# 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

## Outline

- Embedded System Design in Zynq using IP Integrator
- Creating Hardware Accelerator IP
- Integrating the Hardware Accelerator IP in Embedded System
- Summary

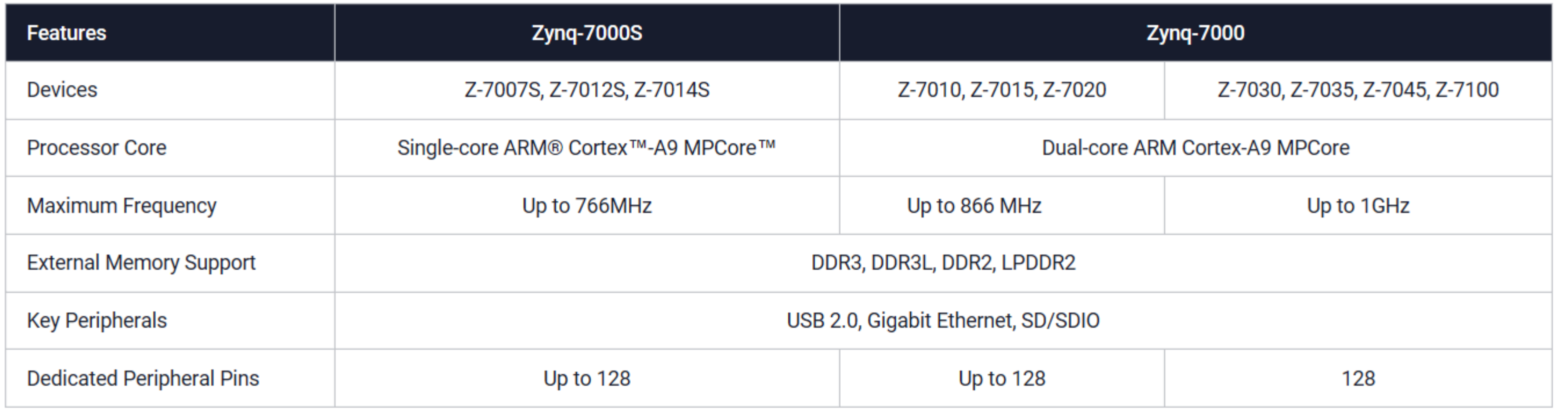
# Embedded System Design in Zynq using IP Integrator

## 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)

## 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



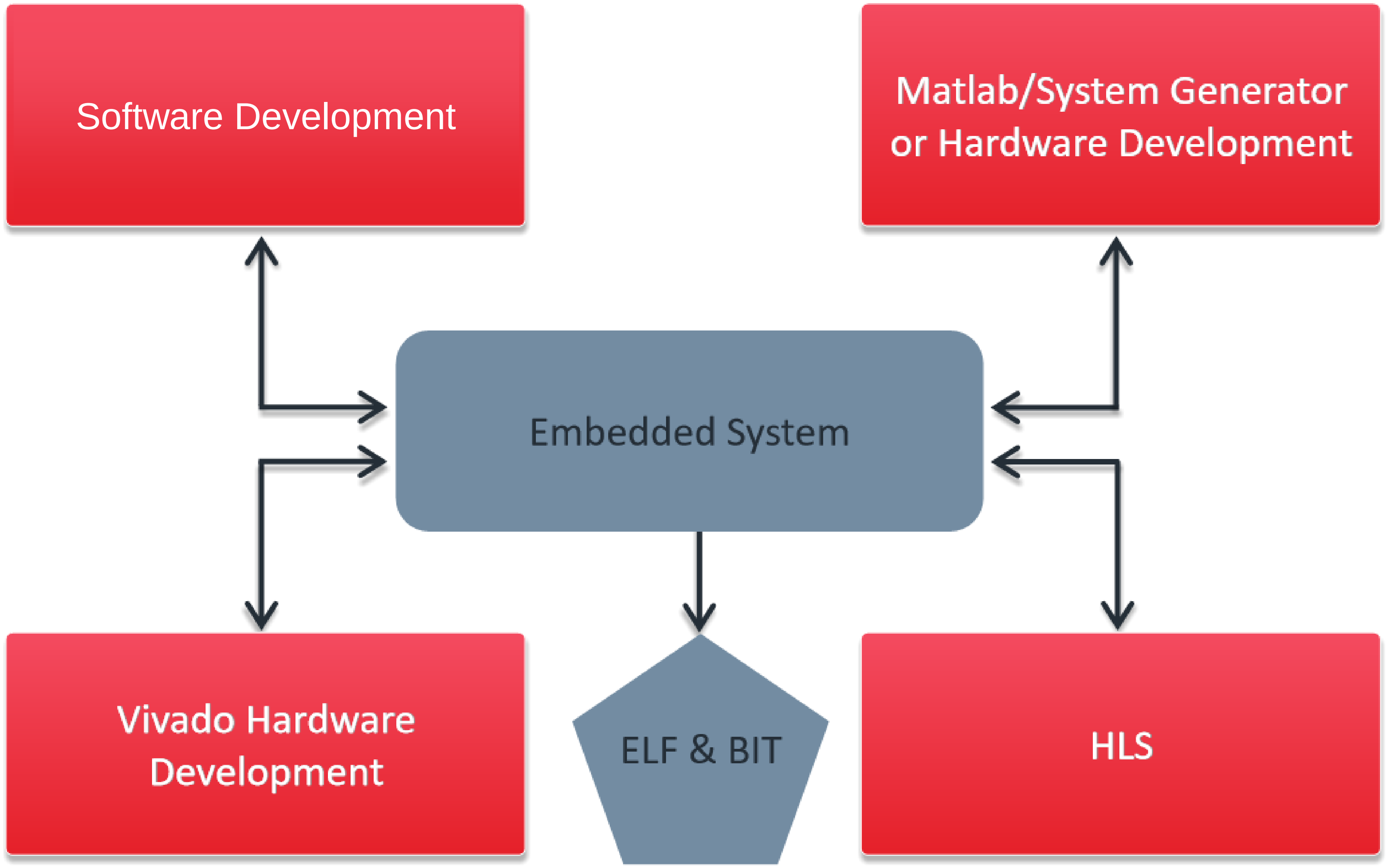
| Features | Zynq-7000S | Zynq-7000 | |
|---|---|---|---|
| Devices | Z-7007S, Z-7012S, Z-7014S | Z-7010, Z-7015, Z-7020 | Z-7030, Z-7035, Z-7045, Z-7100 |
| Processor Core | Single-core ARM® Cortex™-A9 MPCore™ | Dual-core ARM Cortex-A9 MPCore | |
| Maximum Frequency | Up to 766MHz | Up to 866 MHz | 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 |

## Vivado

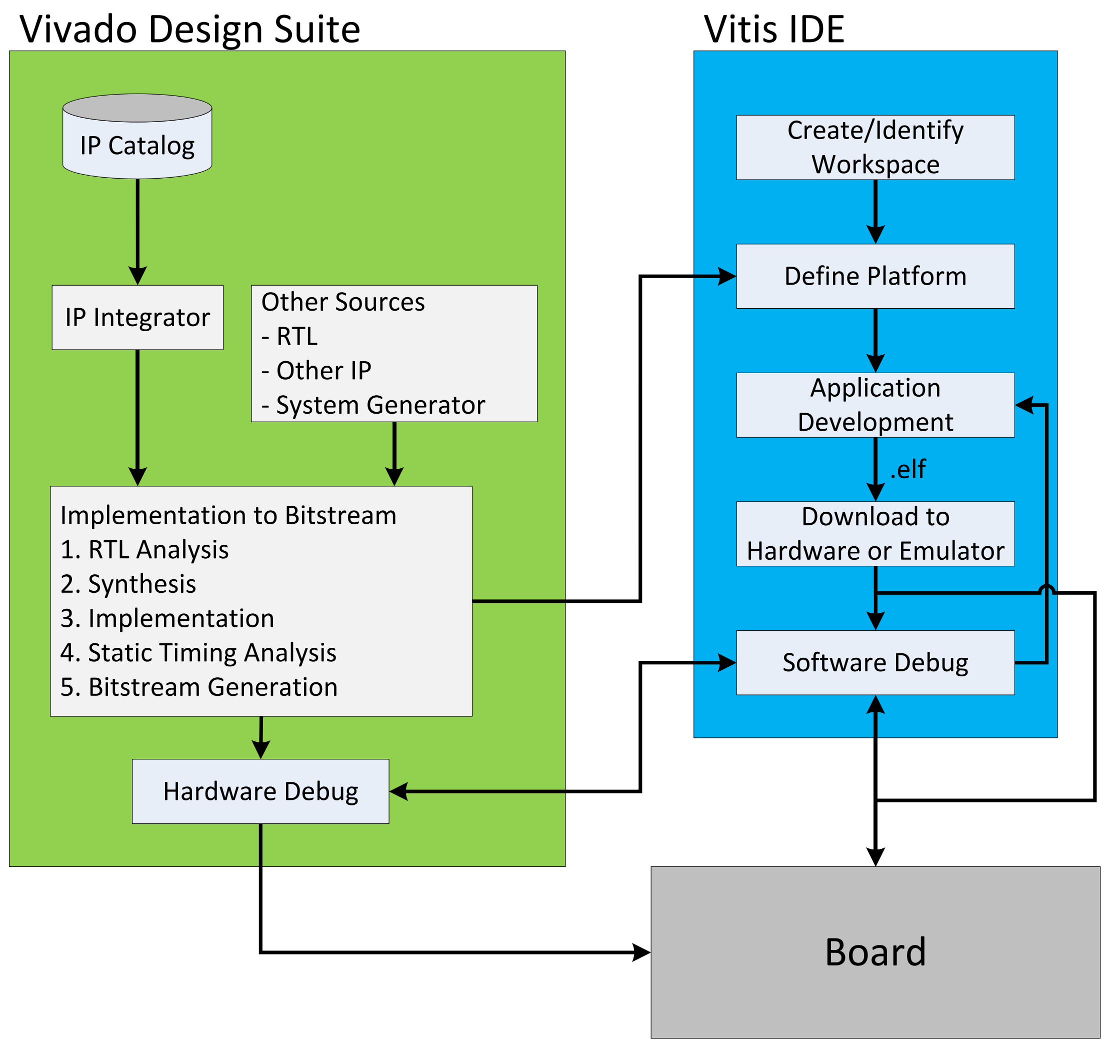
- What are Vivado, IP Integrator and Vitis?
  - Vivado is the tool suite for Xilinx FPGA design and includes capability for embedded system design
    - IP Integrator, is part of Vivado and allows block level design of the hardware part of an Embedded system
    - Vivado includes all the tools, IP, and 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

