Creating Processor System

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 AP 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 AP SoC architecture consists of two major sections**
  - PS: Processing system
    - 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
      - Artix™-based devices: Z-7010, Z-7015, and Z-7020 (high-range I/O banks only)
      - Single core versions: Z-7017S, Z-7012S, and Z-7014S
      - Kintex™-based devices: Z-7030, Z-7035, Z-7045, and Z-7100 (mix of high-range and high-performance I/O banks)

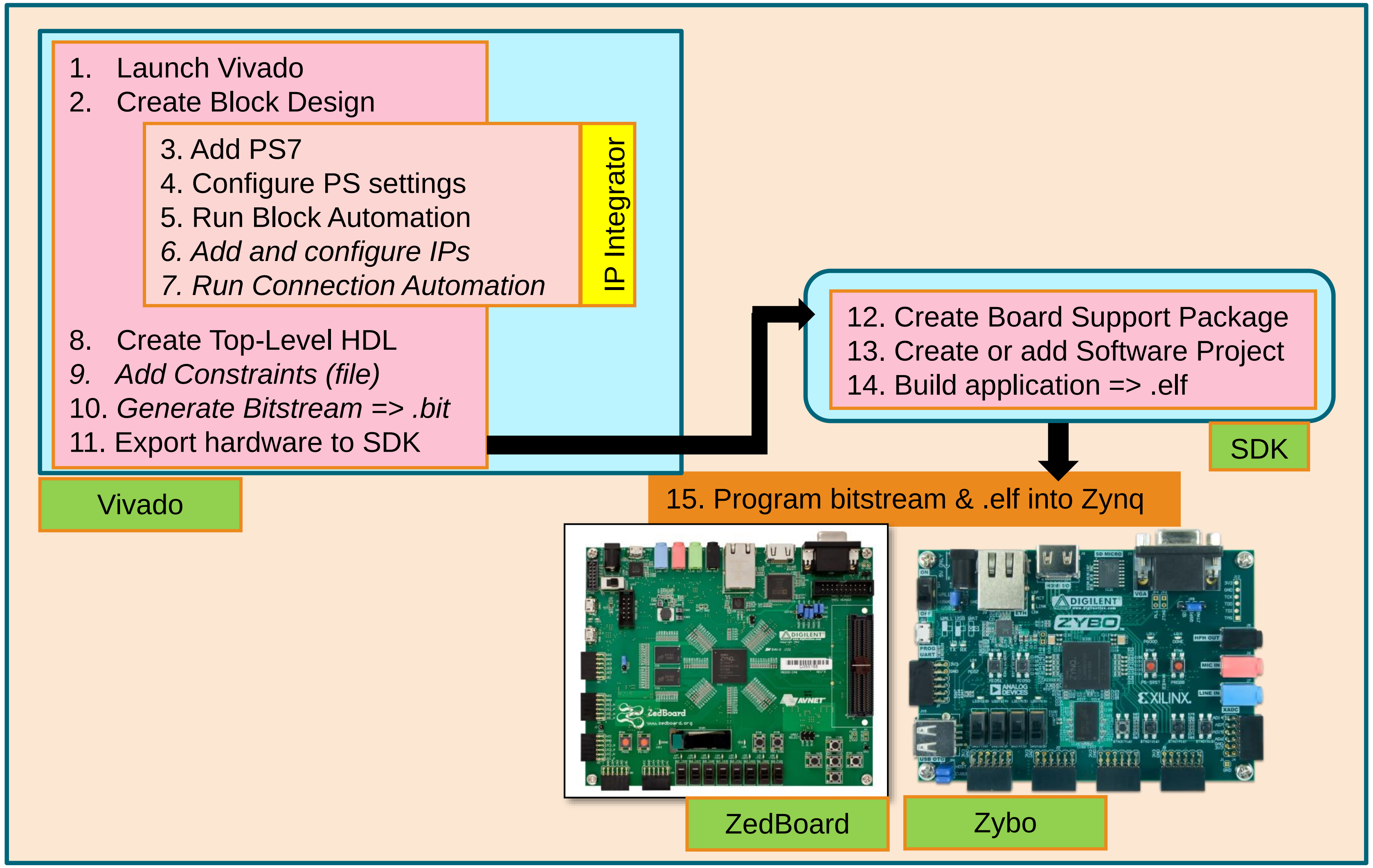
Vivado

- **What are Vivado, IP Integrator and SDK?**
  - 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
    - Integrated into Vivado
    - Vivado includes all the tools, IP, and documentation that are required for designing systems with the Zynq-7000 AP SoC hard core and/or Xilinx MicroBlaze soft core processor
    - Vivado + IPI replaces ISE/EDK
  - SDK is an Eclipse-based software design environment
    - Enables the integration of hardware and software components
    - Links from Vivado

- **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

Embedded System Design using Vivado

https://github.com/wady100/Embedded-System-Design-Flow-on-Zynq



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 SDK

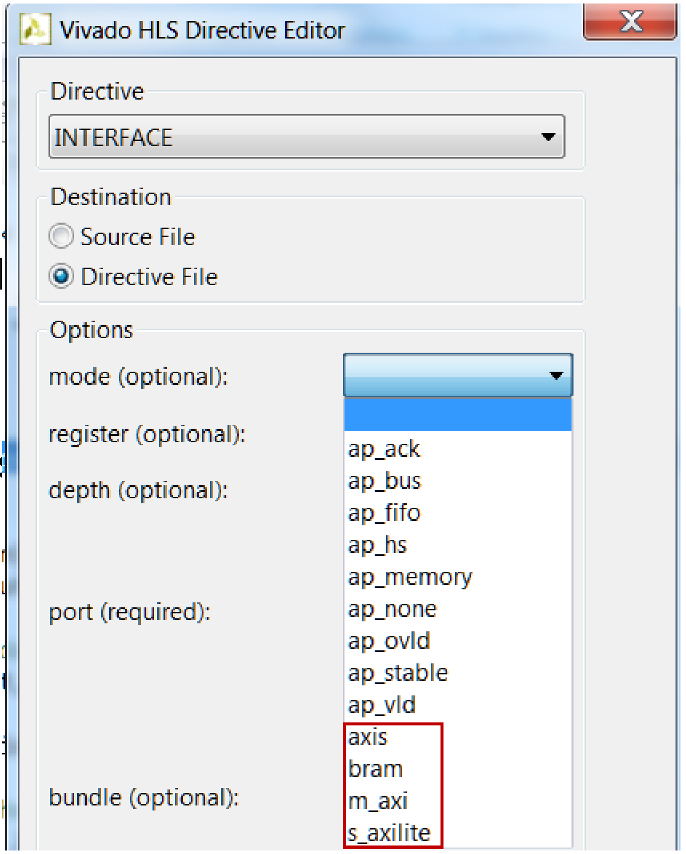
12. Create Board Support Package
13. Create or add Software Project
14. Build application => .elf

