection
- E: Project view/Preview Panel
- F: Console, Messages, Logs

AMD XILINX

Embedded System Design using Vivado

1. Launch Vivado
2. Create Block Design

IP Integrator

3. Add PS7
4. Configure PS settings
5. Run Block Automation
6. *Add and configure IPs*
7. *Run Connection Automation*

8. Create Top-Level HDL
9. *Add Constraints (file)*
10. *Generate Bitstream => .bit*
11. Export hardware to VITIS

Vivado

12. Create platform project from XSA
13. Create or add Software Project
14. Build application => .elf

VITIS

15. Program bitstream & .elf into Zynq

PYNQ-Z1

PYNQ-Z2

AMD
XILINX

Vitis Embedded Software Development Flow

Embedded Software Development Flow

Vivado — XSA → **Vitis Platform Project** → **Vitis Application Project**

HW Components

SW Components

PetaLinux

rootfs
zynqmp_fsbl.elf
pmufw.elf
bl31.elf
u-boot.elf
sysroot
system.dtb

FSBL, BSP
PMU, BSP
Bare-metal domain
Linux domain
Additional domains

Vitis Platform Creation

Hello_World (bare-metal)
TestApp (bare-metal)
Linux Hello_World

Application Development

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

# Add IP Integrator Block Diagram

▸ IP Integrator Block Diagram opens a blank canvas

▸ IP can be added from the IP catalog

▸ Drag and drop interface

▸ Intelligent Design environment
  - Design Assistance
  - Connection automation
  - Highlights valid connections
  - Group, create hierarchal blocks

▸ Can import custom IP using IP Packager

AMD XILINX
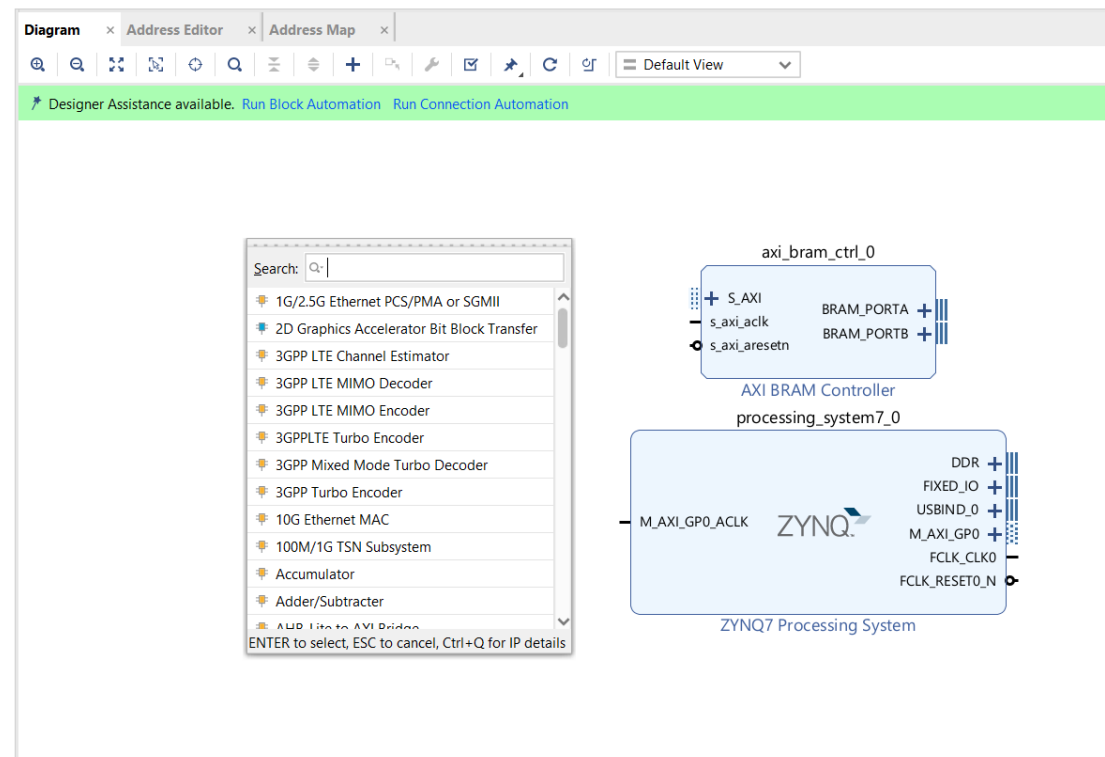
Configuring and Connecting Hardware in IP Integrator