[Public]

## Introduction to Embedded System Development


Software Development
Matlab/System Generator or Hardware Development
Embedded System
Vivado Hardware Development
ELF & BIT
HLS

## Embedded System Design Flow for Zynq-7000 SoC



## Embedded System Tools: Hardware

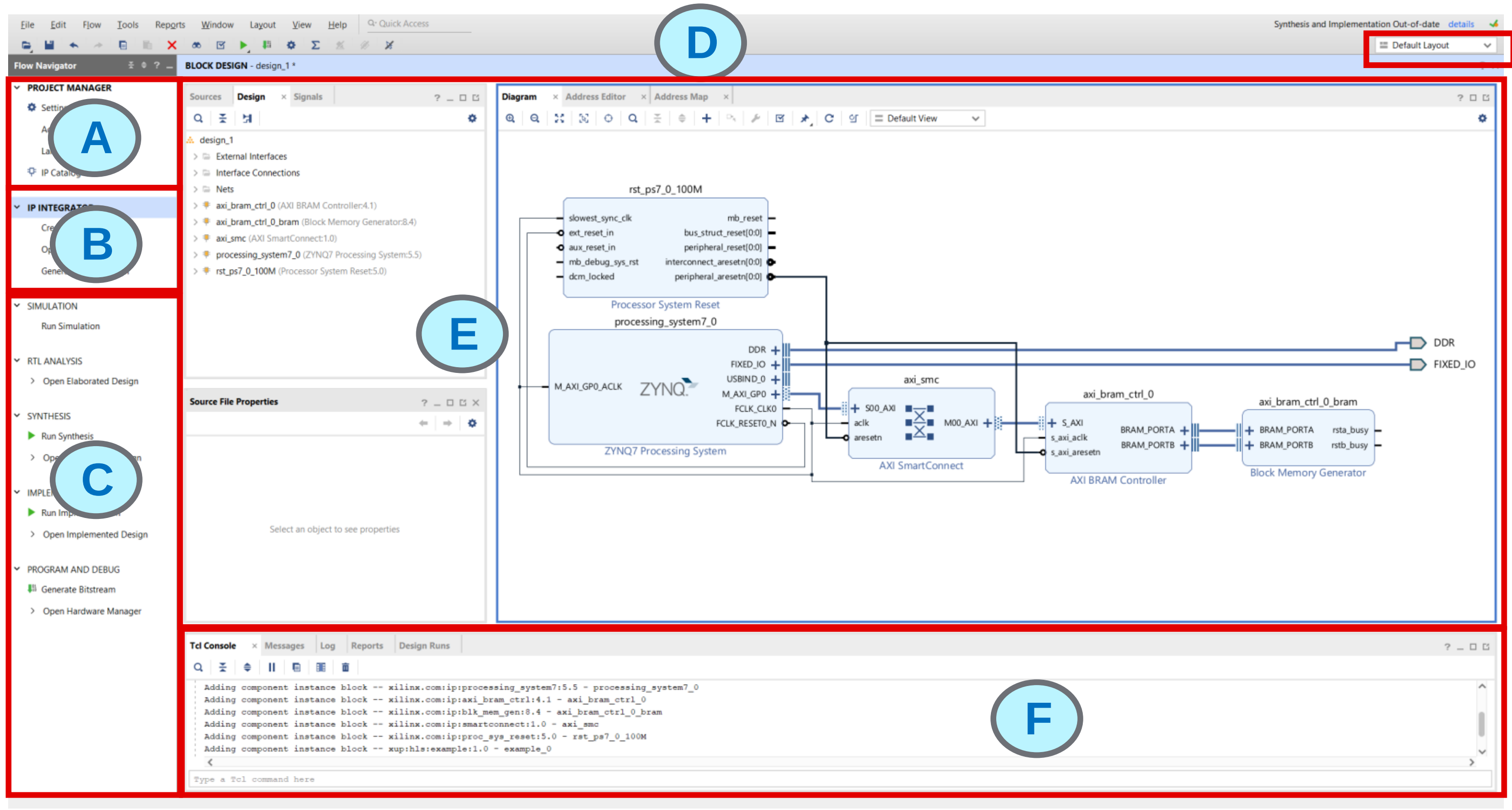
- Hardware development tools
  - IP Integrator
  - IP Packager
  - Hardware netlist generation
  - Simulation model generation
  - Hardware debugging using Vivado analyzer cores