15. Program bitstream & .elf into Zynq

Creating IP-XACT Hardware Accelerator

Port-Level Interfaces

- **The AXI4 interfaces supported by Vivado HLS include**
  - The AXI4-Stream (axis)
    - Specify on input arguments or output arguments only, not on input/output argume
  - The AXI4 master (m_axi)
    - Specify on arrays and pointers (and references in C++) only. You can group multi AXI4-Lite interface using the `bundle` option
  - The AXI4-Lite (s_axilite)
    - Specify on any type of argument except arrays. You can group multiple argument interface using the `bundle` option


Vivado HLS Directive Editor

Directive
INTERFACE

Destination
Source File
Directive File

Options
mode (optional):
register (optional):
depth (optional):
port (required):
bundle (optional):

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

```
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;
}
```

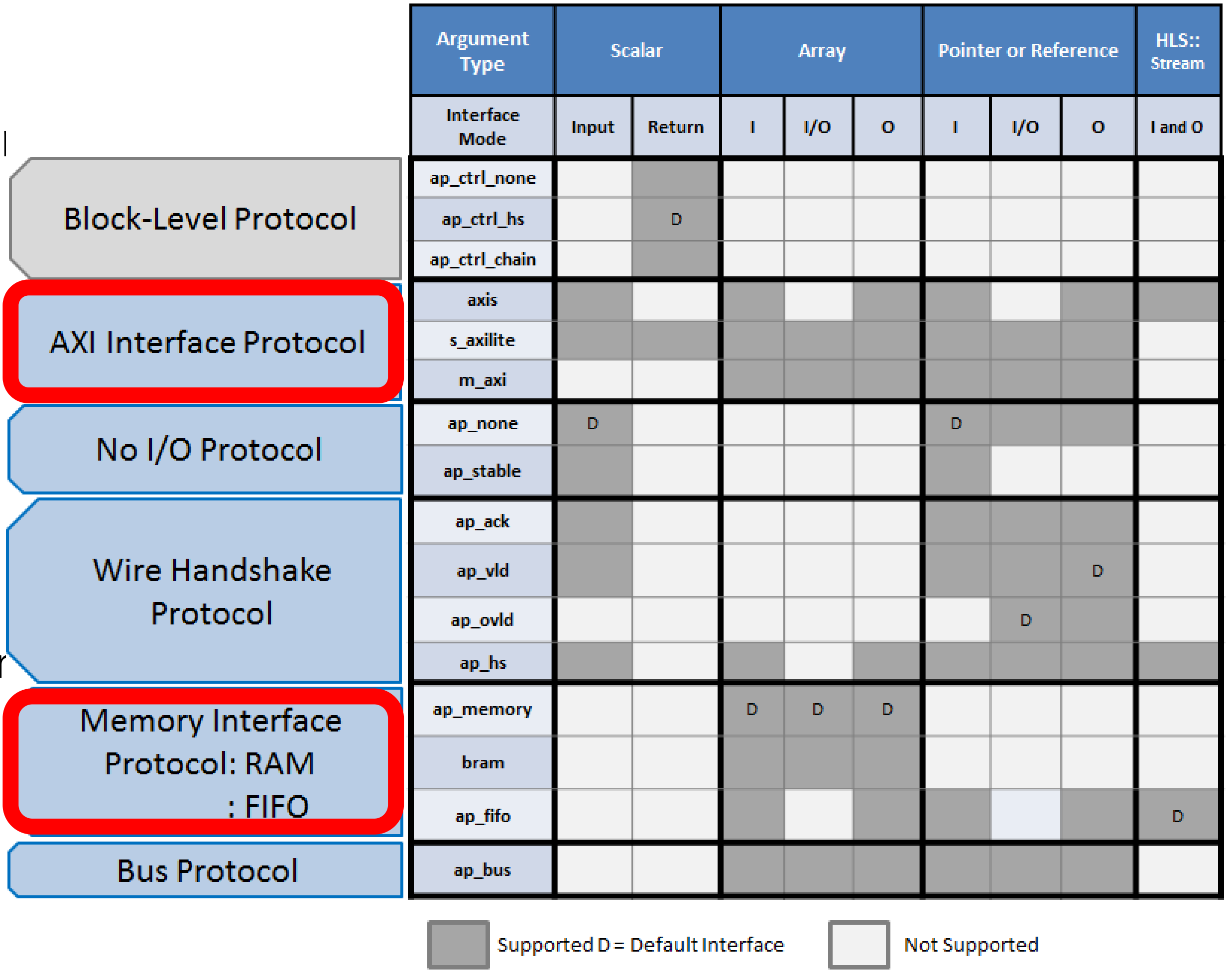
Interface Modes

Native AXI Interfaces

- AXI4 Slave Lite, AXI4 Master, AXI Stream suppo
  - 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 i