▸ Double click blocks to access configuration options

▸ Drag pointer to make connections
- Highlights valid connections

▸ Connection Automation
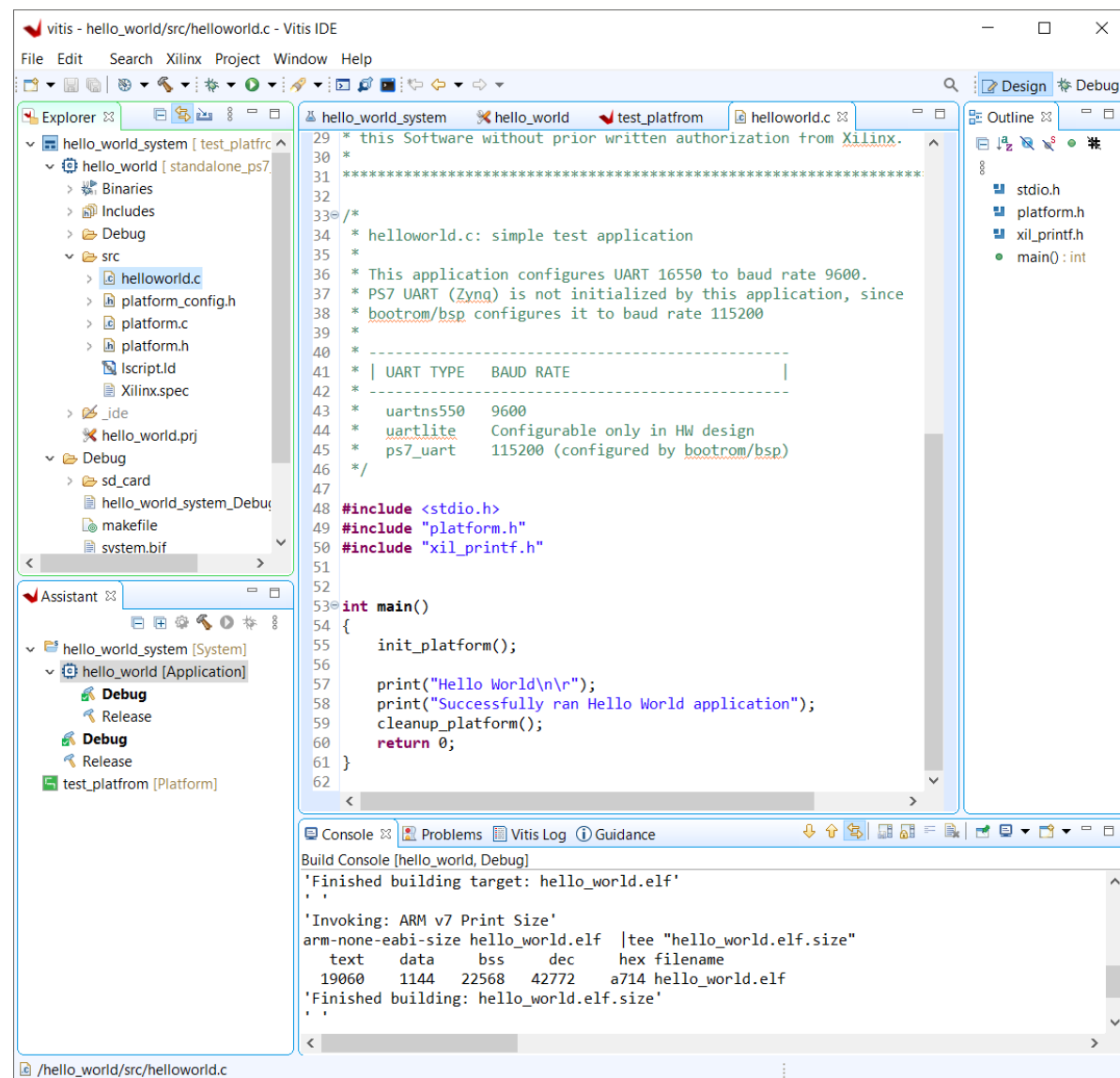- Automatically connect recognised interfaces

▸ Automatically redraw system
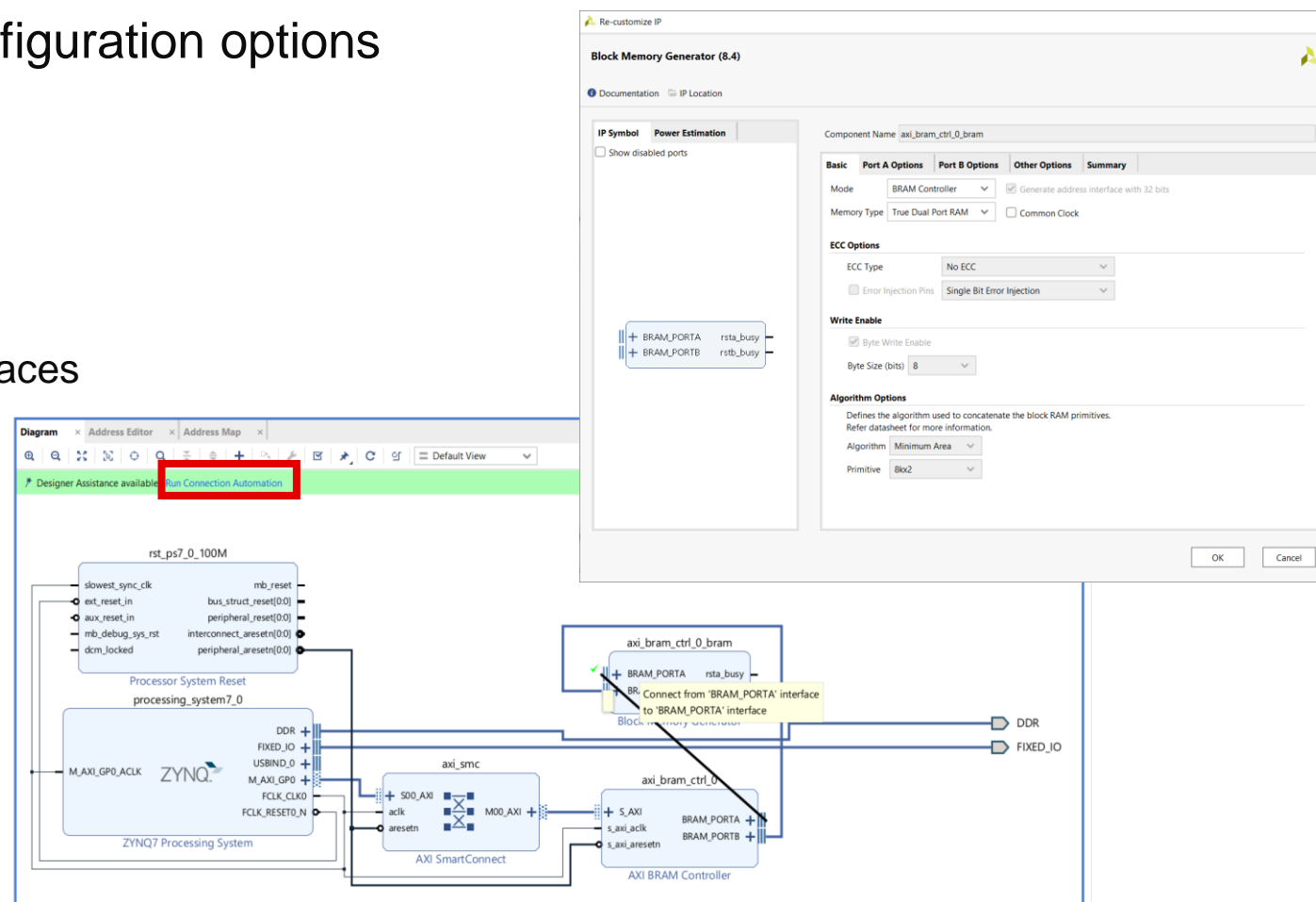
Exporting to VITIS