## 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

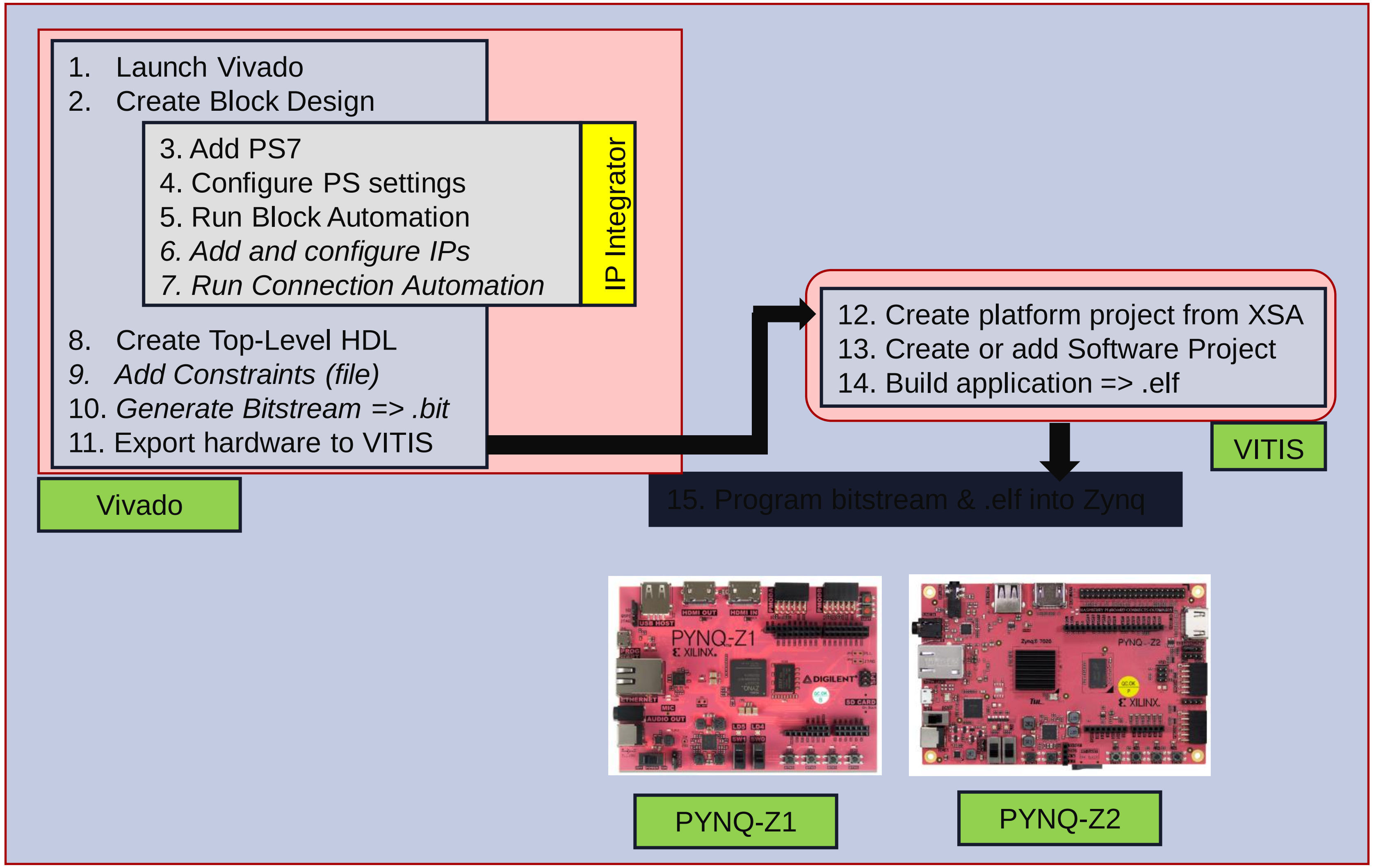
# Vivado View

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



```
Adding component instance block -- xilinx.com:ip:processing_system7:5.5 - processing_system7_0
Adding component instance block -- xilinx.com:ip:axi_bram_ctrl:4.1 - axi_bram_ctrl_0
Adding component instance block -- xilinx.com:ip:blk_mem_gen:8.4 - axi_bram_ctrl_0_bram
Adding component instance block -- xilinx.com:ip:smartconnect:1.0 - axi_smc
Adding component instance block -- xilinx.com:ip:proc_sys_reset:5.0 - rst_ps7_0_100M
Adding component instance block -- xup:hls:example:1.0 - example_0
Type a Tcl command here
```

# Embedded System Design using Vivado



1. Launch Vivado
2. Create Block Design

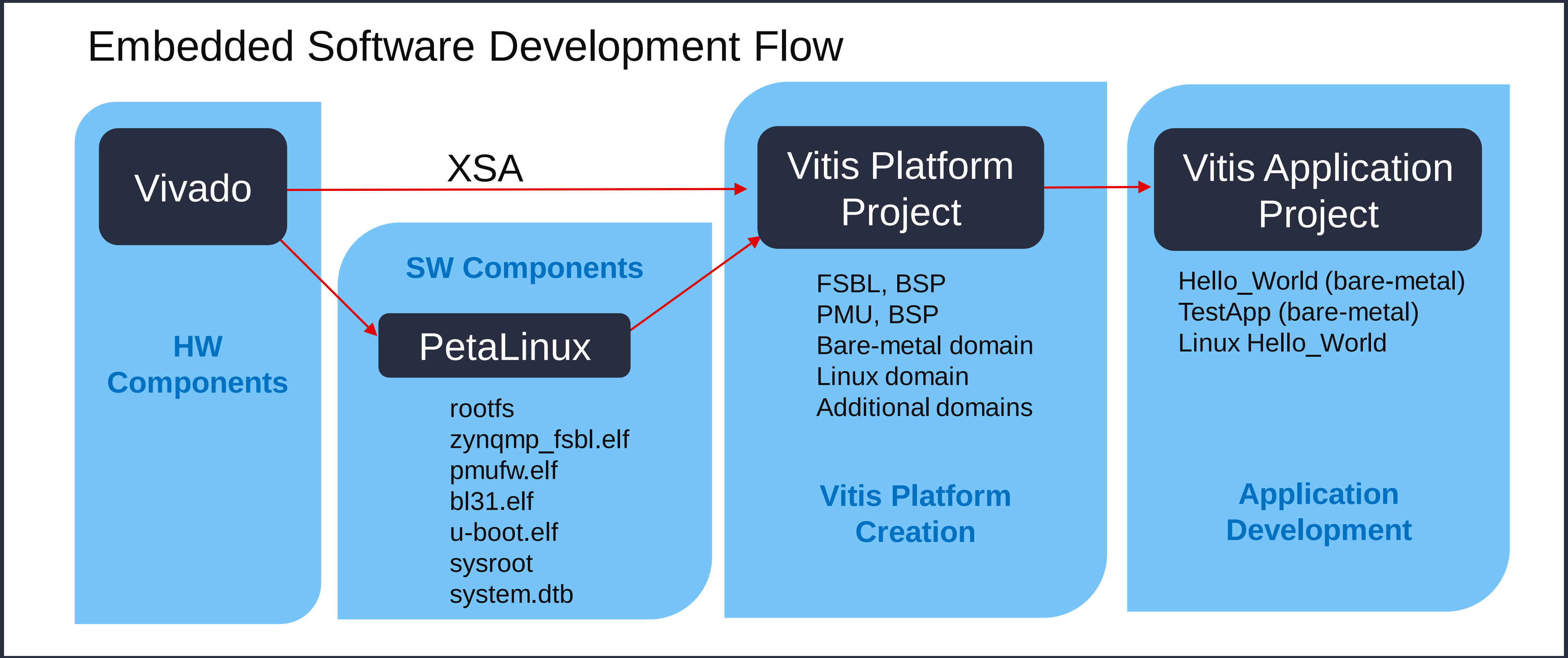
3. Add PS7
4. Configure PS settings
5. Run Block Automation
*6. Add and configure IPs*
*7. Run Connection Automation*

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

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

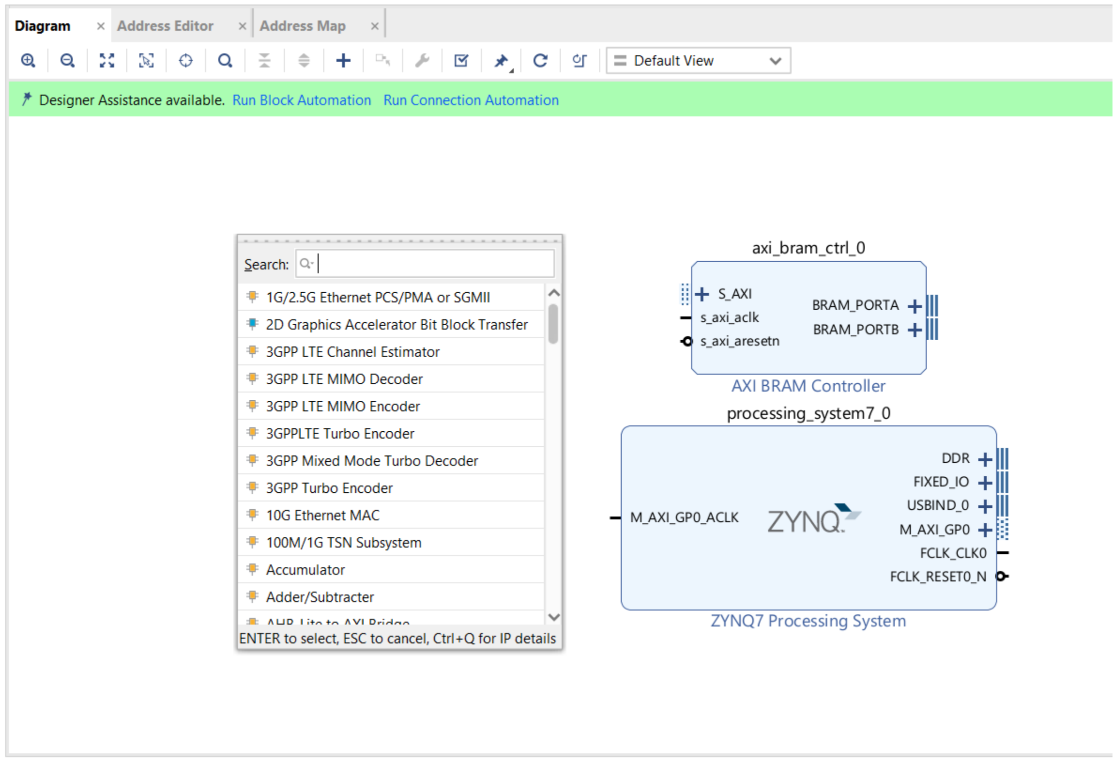
15. Program bitstream & .elf into Zynq

## Vitis Embedded Software Development Flow



# 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



Designer Assistance available.