	Argument Type	Scalar		Array			Pointer or Reference			HLS:: Stream
	Interface Mode	Input	Return	I	I/O	O	I	I/O	O	I and O
Block-Level Protocol	ap_ctrl_none	□	▨	□	□	□	□	□	□	□
	ap_ctrl_hs	□	▨ D	□	□	□	□	□	□	□
	ap_ctrl_chain	□	▨	□	□	□	□	□	□	□
AXI Interface Protocol	axis	▨	□	▨	□	▨	▨	□	▨	▨
	s_axilite	▨	▨	▨	▨	▨	▨	▨	▨	□
	m_axi	□	□	▨	▨	▨	▨	▨	▨	□
No I/O Protocol	ap_none	▨ D	□	□	□	□	▨ D	▨	▨	□
	ap_stable	▨	□	□	□	□	▨	□	□	□
Wire Handshake Protocol	ap_ack	▨	□	□	□	□	▨	▨	▨	□
	ap_vld	▨	□	□	□	□	▨	▨	▨ D	□
	ap_ovld	□	□	□	□	□	□	▨ D	▨	□
	ap_hs	▨	□	▨	□	▨	▨	▨	▨	▨
Memory Interface Protocol: RAM : FIFO	ap_memory	□	□	▨ D	▨ D	▨ D	□	□	□	□
	bram	□	□	▨	▨	▨	□	□	□	□
	ap_fifo	□	□	▨	□	▨	▨	□	▨	▨ D
Bus Protocol	ap_bus	□	□	▨	▨	▨	▨	▨	▨	□

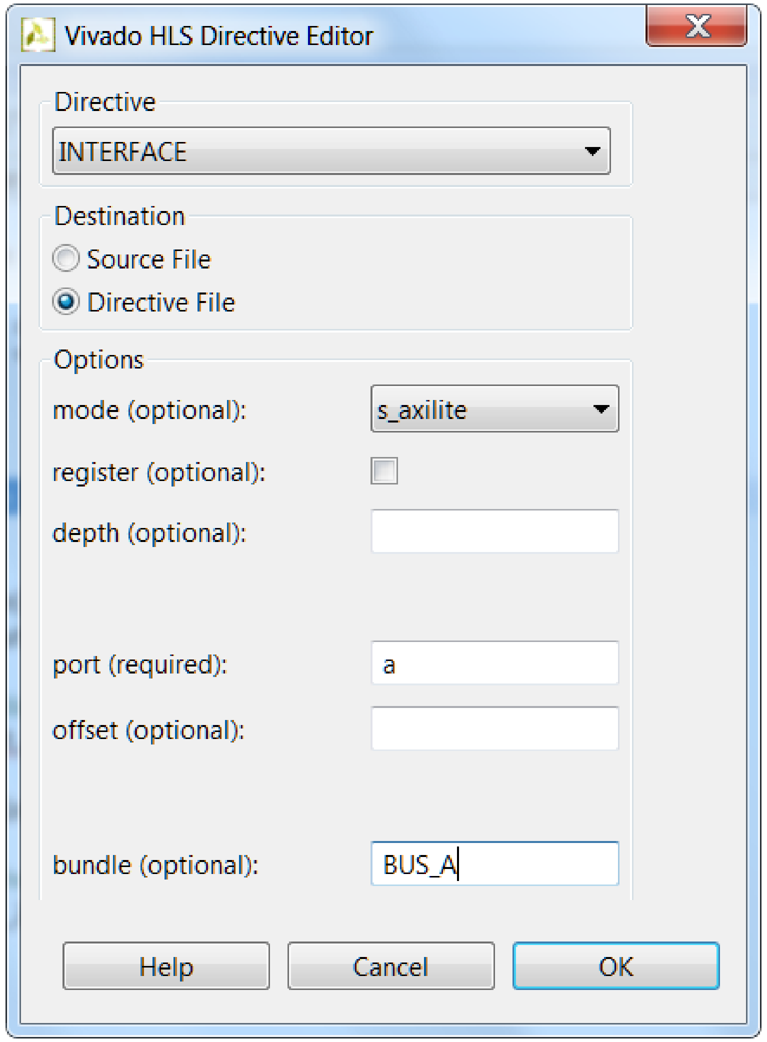
▨ Supported D = Default Interface □ Not Supported

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 port)
  - Default mode can be changed with additional INTERFACE directives

```
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
#pragma HLS INTERFACE ap_hs     port=a
#pragma HLS INTERFACE ap_vld    port=b
#pragma HLS INTERFACE ap_none   port=c register
```



Directive: INTERFACE

Destination: ( ) Source File [✓ Directive File]

Options

mode (optional):	s_axilite
register (optional):	☐
depth (optional):	
port (required):	a
offset (optional):	
bundle (optional):	BUS_A