▸ Export hardware platform first
  - The XSA file has the hardware specifications like processor configuration properties, peripheral connection information, address map, and device initialization code.

  - Include bitstream if generated

▸ Launch VITIS
  - Software development is performed with the Vitis Unified Software Platform

▸ The VITIS tool will then associate user software projects to hardware

©2022 Advanced Micro Devices, Inc.

AMD XILINX
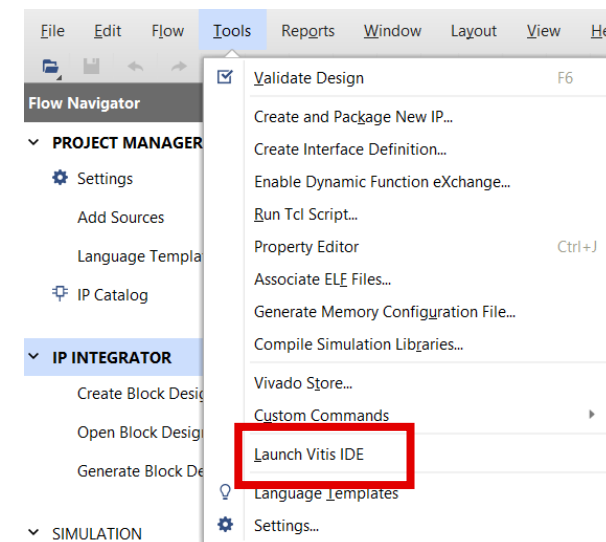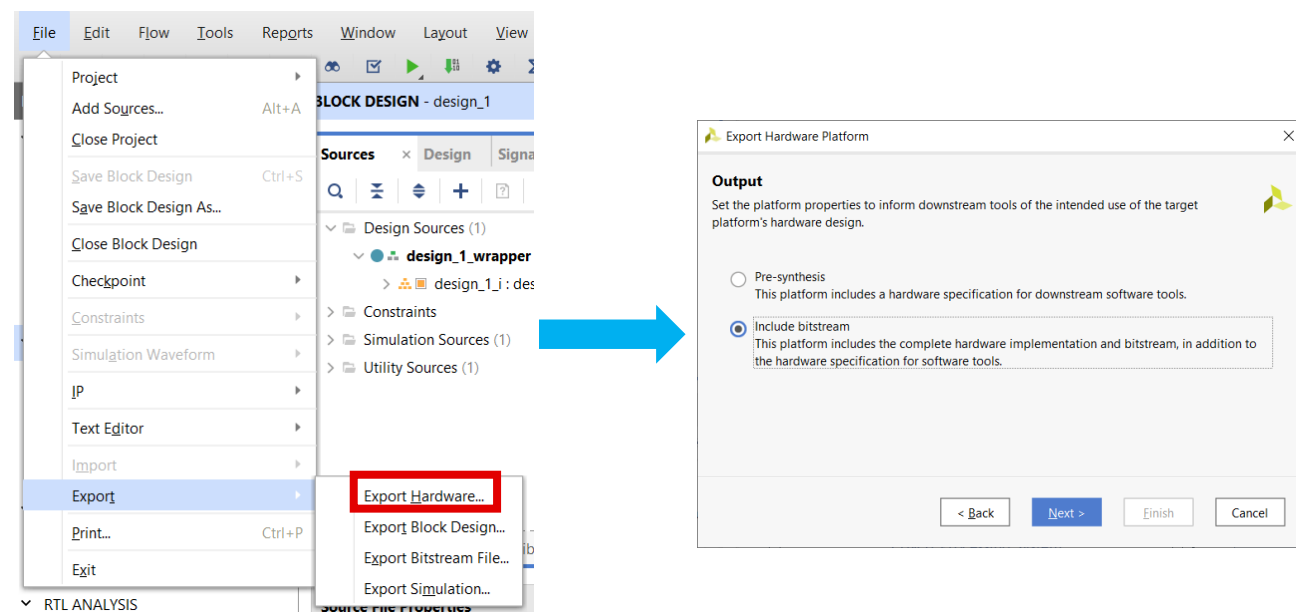
Software Development Flow

▸ Create/Import platform project

  - Import Hardware Spec from XSA to Vitis

  - Domain Creation

▸ Create software application

▸ Update linker script, if needed

▸ Build project

  - compile, assemble, link output file
    <app_project>.elf

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

Creating Hardware Accelerator IP

AMD
XILINX

Port-Level Interfaces

▸ The AXI4 interfaces supported by Vitis HLS include

- The AXI4-Stream (axis)
  - Specify on input arguments or output arguments only, not on input/output arguments

- The AXI4 master (m_axi)
  - Specify on arrays and pointers (and references in C++) only. You can group multiple arguments into the same AXI4 interface using the `bundle` option

- The AXI4-Lite (s_axilite)
  - Specify an AXI4-Lite slave I/O protocol on any type of argument except streams. You can group multiple arguments into the same AXI4-Lite interface using the `bundle` option

```
void example(char *a, char *b, char *c)
{
#pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=a        bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=b        bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=c        bundle=BUS_A offset=0x0400
#pragma HLS INTERFACE ap_vld port=b

    *c += *a + *b;
}
```