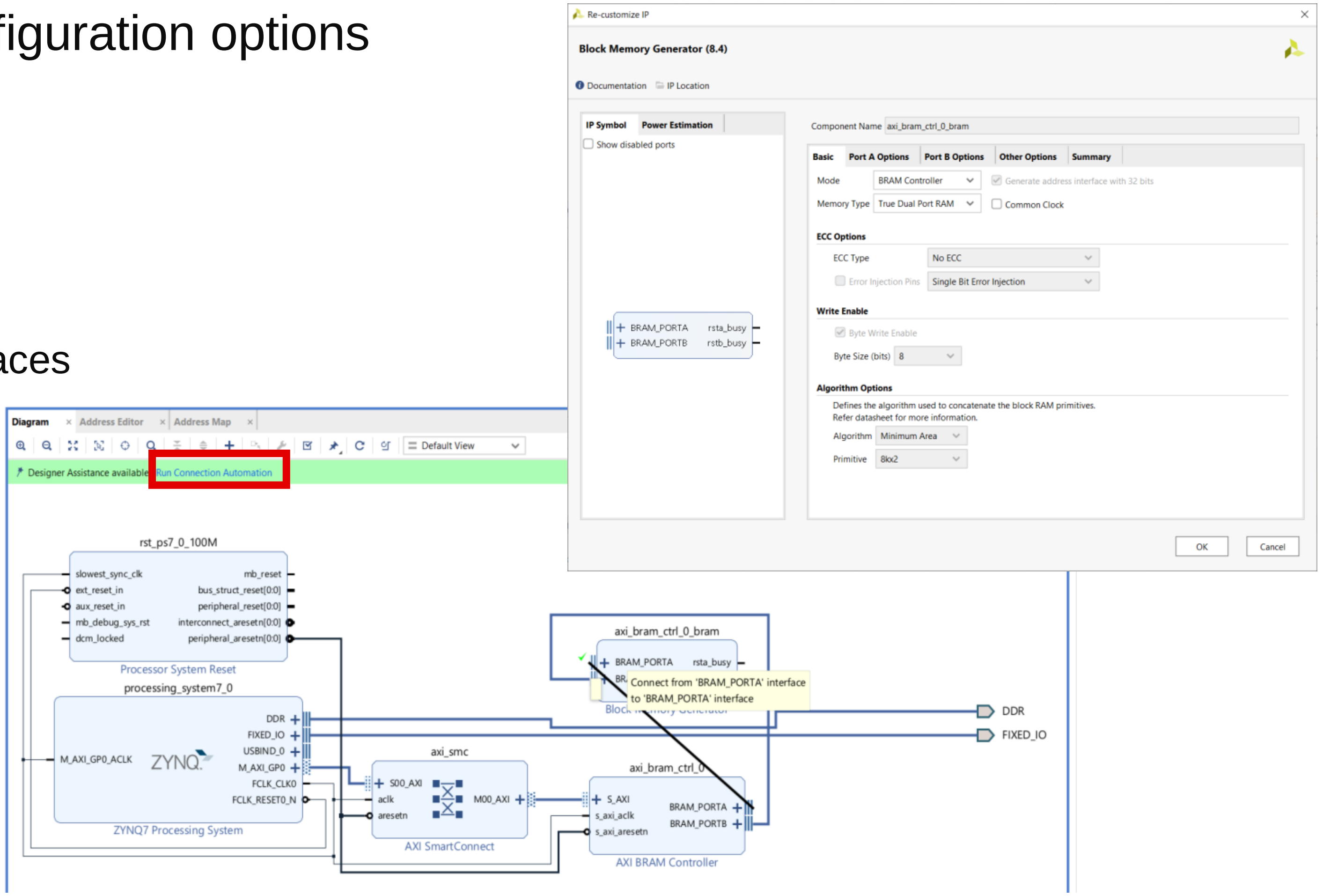
Search:

1G/2.5G Ethernet PCS/PMA or SGMII
2D Graphics Accelerator Bit Block Transfer
3GPP LTE Channel Estimator
3GPP LTE MIMO Decoder
3GPP LTE MIMO Encoder
3GPPLTE Turbo Encoder
3GPP Mixed Mode Turbo Decoder
3GPP Turbo Encoder
10G Ethernet MAC
100M/1G TSN Subsystem
Accumulator
Adder/Subtracter
AHB-Lite to AXI Bridge

ENTER to select, ESC to cancel, Ctrl+Q for IP details

## 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



Re-customize IP

Block Memory Generator (8.4)

Component Name: axi_bram_ctrl_0_bram

Basic tab:

- Mode: BRAM Controller
- [✓] Generate address interface with 32 bits
- Memory Type: True Dual Port RAM
- [ ] Common Clock

ECC Options

- ECC Type: No ECC
- [ ] Error Injection Pins: Single Bit Error Injection

Write Enable

- [✓] Byte Write Enable
- Byte Size (bits): 8

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.

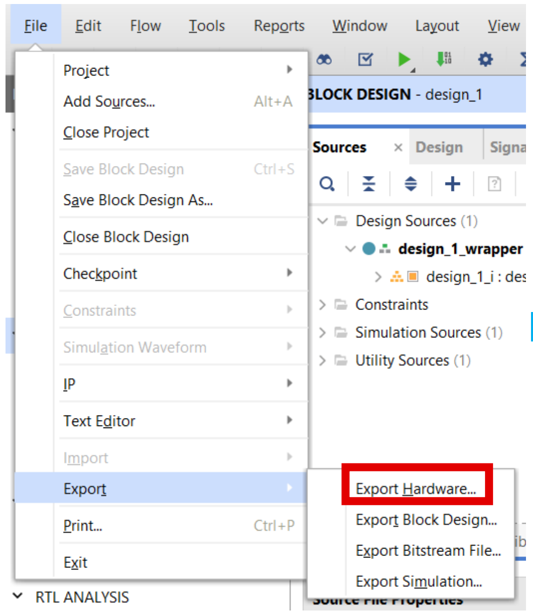
- Algorithm: Minimum Area
- Primitive: 8kx2

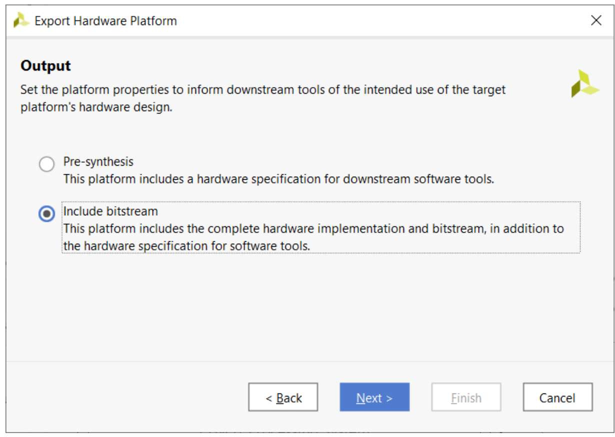
Designer Assistance available. Run Connection Automation