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

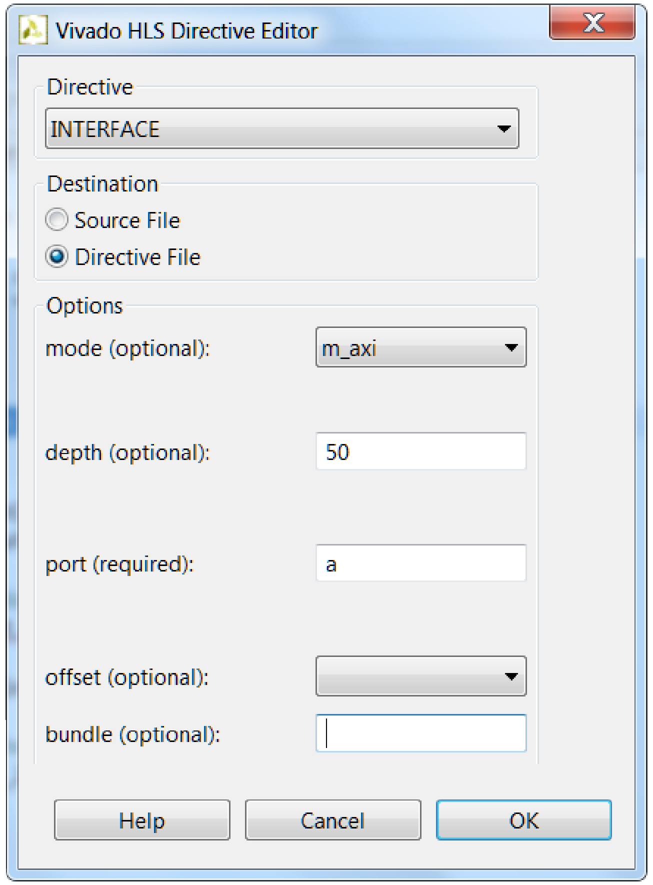
```
void hls_sig_gen_bram2axis(hls::stream<axis_last_t<data_t> >& dout,
                           data_t sig_buf[MAX_SIG_PERIOD], short sig_period)
{
#pragma HLS INTERFACE port=return     s_axilite bundle=ctrl
#pragma HLS INTERFACE port=sig_buf    s_axilite bundle=ctrl offset=0x1000
#pragma HLS INTERFACE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



Vivado HLS Directive Editor

Directive: INTERFACE

Destination
- Source File
- [✓ Directive File]

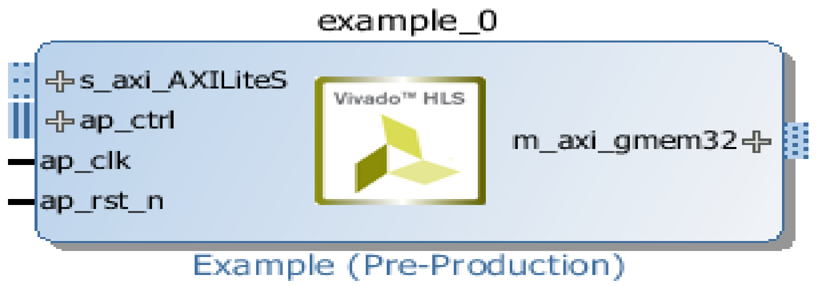
Options

mode (optional):	m_axi
depth (optional):	50
port (required):	a
offset (optional):	
bundle (optional):	

Help | Cancel | OK

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

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



Burst Accesses Inferred for AXI4 Master

- **There are two types of accesses on an AXI Master**

Single Access

```
void example(int *a)
{
#pragma HLS INTERFACE m_axi port=a depth=...
   ...
   int val[i] = *(a + i);
   ...
```

Burst Access

```
void example(int *a)
{
#pragma HLS INTERFACE m_axi port=a depth=...
   ...
   memcpy(vals, a, N * sizeof(int));
   ...
```

  – Burst accesses are more efficient
  – Burst access has until now required the use of memcpy()

- **Burst Accesses are now inferred**
  – From operations in a for-loop and from sequential operations in the code
  – However: there are some limitations
    - Single for-loops only, no nested loops

XILINX

Byte-Enable Accesses on AXI4 Master

- **Byte-Enable Accesses Support on AXI4 Master Interfaces**
  - Single bytes are now written and read
  - Improved AXI4 Master performance
- **Improved Performance**
  - This code uses 8-bit data

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

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

    - Previously, accessing this required reading/writing full 32-bit
    - This implied a required read-modify-write behavior: Impacted performance
  - Similar performance improvement when accessing struct members
    - Also often implied read-modify-write behavior
  - Improved Port Bundling
    - Variables of different sizes can be grouped into same AXI4 Master port

AXI4 Port Bundling

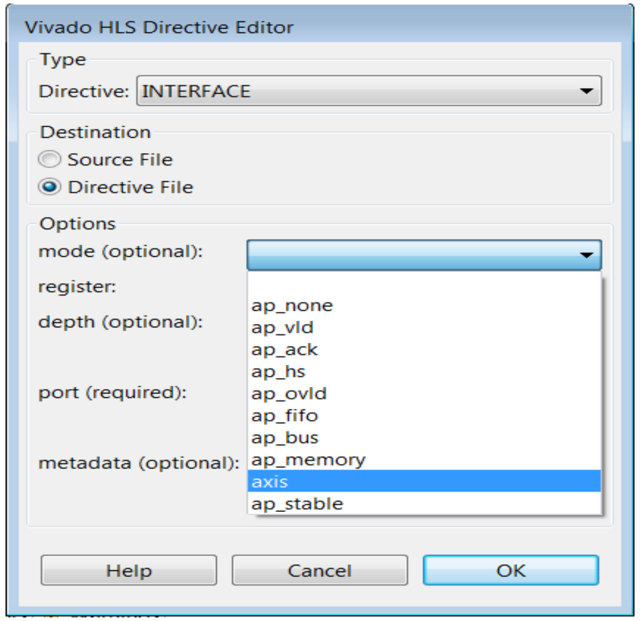
- **AXI4 Master and Lite Port Bundling**
  - The bundle options groups arguments into the same AXI4 port
  - For example, group 3 arguments into AXI4 port “ctrl” :

```
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     s_axilite bundle=ctrl
#pragma HLS INTERFACE port=sig_buf    s_axilite bundle=ctrl
#pragma HLS INTERFACE port=sig_period s_axilite bundle=ctrl
#pragma HLS INTERFACE port=dout 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


Vivado HLS Directive Editor

Type
Directive: INTERFACE

Destination
Source File
[✓ Directive File]