Vitis HLS Directive Editor

Directive
INTERFACE

Destination
● Source File
○ Directive File

Options
mode (optional):
bundle (optional):
clock (optional):
depth (optional):
latency (optional):
max_widen_bitwidth (optional):
name (optional):
port (optional):
register (optional):
storage_impl (optional):
storage_type (optional):

ap_ack
ap_fifo
ap_hs
ap_memory
ap_none
ap_ovld
ap_stable
ap_vld
axis
bram
m_axi
s_axilite

Help   Cancel   OK

AMD
XILINX

Interface Modes

▸ Native AXI Interfaces

- AXI4 Slave Lite, AXI4 Master, AXI Stream supported by INTERFACE directive
  - Provided in RTL after Synthesis
  - Supported by C/RTL Co-simulation
  - Supported for Verilog and VHDL

▸ BRAM Memory Interface

- Identical IO protocol to ap_memory
- Bundled differently in IP Integrator
  - Provides easier integration to memories with BRAM interface

Block Level Protocol

AXI4 Interfaces

No IO Protocol

Wire handshake protocols

Memory protocols
: RAM
: FIFO

Argument Type	Scalar		Array			Pointer or Reference			Hls::stream
Interface Mode	Input	Return	I	I/O	O	I	I/O	O	I and O
ap_ctrl_none	3	1	3	3	3	3	3	3	3
ap_ctrl_hs	3	1	3	3	3	3	3	3	3
ap_ctrl_chain	3	D[1]	3	3	3	3	3	3	3
axis	1	3	1	3	1	1	3	1	3
s_axilite	1	1	1	1	1	1	1	1	3
m_axi	3	3	1	1	1	1	1	1	3
ap_none	D[1]	3	3	3	3	D[1]	1	1	3
ap_stable	1	3	3	3	3	1	3	3	3
ap_ack	1	3	3	3	3	1	1	1	3
ap_vld	1	3	3	3	3	1	1	D[1]	3
ap_ovld	3	3	3	3	3	3	D[1]	1	3
ap_hs	1	3	1	3	1	1	1	1	1
ap_memory	3	3	D[1]	D[1]	D[1]	3	3	3	3
bram	3	3	1	1	1	3	3	3	3
ap_fifo	3	3	1	3	1	1	3	1	D[1]

AMD XILINX

Native AXI Slave Lite Interface

▸ Interface Mode: s_axilite

- Supported with INTERFACE directive
- Multiple ports may be grouped into the same Slave Lite interface
    - All ports which use the same bundle name are grouped

▸ Grouped Ports

- Default mode is ap_none for input ports
- Default mode is ap_vld for output ports
- Default mode ap_ctrl_hs for function return
- Default mode can be changed with additional INTERFACE directives

```
void example(char *a, char *b, char *c)
{
#pragma HLS INTERFACE mode=s_axilite port=a bundle=BUS_A
#pragma HLS INTERFACE mode=s_axilite port=b bundle=BUS_A
#pragma HLS INTERFACE mode=s_axilite port=c bundle=OUT
#pragma HLS INTERFACE mode=s_axilite port=return bundle=BUS_A
#pragma HLS INTERFACE mode=ap_vld port=b
  *c += *a + *b;
}
```

AMD
XILINX

Controllable Register Maps in AXI4 Lite

▸ Assigning offset to array (RAM) interfaces

- Specified value is offset to base of array

- Array's address space is always contiguous and linear

```
void hls_sig_gen_bram2axis(hls::stream<axis_last_t<data_t> >& dout,
                data_t sig_buf[MAX_SIG_PERIOD], short sig_period)
{
#pragma HLS INTERFACE port=return     mode=s_axilite bundle=ctrl
#pragma HLS INTERFACE port=sig_buf    mode=s_axilite bundle=ctrl offset=0x1000
#pragma HLS INTERFACE port=sig_period mode=s_axilite bundle=ctrl offset=0x0400
```
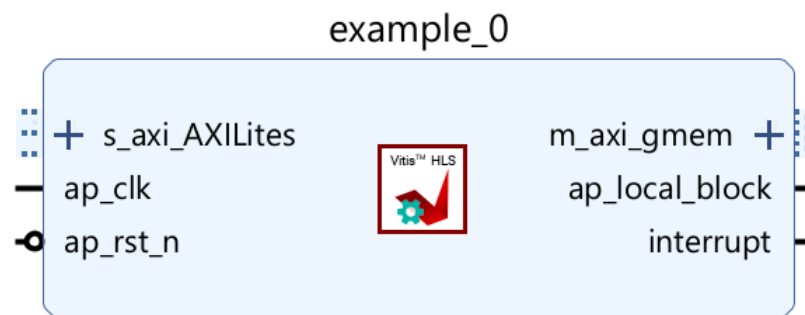
▸ C Driver Files include offset information

- In generated driver file xhls_sig_gen_bram2axis.h

```
...
#define XHLS_SIG_GEN_BRAM2AXIS_CTRL_ADDR_SIG_PERIOD_DATA 0x0400
...
#define XHLS_SIG_GEN_BRAM2AXIS_CTRL_ADDR_SIG_BUF_BASE   0x1000
#define XHLS_SIG_GEN_BRAM2AXIS_CTRL_ADDR_SIG_BUF_HIGH   0x17ff
...
```

AMD
XILINX

Native AXI4 Master

▸ Interface Mode: m_axi

- Supported with INTERFACE directive

▸ Options

- Multiple ports may be grouped into the same AXI4 Master interface
  - All ports which use the same bundle name are grouped
- Depth option is required for C/RTL co-simulation
  - Required for pointers, not arrays
  - Set to the number of values read/written
- Option to support offset or base address

```
void example(volatile int *a)
{

#pragma HLS INTERFACE mode=m_axi depth=50 port=a
```

example_0

s_axi_AXILites   m_axi_gmem

ap_clk   Vitis™ HLS   ap_local_block

ap_rst_n   interrupt

Example (Pre-Production)