Connect from 'BRAM_PORTA' interface to 'BRAM_PORTA' interface

# 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



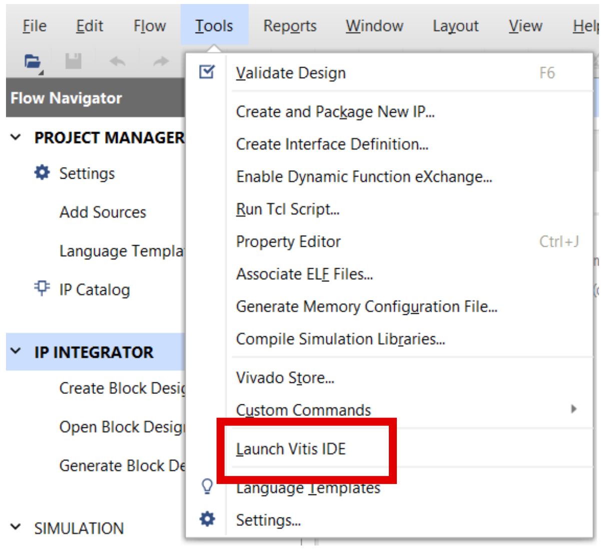

Export Hardware Platform

**Output**

Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

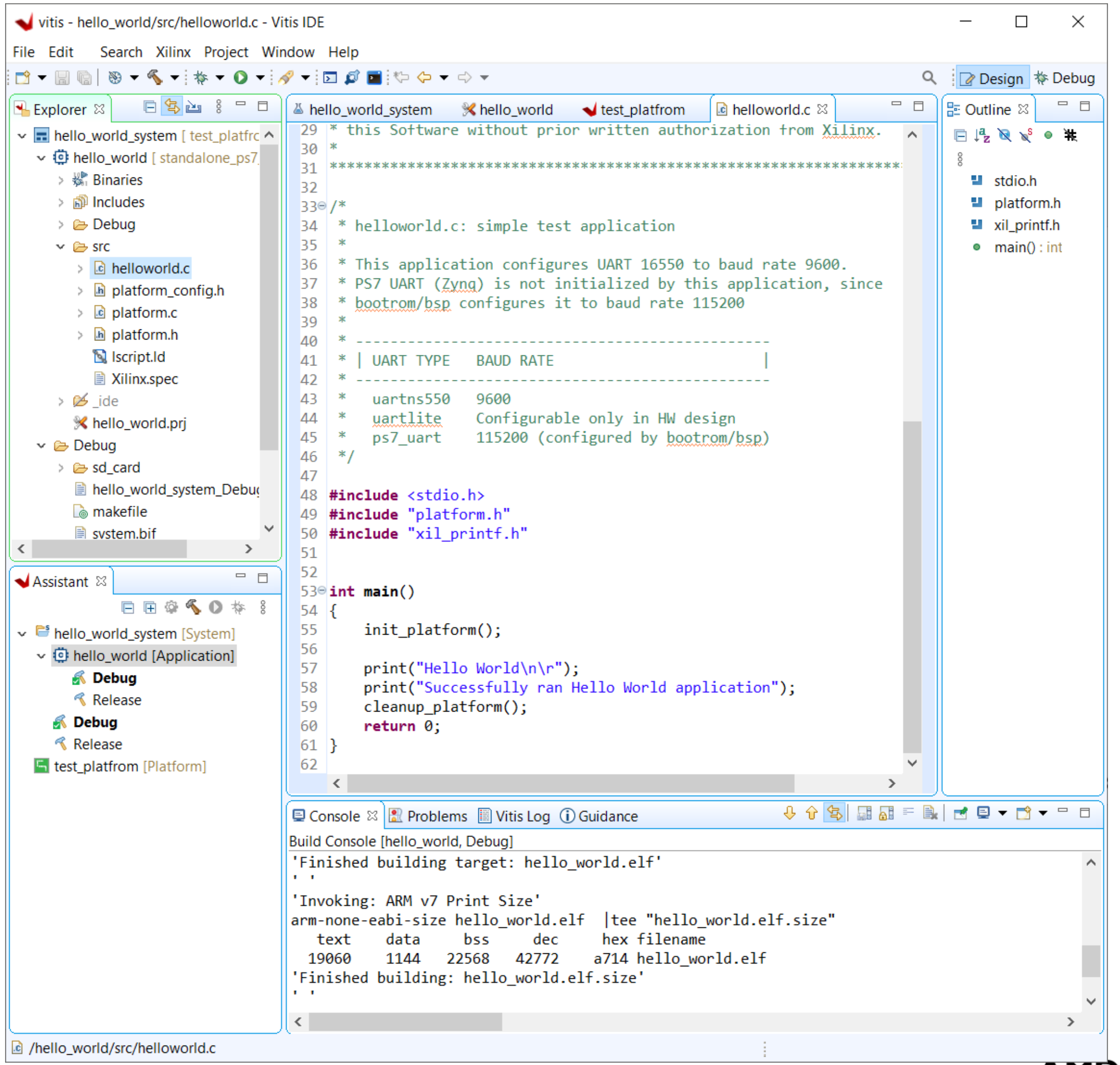
[ ] Pre-synthesis
This platform includes a hardware specification for downstream software tools.

[✓ Include bitstream]
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.



# 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*



```c
* this Software without prior written authorization from Xilinx.
*
*******************************************************************

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

Build Console [hello_world, Debug]

```
'Finished building target: hello_world.elf'
' '
'Invoking: ARM v7 Print Size'
arm-none-eabi-size hello_world.elf  |tee "hello_world.elf.size"
   text    data     bss     dec     hex filename
  19060    1144   22568   42772    a714 hello_world.elf
'Finished building: hello_world.elf.size'
' '
```

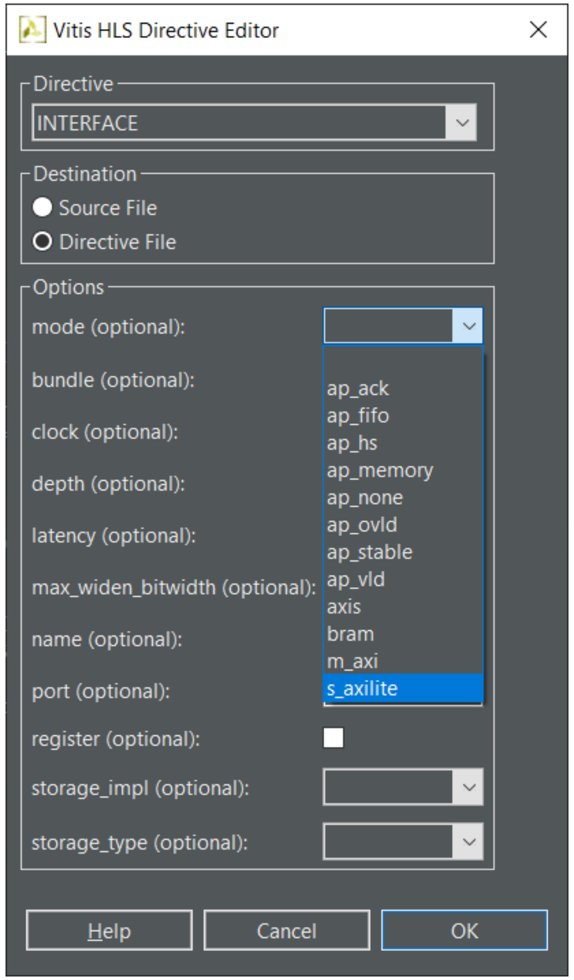
## Creating Hardware Accelerator IP

# 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

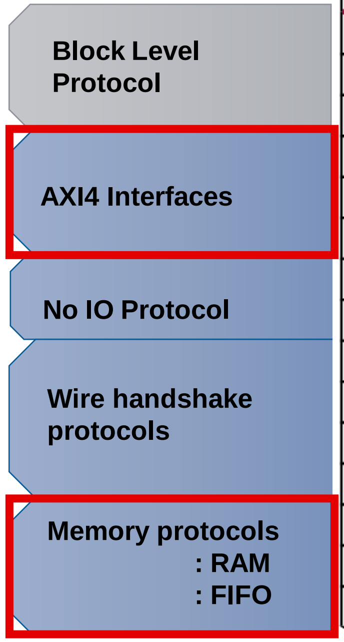
Directive: INTERFACE

Destination: Source File / Directive File

Options:
mode (optional): ap_ack, ap_fifo, ap_hs, ap_memory, ap_none, ap_ovld, ap_stable, ap_vld, axis, bram, m_axi, [✓ s_axilite]
bundle (optional):
clock (optional):
depth (optional):
latency (optional):
max_widen_bitwidth (optional):
name (optional):
port (optional):
register (optional):
storage_impl (optional):
storage_type (optional):

# 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

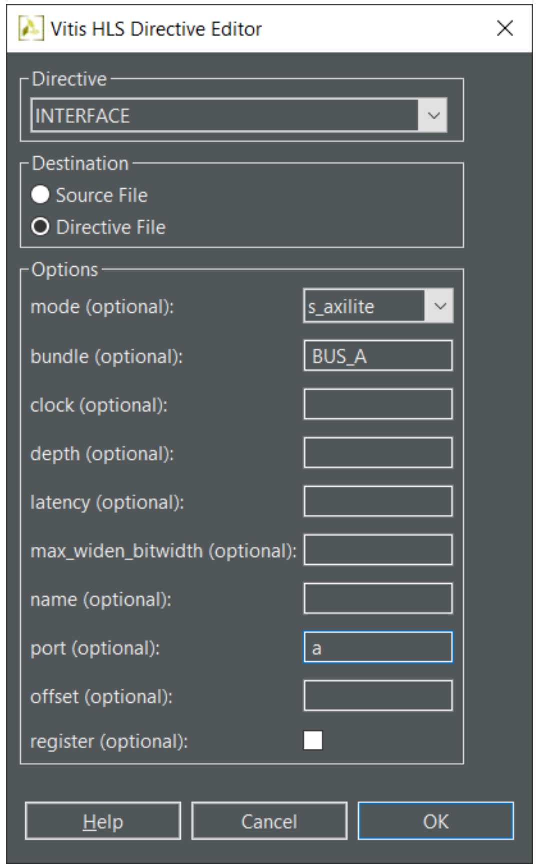


| Argument Type | Scalar | | Array | | | Pointer or Reference | | | Hls::stream |
|---|---|---|---|---|---|---|---|---|---|
| Interface Mode | Input | Return | I | I/O | O | I | I/O | O | I and O |
| ap_ctrl_none | 3 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| ap_ctrl_hs | 3 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| ap_ctrl_chain | 3 | D[1] | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| axis | 1 | 3 | 1 | 3 | 1 | 1 | 3 | 1 | 1 |
| 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] |

# 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;
}
```


Vitis HLS Directive Editor

Directive: INTERFACE

Destination: Source File / Directive File

Options:
mode (optional): s_axilite
bundle (optional): BUS_A
clock (optional):
depth (optional):
latency (optional):
max_widen_bitwidth (optional):
name (optional):
port (optional): a
offset (optional):
register (optional):

# 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

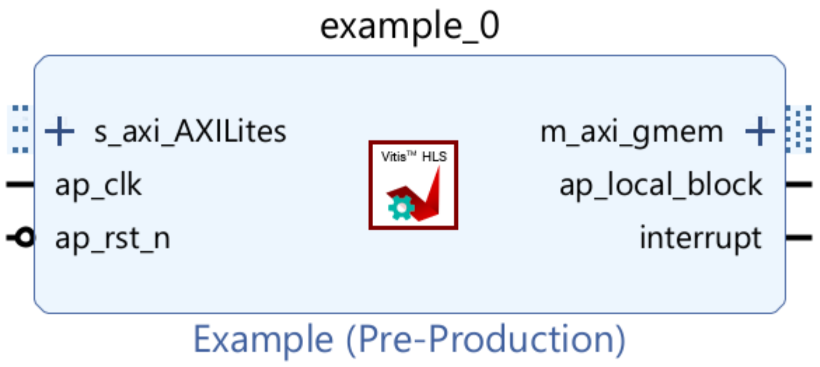
```
...
#define XHLS_SIG_GEN_BRAM2AXIS_CTRL_ADDR_SIG_PERIOD_DATA 0x0400
...
#define XHLS_SIG_GEN_BRAM2AXIS_CTRL_ADDR_SIG_BUF_BASE    0x1000
#define XHLS_SIG_GEN_BRAM2AXIS_CTRL_ADDR_SIG_BUF_HIGH    0x17ff
...
```