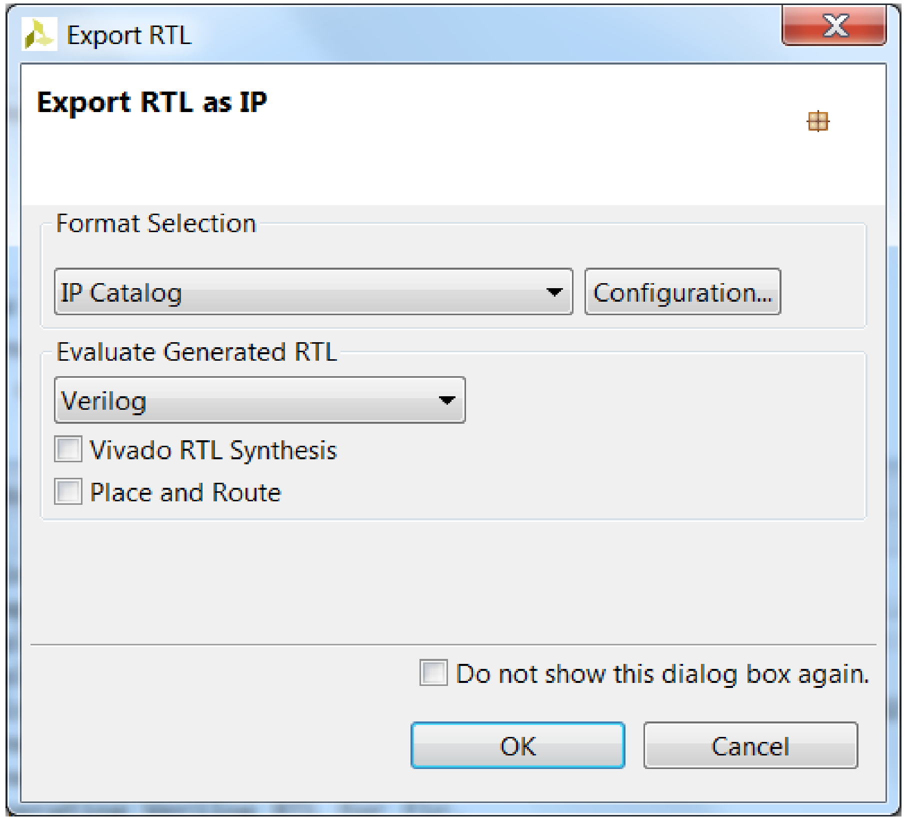
Options
mode (optional):
register:
depth (optional):
port (required):
metadata (optional):

Interface Type "axis" is AXI4 Stream

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

Generate the hardware accelerator

- **Select Solution > Export RTL**
- **Select IP Catalog, System Generator for Vivado or design check point (dcp)**
- **Click on Configuration… if you want to change the version number or other information**
  - Default is v1_00_a
- **Click on OK**
  - The directory (ip) will be generated under the impl folder under the current project directory and current solution
  - RTL code will be generated, both for Verilog and VHDL languages in their respective folders