Vitis HLS Directive Editor   ×

Directive
INTERFACE

Destination
● Source File
○ Directive File

Options

mode (optional):   m_axi
bundle (optional):
depth (optional):   50
latency (optional):
max_read_burst_length (optional):
max_widen_bitwidth (optional):
max_write_burst_length (optional):
name (optional):
num_read_outstanding (optional):
num_write_outstanding (optional):
port (optional):   a

Help   Cancel   OK

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

Native AXI4 Master : Offset Support

▶ Address Offset / Base Address Support

- Support provided for address offset

▶ Port Offset

- Defines the offset for the port

- May be set on individual interfaces using the INTERFACE directive

▶ Global Offset

- Globally controls the offset ports of all M_AXI interface in the design

- May be set using the interface configuration

  - Using Tcl command config_interface -m_axi_offset option
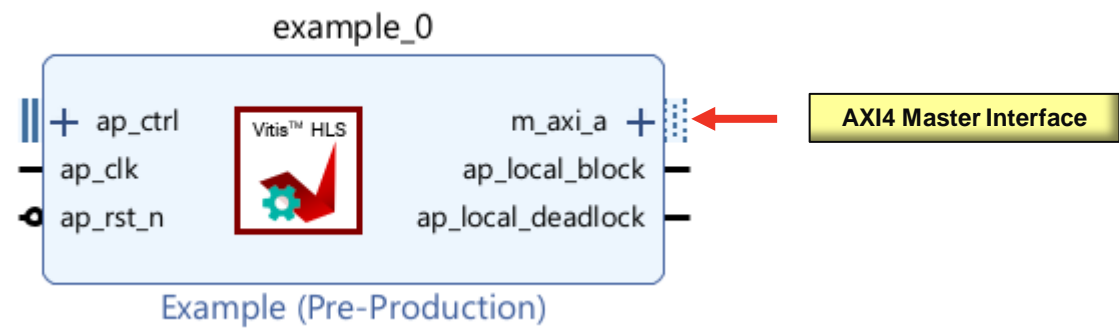
©2022 Advanced Micro Devices, Inc.
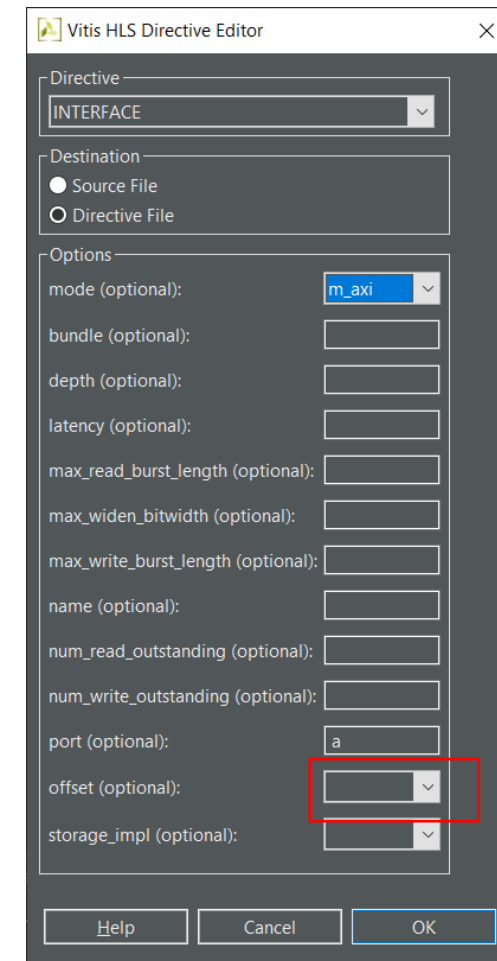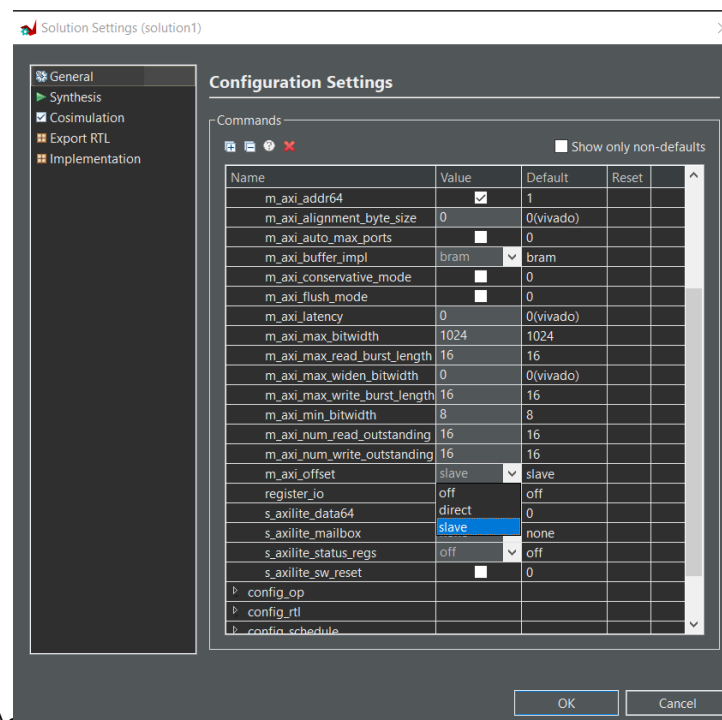
Native AXI4 Master: Offset=off