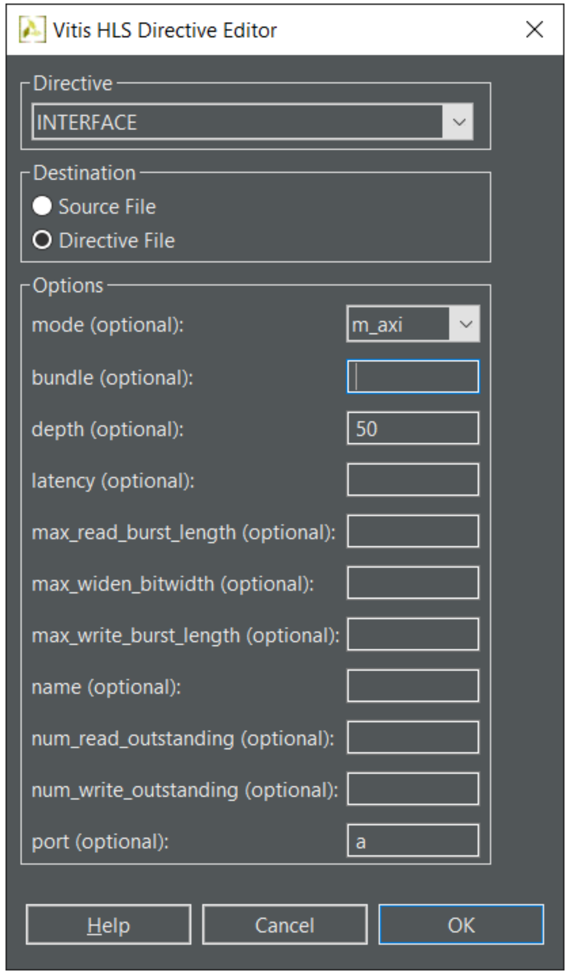
## Native AXI4 Master

- 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
```




Vitis HLS Directive Editor

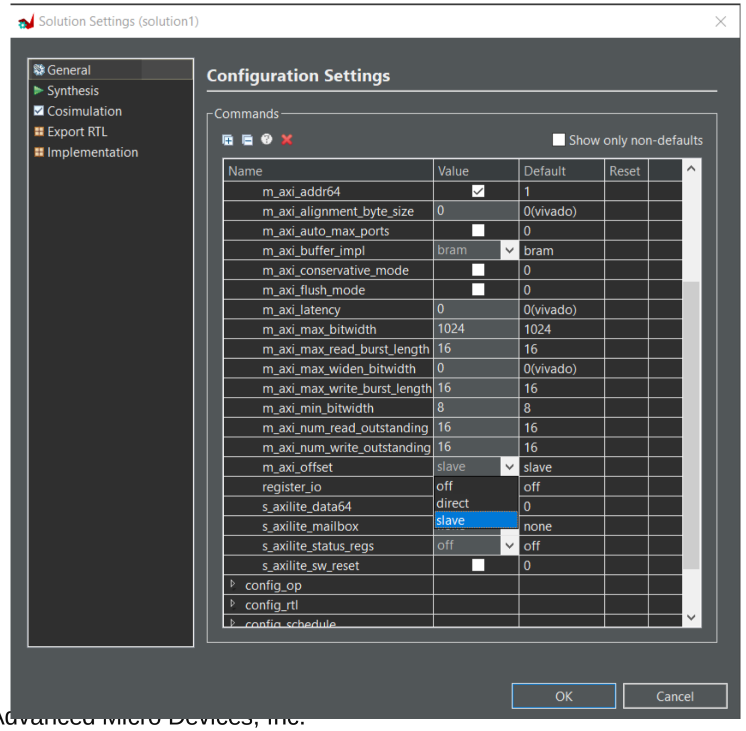
Directive: INTERFACE

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 |

## 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



Solution Settings (solution1)

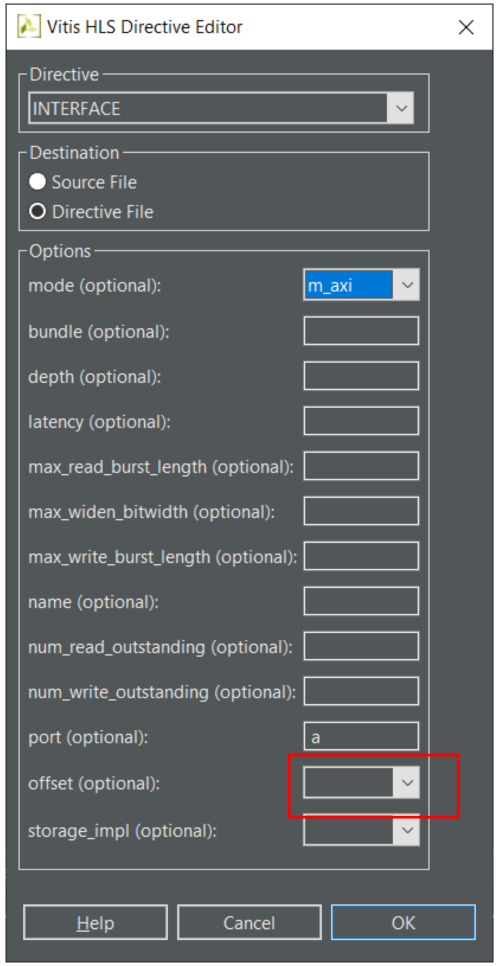
General | Synthesis | Cosimulation | Export RTL | Implementation

**Configuration Settings**

Commands

☐ Show only non-defaults

| Name | Value | Default | Reset |
|---|---|---|---|
| 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 | direct | 0 | |
| s_axilite_mailbox | slave | none | |
| s_axilite_status_regs | off | off | |
| s_axilite_sw_reset | ☐ | 0 | |
| config_op | | | |
| config_rtl | | | |
| config_schedule | | | |



Vitis HLS Directive Editor

Directive: INTERFACE

Destination: ● Source File ○ Directive File

Options

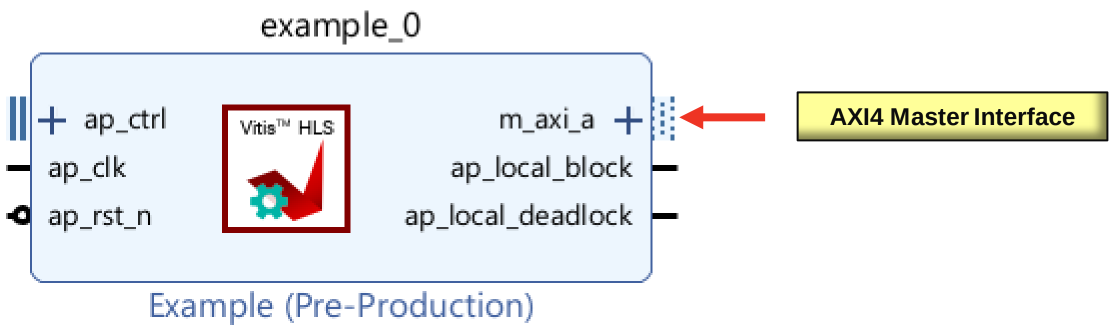
| | |
|---|---|
| mode (optional): | m_axi |
| bundle (optional): | |
| depth (optional): | |
| latency (optional): | |
| max_read_burst_length (optional): | |
| max_widen_bitwidth (optional): | |
| max_write_burst_length (optional): | |
| name (optional): | |
| num_read_outstanding (optional): | |
| num_write_outstanding (optional): | |
| port (optional): | a |
| offset (optional): | |
| storage_impl (optional): | |

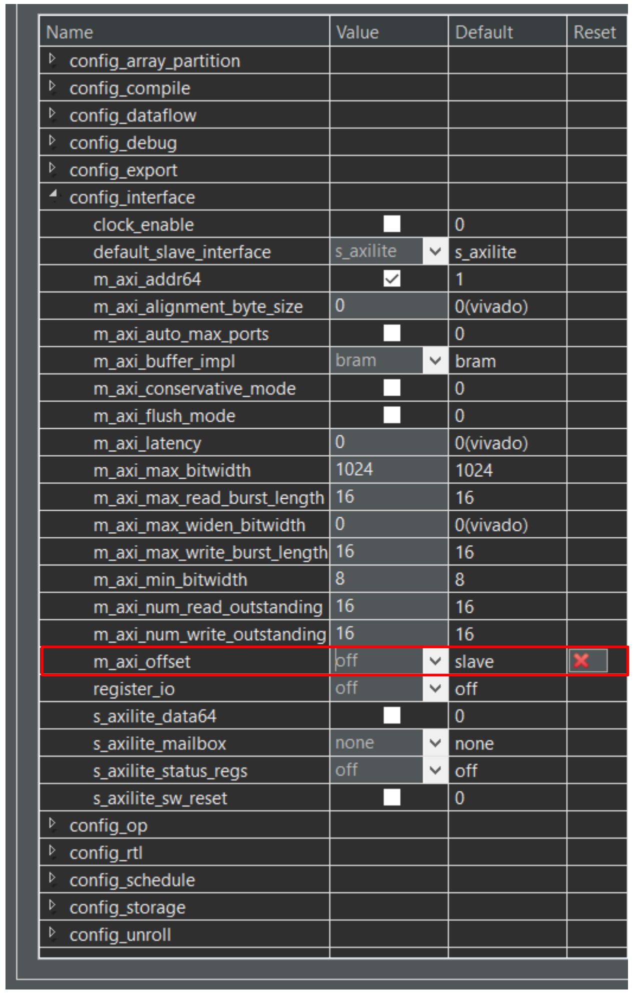
AMD XILINX

# 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 not be changed on the fly