Export RTL

Export RTL as IP

Format Selection: IP Catalog

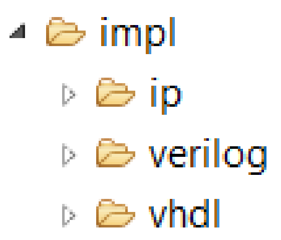
Evaluate Generated RTL: Verilog

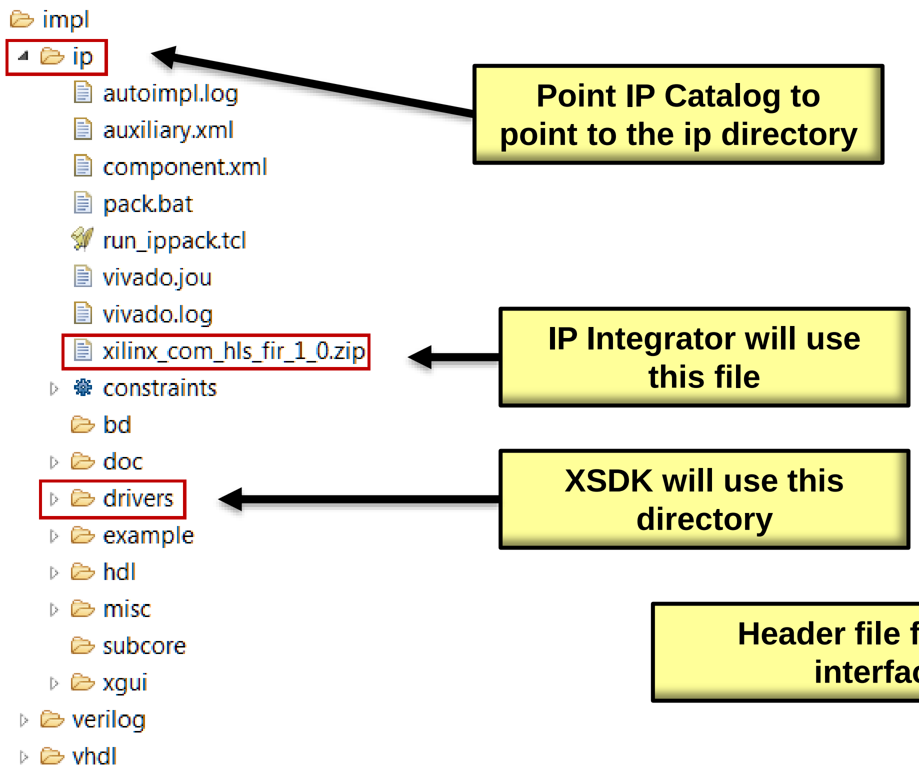
[ ] Vivado RTL Synthesis

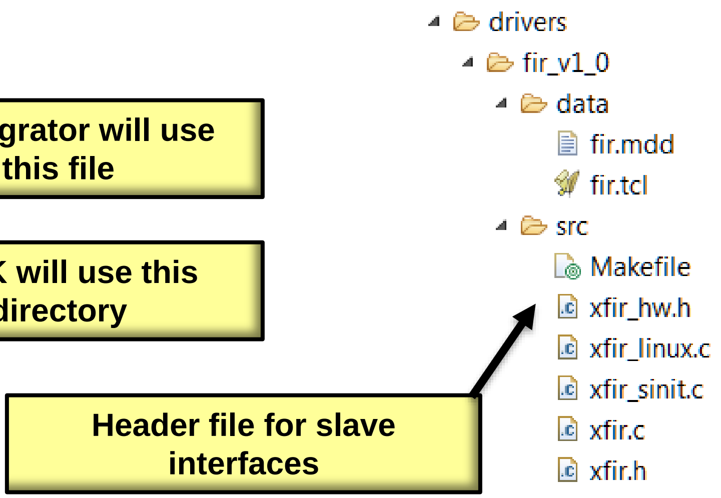
[ ] Place and Route

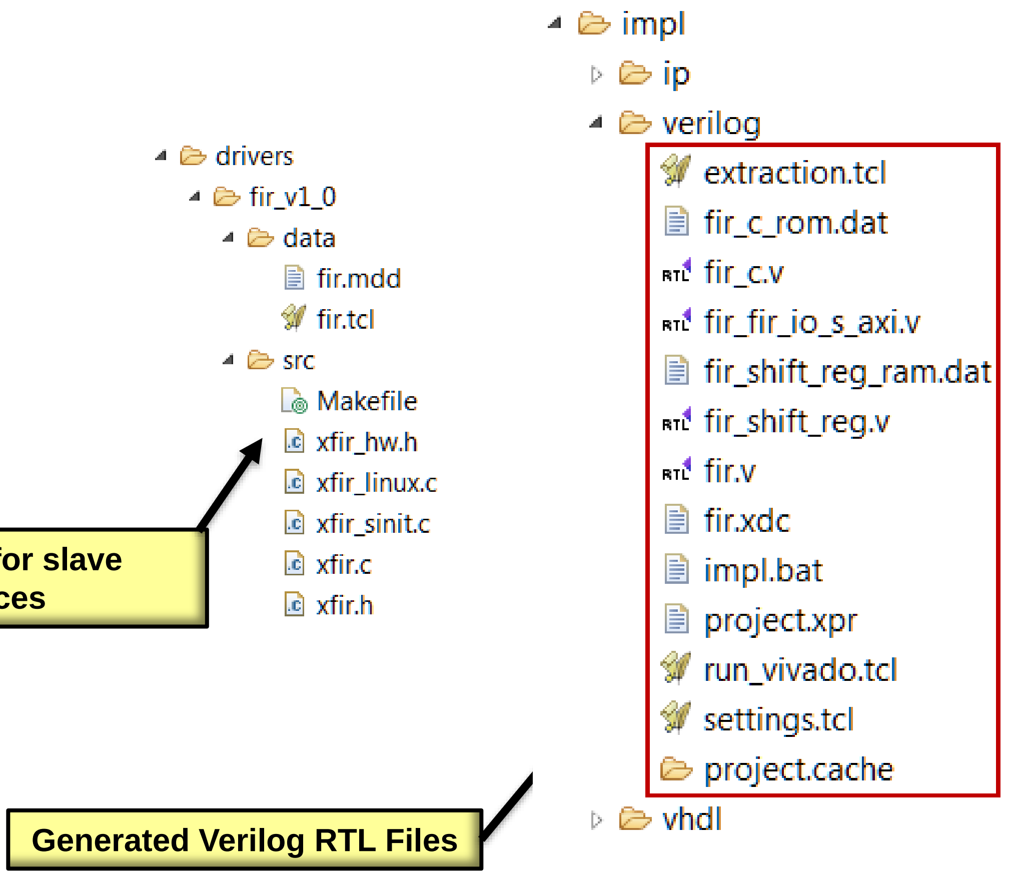
[ ] Do not show this dialog box again.