▶ **AXI4 Master Interface**

- No offset is provided for the address

  - Same as existing behavior

- The offset (BASEADDR) is set IPI

  - Using IP customization GUI

- The offset can <u>not</u> be changed on the fly
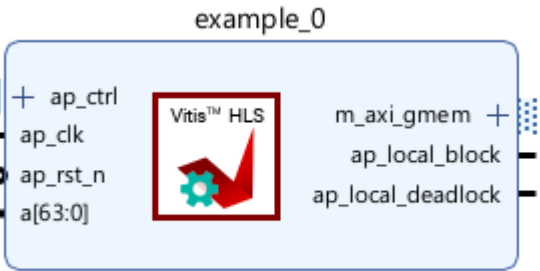
`config_interface -m_axi_offset off`

example_0

Example (Pre-Production)

AXI4 Master Interface

Name	Value	Default	Reset
▷ config_array_partition			
▷ config_compile			
▷ config_dataflow			
▷ config_debug			
▷ config_export			
◢ config_interface			
clock_enable	☐	0	
default_slave_interface	s_axilite ▽	s_axilite	
m_axi_addr64	☑	1	
m_axi_alignment_byte_size	0	0(vivado)	
m_axi_auto_max_ports	☐	0	
m_axi_buffer_impl	bram ▽	bram	
m_axi_conservative_mode	☐	0	
m_axi_flush_mode	☐	0	
m_axi_latency	0	0(vivado)	
m_axi_max_bitwidth	1024	1024	
m_axi_max_read_burst_length	16	16	
m_axi_max_widen_bitwidth	0	0(vivado)	
m_axi_max_write_burst_length	16	16	
m_axi_min_bitwidth	8	8	
m_axi_num_read_outstanding	16	16	
m_axi_num_write_outstanding	16	16	
m_axi_offset	off ▽	slave	✖
register_io	off ▽	off	
s_axilite_data64	☐	0	
s_axilite_mailbox	none ▽	none	
s_axilite_status_regs	off ▽	off	
s_axilite_sw_reset	☐	0	
▷ config_op			
▷ config_rtl			
▷ config_schedule			
▷ config_storage			
▷ config_unroll			

AMD XILINX

Native AXI4 Master: Offset=direct

▶ Direct Interface

- Generates a scalar input offset port

- The offset is set by driving the input port

- It can be changed on the fly by driving the port with a different value

```
config_interface -m_axi_offset direct
```


...Plus port to specify address offset

AXI4 Master Interface ...

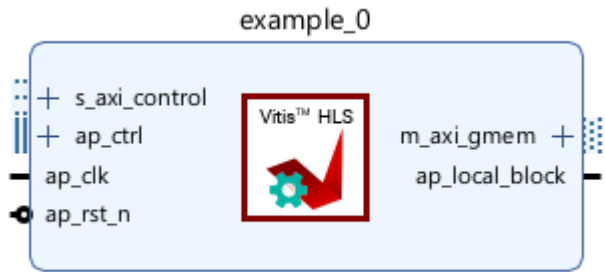| Name	Value	Default	
▷ config_array_partition			
▷ config_compile			
▷ config_dataflow			
▷ config_debug			
▷ config_export			
▼ config_interface			
clock_enable	☐	0	
default_slave_interface	s_axilite ▾	s_axilite	
m_axi_addr64	☑	1	
m_axi_alignment_byte_size	0	0(vivado)	
m_axi_auto_max_ports	☐	0	
m_axi_buffer_impl	bram ▾	bram	
m_axi_conservative_mode	☐	0	
m_axi_flush_mode	☐	0	
m_axi_latency	0	0(vivado)	
m_axi_max_bitwidth	1024	1024	
m_axi_max_read_burst_length	16	16	
m_axi_max_widen_bitwidth	0	0(vivado)	
m_axi_max_write_burst_length	16	16	
m_axi_min_bitwidth	8	8	
m_axi_num_read_outstanding	16	16	
m_axi_num_write_outstanding	16	16	
m_axi_offset	direct ▾	slave	
register_io	off ▾	off	
s_axilite_data64	☐	0	
s_axilite_mailbox	none ▾	none	
s_axilite_status_regs	off ▾	off	
s_axilite_sw_reset	☐	0	
▷ config_op			
▷ config_rtl			
▷ config_schedule			
▷ config_storage			
▷ config_unroll			

AMD
XILINX

Native AXI4 Master: Offset=slave(Default)

▸ Direct Interface

- - Generates an offset port and automatically maps it to an AXI4 Slave Lite interface

- - User must program the offset before starting transactions on the AXI4 Master interface

- - It can changed on the fly by re-programming the offset register

`config_interface -m_axi_offset slave`

example_0

+ s_axi_control

+ ap_ctrl

ap_clk   Vitis™ HLS   m_axi_gmem +

                               ap_local_block

ap_rst_n

Example (Pre-Production)

AXI4 Master Interface …

…Plus AXI4 Slave Lite interface to program the address offset

Name	Value	Default
▷ config_array_partition		
▷ config_compile		
▷ config_dataflow		
▷ config_debug		
▷ config_export		
◢ config_interface		
clock_enable	☐	0
default_slave_interface	s_axilite	s_axilite
m_axi_addr64	☑	1
m_axi_alignment_byte_size	0	0(vivado)
m_axi_auto_max_ports	☐	0
m_axi_buffer_impl	bram	bram
m_axi_conservative_mode	☐	0
m_axi_flush_mode	☐	0
m_axi_latency	0	0(vivado)
m_axi_max_bitwidth	1024	1024
m_axi_max_read_burst_length	16	16
m_axi_max_widen_bitwidth	0	0(vivado)
m_axi_max_write_burst_length	16	16
m_axi_min_bitwidth	8	8
m_axi_num_read_outstanding	16	16
m_axi_num_write_outstanding	16	16
m_axi_offset	slave	slave
register_io	off	off
s_axilite_data64	☐	0
s_axilite_mailbox	none	none
s_axilite_status_regs	off	off
s_axilite_sw_reset	☐	0
▷ config_op		
▷ config_rtl		
▷ config_schedule		
▷ config_storage		
▷ config_unroll		