```
config_interface -m_axi_offset off
```





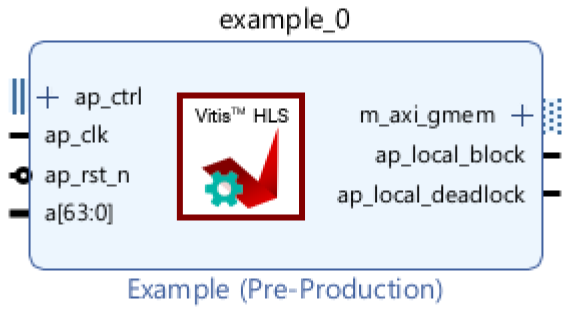
| Name | Value | Default | Reset |
|---|---|---|---|
| config_array_partition | | | |
| config_compile | | | |
| config_dataflow | | | |
| config_debug | | | |
| config_export | | | |
| config_interface | | | |
| clock_enable | ☐ | 0 | |
| default_slave_interface | s_axilite | s_axilite | |
| m_axi_addr64 | ☑ | 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 | | | |

# Native AXI4 Master: Offset=direct

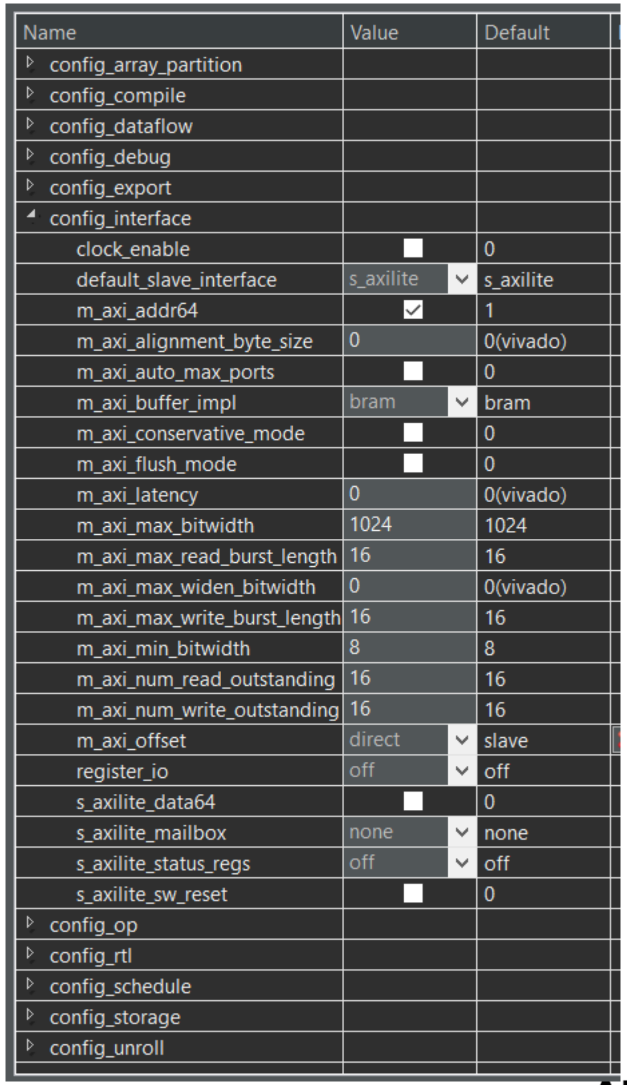
- Direct Interface
  - Generates a scalar input offset port
  - The offset is set by driving the input port
  - It can be changed on the fly by driving the port with a different value

```
config_interface -m_axi_offset direct
```

**...Plus port to specify address offset**



**AXI4 Master Interface ...**

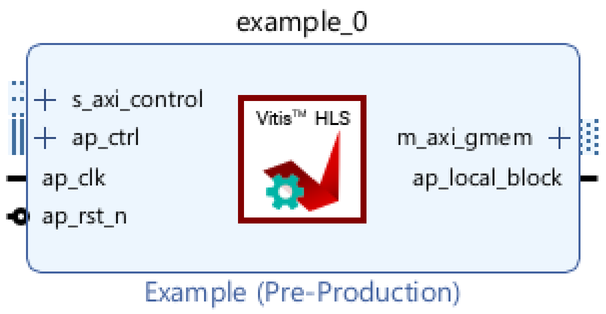


| Name | Value | Default |
|---|---|---|
| config_array_partition | | |
| config_compile | | |
| config_dataflow | | |
| config_debug | | |
| config_export | | |
| config_interface | | |
| clock_enable | ☐ | 0 |
| default_slave_interface | s_axilite | s_axilite |
| m_axi_addr64 | ✓ | 1 |
| m_axi_alignment_byte_size | 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 | | |

## Native AXI4 Master: Offset=slave(Default)

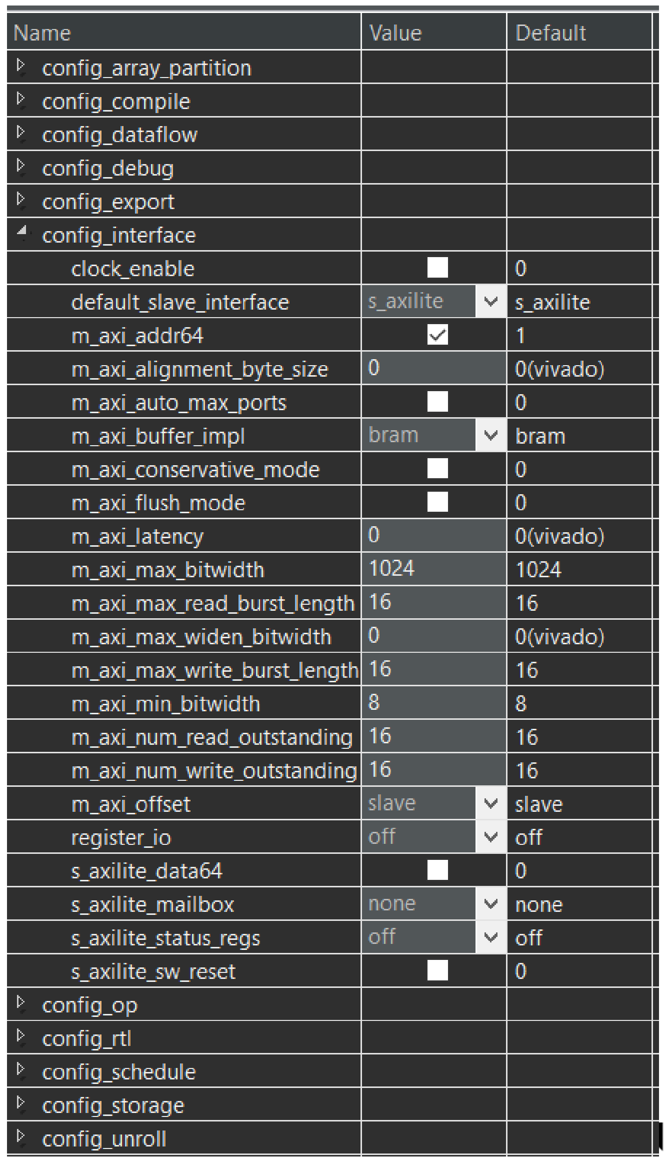
- Direct Interface
  - Generates an offset port and automatically maps it to an AXI4 Slave Lite interface
  - User must program the offset before starting transactions on the AXI4 Master interface
  - It can changed on the fly by re-programming the offset register

```
config_interface -m_axi_offset slave
```



…Plus AXI4 Slave Lite interface to program the 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 | 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];
}
}
```

# 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];
}
…
}
```

## 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
```