Generated impl Directory

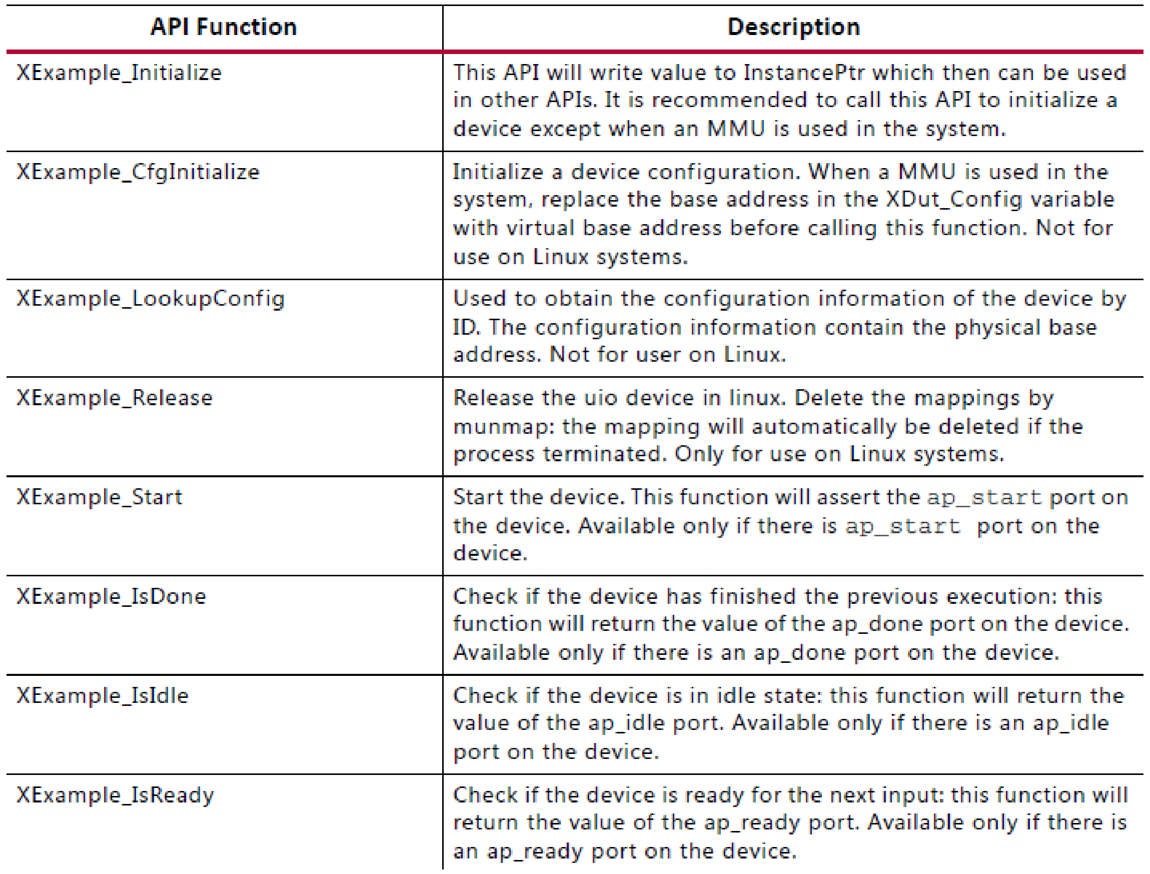
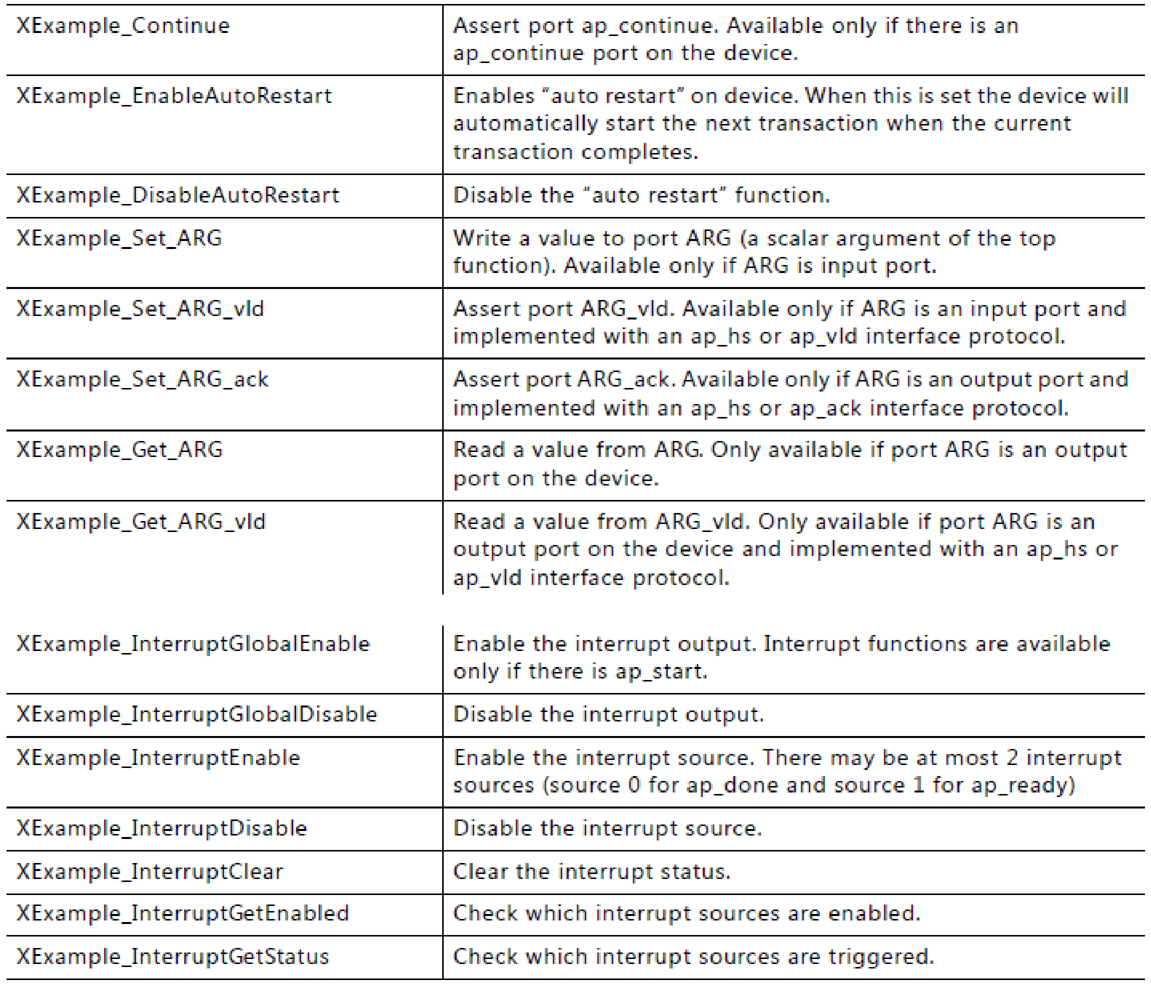