XILINX

AXI4-Master : Burst Access with a Single Port

▸ Burst access with single port

- Loop must be pipelined

- Address must be accessed in increasing order

- Memory accesses cannot be guarded by a conditional statement

- Do not flatten nested loops

- Only one read and write per AXI port allowed in a for loop

- Only one read and one write is allowed in a for loop unless the ports are bundled in different AXI ports

```
//Port a is assigned to an AXI4 master interface
void example(volatile int *a){
#pragma HLS INTERFACE mode=m_axi port=a depth=50
#pragma HLS INTERFACE mode= s_axilite port=return

int i;
int buff[50];

//Burst data into the Design
//memcpy creates a burst access to memory
memcpy(buff,(const int*)a,50*sizeof(int));

//Perform some calculation
for(i=0;i<50;i++){
    buff[i]=buff[i]+100;
}

//Burst Dara out of the design
//Alternatively, for loop creates a burst access to memory
for(i=0;i<50;i++){
#pragma HLS PIPELINE
    a[i]=buff[i];
}
}
```

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

AXI4-Master : Burst Access with Multiple Ports

▶ Burst access with multiple ports

- Vitis HLS implements the port reads as burst transfers

- Port a is specified without using the bundle option and is implemented in the default AXI interface

- Port b is specified using a named bundle and is implemented in a separate AXI interface called d2_port

- In the multiple access bursts

  - Only one access(read or write) per port can be inferred from a for loop

  - No simultaneous read and write access

  - If multiple ports are used, multiple reads or writes can be performed

```
//Two pointers are accessed
void example(volatile int *a, int *b){
#pragma HLS INTERFACE mode= s_axilite port=return

//Two different AXI ports are used
#pragma HLS INTERFACE mode=m_axi port=a depth=50
#pragma HLS INTERFACE mode=m_axi port=b depth=50 bundle=d2_port

int i;
int buff[50];

//Copy data in
for(i=0;i<50;i++){
#pragma HLS PIPELINE
    //separate AXI ports mean both a and b reads implemented as bursts
    buff[i]=a[i]+b[i];
}
…
}
```

AMD
XILINX

AXI4 Port Bundling

▸ AXI4 Master and Lite Port Bundling

- The bundle options groups arguments into the same AXI4 port

- For example, group 3 arguments into AXI4-Lite port "ctrl" :

```
void hls_sig_gen_bram2axis(hls::stream<data_t>& dout,
      data_t sig_buf[MAX_SIG_PERIOD], short sig_period)
{
#pragma HLS INTERFACE port=return     mode=s_axilite bundle=ctrl
#pragma HLS INTERFACE port=sig_buf    mode=s_axilite bundle=ctrl
#pragma HLS INTERFACE port=sig_period mode=s_axilite bundle=ctrl
#pragma HLS INTERFACE port=dout mode=axis
```

▸ Arguments can be Bundled into AXI4 Master and AXI4 Lite ports

- If no bundle name is used a default name is used for all arguments

  - All go into a single AXI4 Master or AXI4 Lite

  - Default name applied if no –bundle option is used

- Group different sized variables into an AXI4 Master port

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

AXI4 Stream Interface: Ease of Use

> **Native Support for AXI4 Stream Interfaces**
>> Native = An AXI4 Stream can be specified with set_directive_interface
   – No longer required to set the interface then add a resource
   – This AXI4 Stream interface is part of the HDL after synthesis
   – This AXI4 Stream interface is simulated by RTL co-simulation

Interface Type "axis" is AXI4 Stream

set_directive_interface –mode axis "foo" portA
Or
#pragma HLS interface mode=axis port=portA

©2022 Advanced Micro Devices, Inc.

AMD
XILINX
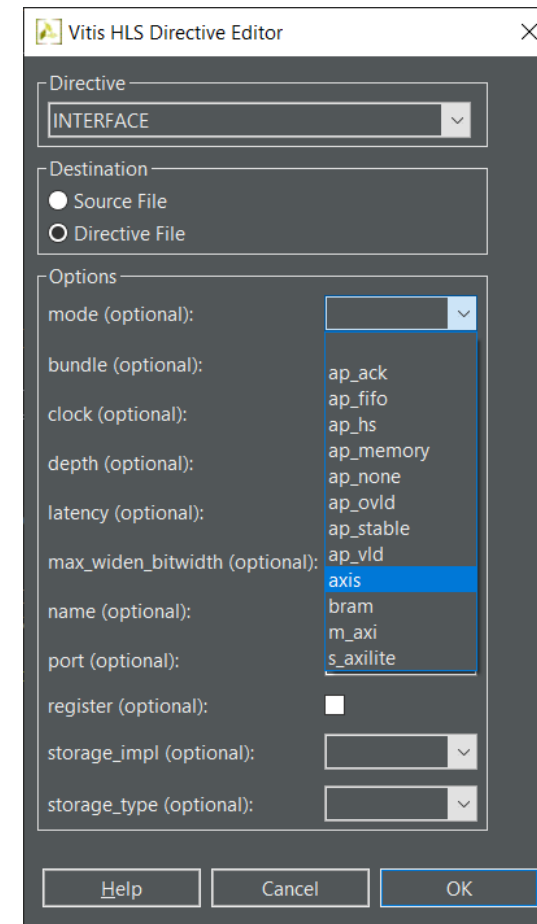
Generate the Hardware Accelerator

▸ Select Solution > Export RTL

▸ Select Vivado IP(.zip), Vivado IP for System Generator for Vivado

▸ Select Output Location

▸ Change the IP Configuration
  - Vendor/Library/Version/Description/Display Name/Taxonomy

▸ Click on OK
  - The IP is exported as a ZIP file that can be added to the Vivado IP catalog.
  - The impl/ip folder also contains the contents of the unzipped IP.