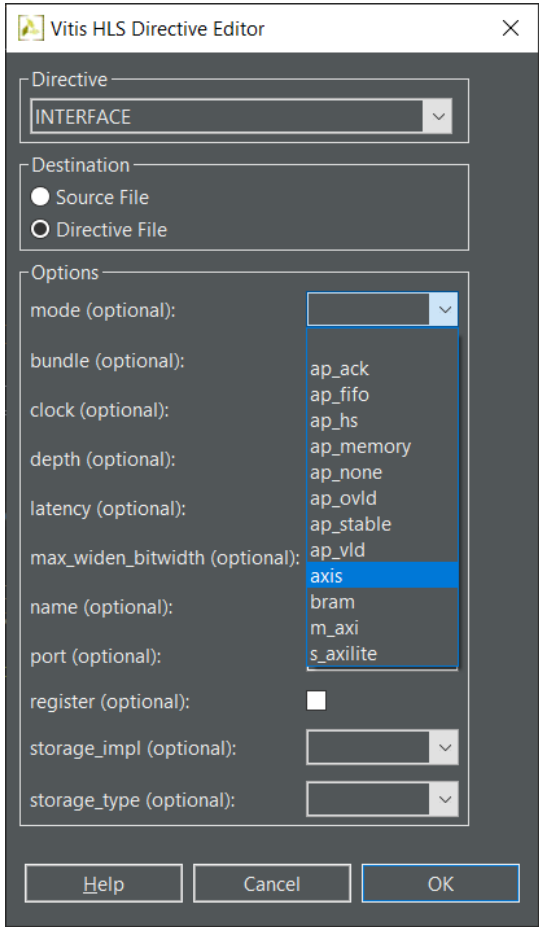
- Arguments can be Bundled into AXI4 Master and AXI4 Lite ports
  - If no bundle name is used a default name is used for all arguments
    - All go into a single AXI4 Master or AXI4 Lite
    - Default name applied if no –bundle option is used
  - Group different sized variables into an AXI4 Master port

# 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
```



Vitis HLS Directive Editor

Directive: INTERFACE

Destination: [✓ Source File] Directive File

Options:

mode (optional): [✓ axis]

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

bundle (optional):

clock (optional):

depth (optional):

latency (optional):

max_widen_bitwidth (optional):

name (optional):

port (optional):

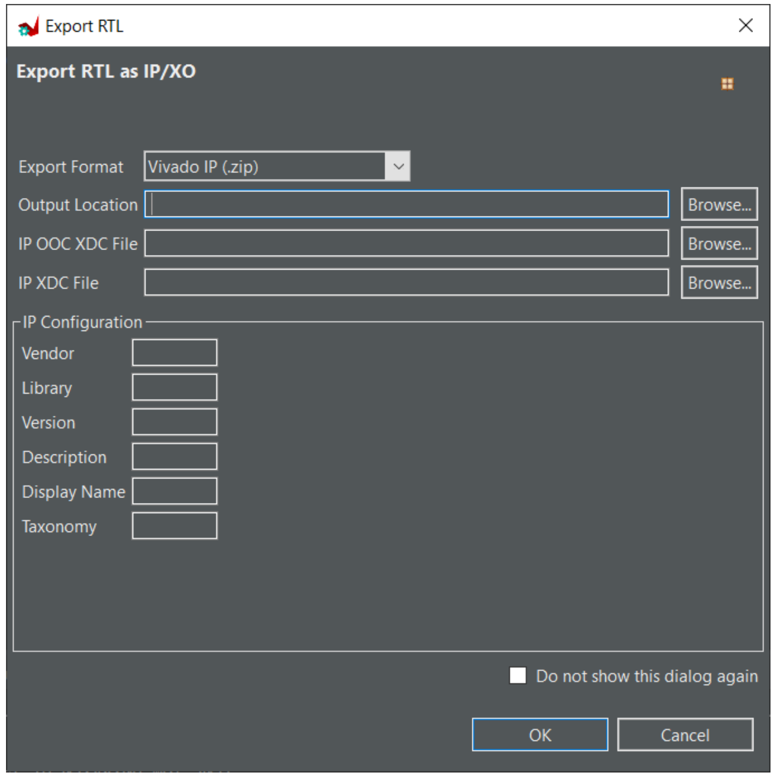
register (optional):

storage_impl (optional):

storage_type (optional):

## 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.


Export RTL

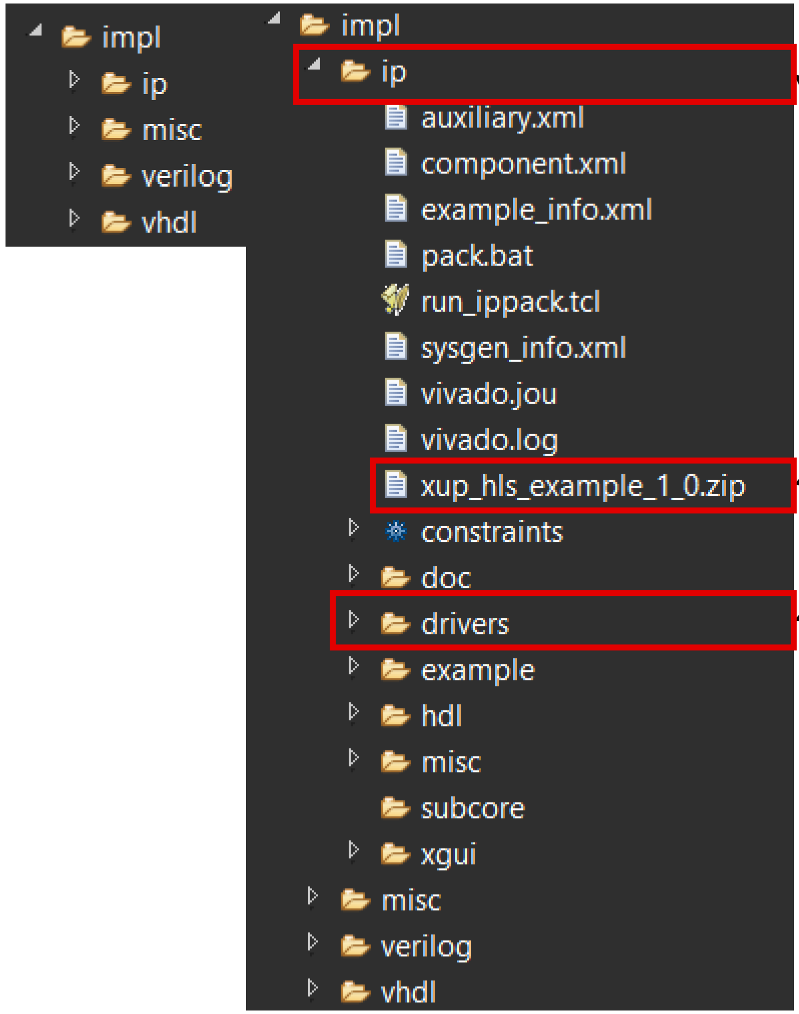
Export RTL as IP/XO

Export Format: Vivado IP (.zip)
Output Location:
IP OOC XDC File:
IP XDC File:

IP Configuration
Vendor:
Library:
Version:
Description:
Display Name:
Taxonomy:

AMD XILINX

# Generated Impl Directory



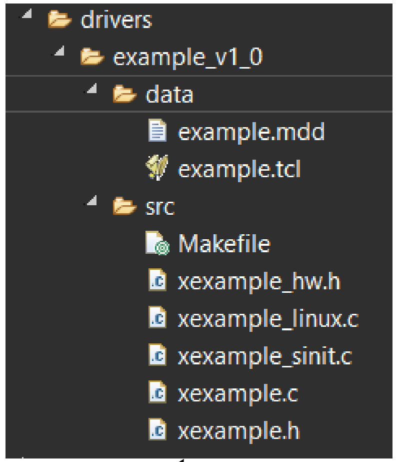
```
impl
  ip
  misc
  verilog
  vhdl
```

```
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**

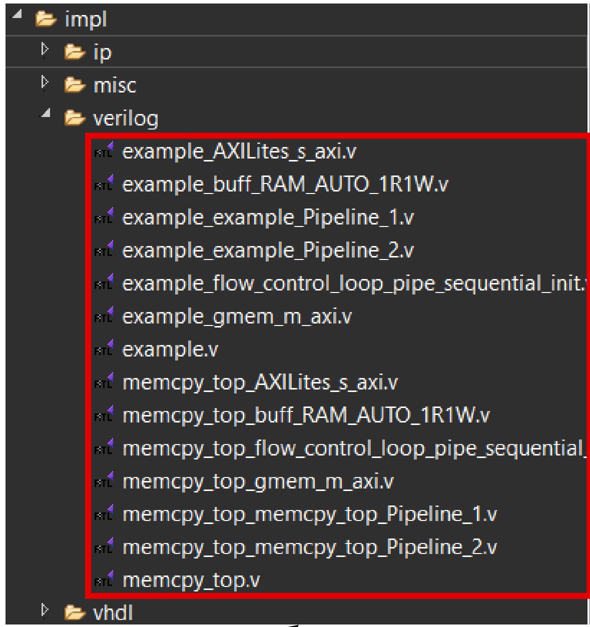
**IP Integrator will use this file**

**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**

# Integrating the Hardware Accelerator IP in Embedded System

# 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)

## Summary

## 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

# Thank You

## Disclaimer and Attribution