©2022 Advanced Micro Devices, Inc.

AMD
XILINX
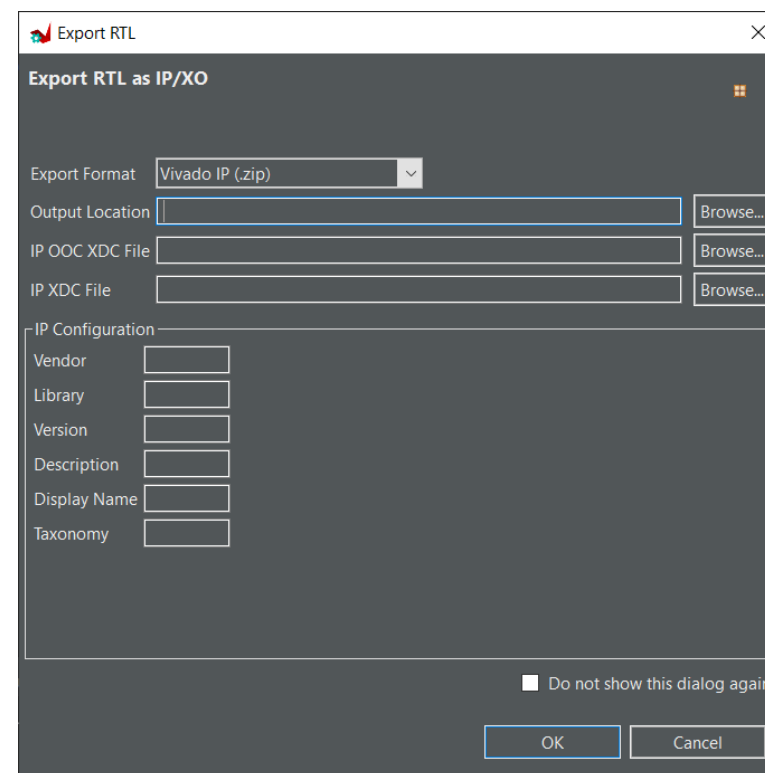
Generated Impl Directory

```
▲  📁 impl
   ▷  📁 ip
   ▷  📁 misc
   ▷  📁 verilog
   ▷  📁 vhdl
```

```
▲  📁 impl
   ▲  📁 ip
         📄 auxiliary.xml
         📄 component.xml
         📄 example_info.xml
         📄 pack.bat
         🔆 run_ippack.tcl
         📄 sysgen_info.xml
         📄 vivado.jou
         📄 vivado.log
         📄 xup_hls_example_1_0.zip
      ▷  ❄ constraints
      ▷  📁 doc
      ▷  📁 drivers
      ▷  📁 example
      ▷  📁 hdl
      ▷  📁 misc
         📁 subcore
      ▷  📁 xgui
   ▷  📁 misc
   ▷  📁 verilog
   ▷  📁 vhdl
```

Point IP Catalog to point to the ip directory

IP Integrator will use this file

VITIS will use this directory

```
▲  📁 drivers
   ▲  📁 example_v1_0
      ▲  📁 data
            📄 example.mdd
            🔆 example.tcl
      ▲  📁 src
            🖿 Makefile
            🄲 xexample_hw.h
            🄲 xexample_linux.c
            🄲 xexample_sinit.c
            🄲 xexample.c
            🄲 xexample.h
```

Header file

```
▲  📁 impl
   ▷  📁 ip
   ▷  📁 misc
   ▲  📁 verilog
         example_AXILites_s_axi.v
         example_buff_RAM_AUTO_1R1W.v
         example_example_Pipeline_1.v
         example_example_Pipeline_2.v
         example_flow_control_loop_pipe_sequential_init.
         example_gmem_m_axi.v
         example.v
         memcpy_top_AXILites_s_axi.v
         memcpy_top_buff_RAM_AUTO_1R1W.v
         memcpy_top_flow_control_loop_pipe_sequential_
         memcpy_top_gmem_m_axi.v
         memcpy_top_memcpy_top_Pipeline_1.v
         memcpy_top_memcpy_top_Pipeline_2.v
         memcpy_top.v
   ▷  📁 vhdl
```

Generated Verilog RTL Files

©2022 Advanced Micro Devices, Inc.

AMD
XILINX

Integrating the Hardware Accelerator IP in Embedded System

AMD
XILINX

Embedded System Design using Vivado

▸ Create a new Vivado project, or open an existing project

▸ Invoke IP Integrator

▸ Construct(modify) the hardware portion of the embedded design by adding the IP-XACT hardware accelerator created in Vitis HLS

▸ Create (Update) top level HDL wrapper

▸ Synthesize any non-embedded components and implement in Vivado

▸ Export the hardware description(XSA File), and launch VITIS

▸ Create a new platform project from XSA and application projects in the VITIS

▸ Compile the software with the GNU cross-compiler in VITIS

▸ Download the programmable logic's completed bitstream using Xilinx Tools > Program FPGA in VITIS

▸ Use VITIS to download and execute the program (the ELF file)

AMD
XILINX

Summary

AMD
XILINX

Summary

▶ Embedded system development flow in FPGA involves

- Developing hardware using IP Integrator and Vivado
- Developing software using VITIS

▶ hardware accelerator provides wide support of AXI interfaces, System Generator design, and design check point(dcp)

- Use the INTERFACE directive
- The choice of hardware accelerator is a function of the C variable type (pointer, etc.)

▶ Start with the correct C argument type

- Verify the design at the C level
- Accept the default block-level I/O protocol
- Select the port-level I/O protocol that gives the required hardware accelerator interface
- Optionally group ports

AMD
XILINX

Thank You

Disclaimer and Attribution

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© Copyright 2021 Advanced Micro Devices, Inc. All rights reserved. Xilinx, the Xilinx logo, AMD, the AMD Arrow logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

AMD
XILINX