C Driver API for AXI4-Lite Interface

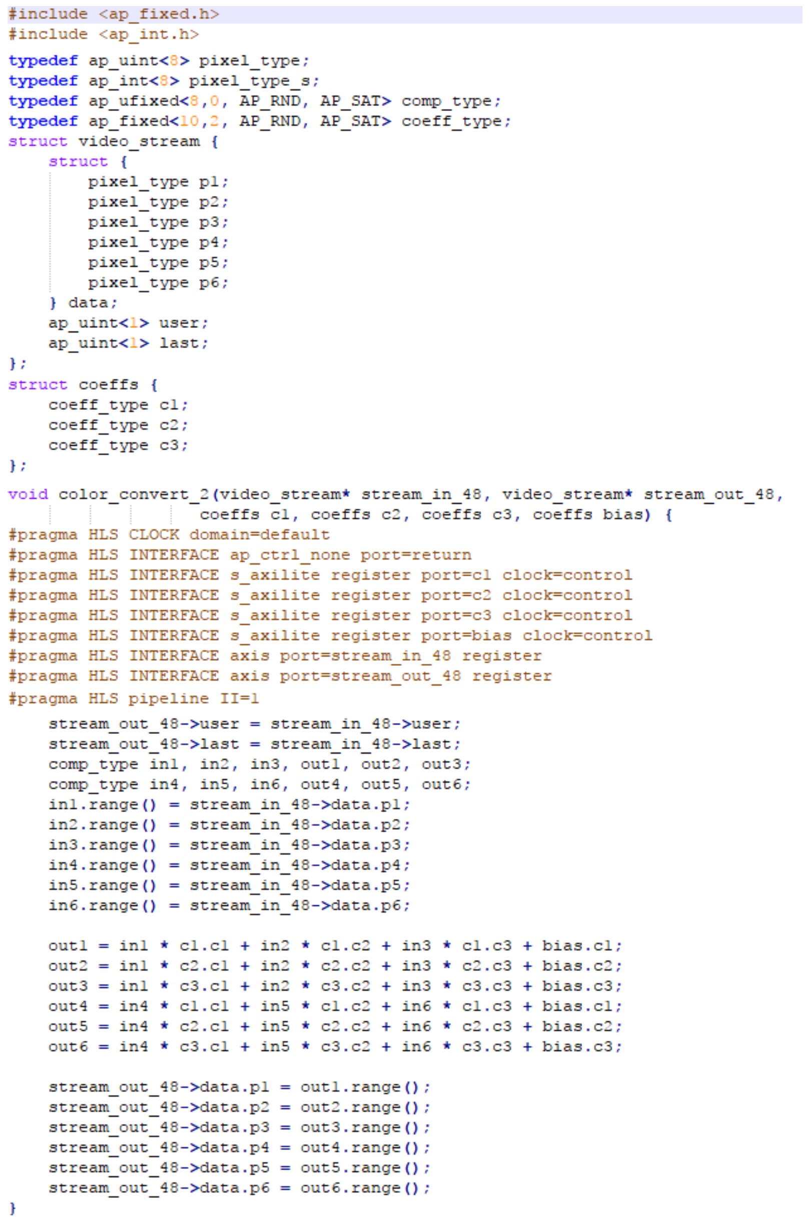



API Function	Description
XExample_Initialize	This API will write value to InstancePtr which then can be used in other APIs. It is recommended to call this API to initialize a device except when an MMU is used in the system.
XExample_CfgInitialize	Initialize a device configuration. When a MMU is used in the system, replace the base address in the XDut_Config variable with virtual base address before calling this function. Not for use on Linux systems.
XExample_LookupConfig	Used to obtain the configuration information of the device by ID. The configuration information contain the physical base address. Not for user on Linux.
XExample_Release	Release the uio device in linux. Delete the mappings by munmap: the mapping will automatically be deleted if the process terminated. Only for use on Linux systems.
XExample_Start	Start the device. This function will assert the `ap_start` port on the device. Available only if there is `ap_start` port on the device.
XExample_IsDone	Check if the device has finished the previous execution: this function will return the value of the ap_done port on the device. Available only if there is an ap_done port on the device.
XExample_IsIdle	Check if the device is in idle state: this function will return the value of the ap_idle port. Available only if there is an ap_idle port on the device.
XExample_IsReady	Check if the device is ready for the next input: this function will return the value of the ap_ready port. Available only if there is an ap_ready port on the device.
XExample_Continue	Assert port ap_continue. Available only if there is an ap_continue port on the device.
XExample_EnableAutoRestart	Enables "auto restart" on device. When this is set the device will automatically start the next transaction when the current transaction completes.
XExample_DisableAutoRestart	Disable the "auto restart" function.
XExample_Set_ARG	Write a value to port ARG (a scalar argument of the top function). Available only if ARG is input port.
XExample_Set_ARG_vld	Assert port ARG_vld. Available only if ARG is an input port and implemented with an ap_hs or ap_vld interface protocol.
XExample_Set_ARG_ack	Assert port ARG_ack. Available only if ARG is an output port and implemented with an ap_hs or ap_ack interface protocol.
XExample_Get_ARG	Read a value from ARG. Only available if port ARG is an output port on the device.
XExample_Get_ARG_vld	Read a value from ARG_vld. Only available if port ARG is an output port on the device and implemented with an ap_hs or ap_vld interface protocol.
XExample_InterruptGlobalEnable	Enable the interrupt output. Interrupt functions are available only if there is ap_start.
XExample_InterruptGlobalDisable	Disable the interrupt output.
XExample_InterruptEnable	Enable the interrupt source. There may be at most 2 interrupt sources (source 0 for ap_done and source 1 for ap_ready)
XExample_InterruptDisable	Disable the interrupt source.
XExample_InterruptClear	Clear the interrupt status.
XExample_InterruptGetEnabled	Check which interrupt sources are enabled.
XExample_InterruptGetStatus	Check which interrupt sources are triggered.

Integrating the Hardware Accelerator in AXI System

Embedded System Design using Vivado

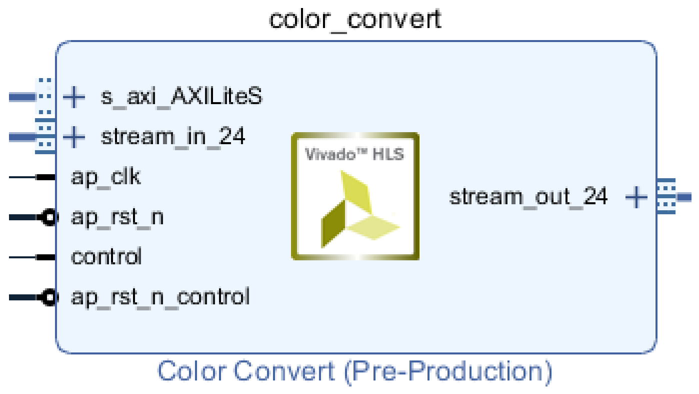
- Create a new Vivado project, or open an existing project
- Invoke IP Integrator
- Construct(modify) the hardware portion of the embedded design by adding the IP-XACT hardware accelerator created in Vivado HLS
- Create (Update) top level HDL wrapper
- Synthesize any non-embedded components and implement in Vivado
- Export the hardware description, and launch XSDK
- Create a new software board support package and application projects in the XSDK
- Compile the software with the GNU cross-compiler in XSDK
- Download the programmable logic’s completed bitstream using Xilinx Tools > Program FPGA in XSDK
- Use XSDK to download and execute the program (the ELF file)

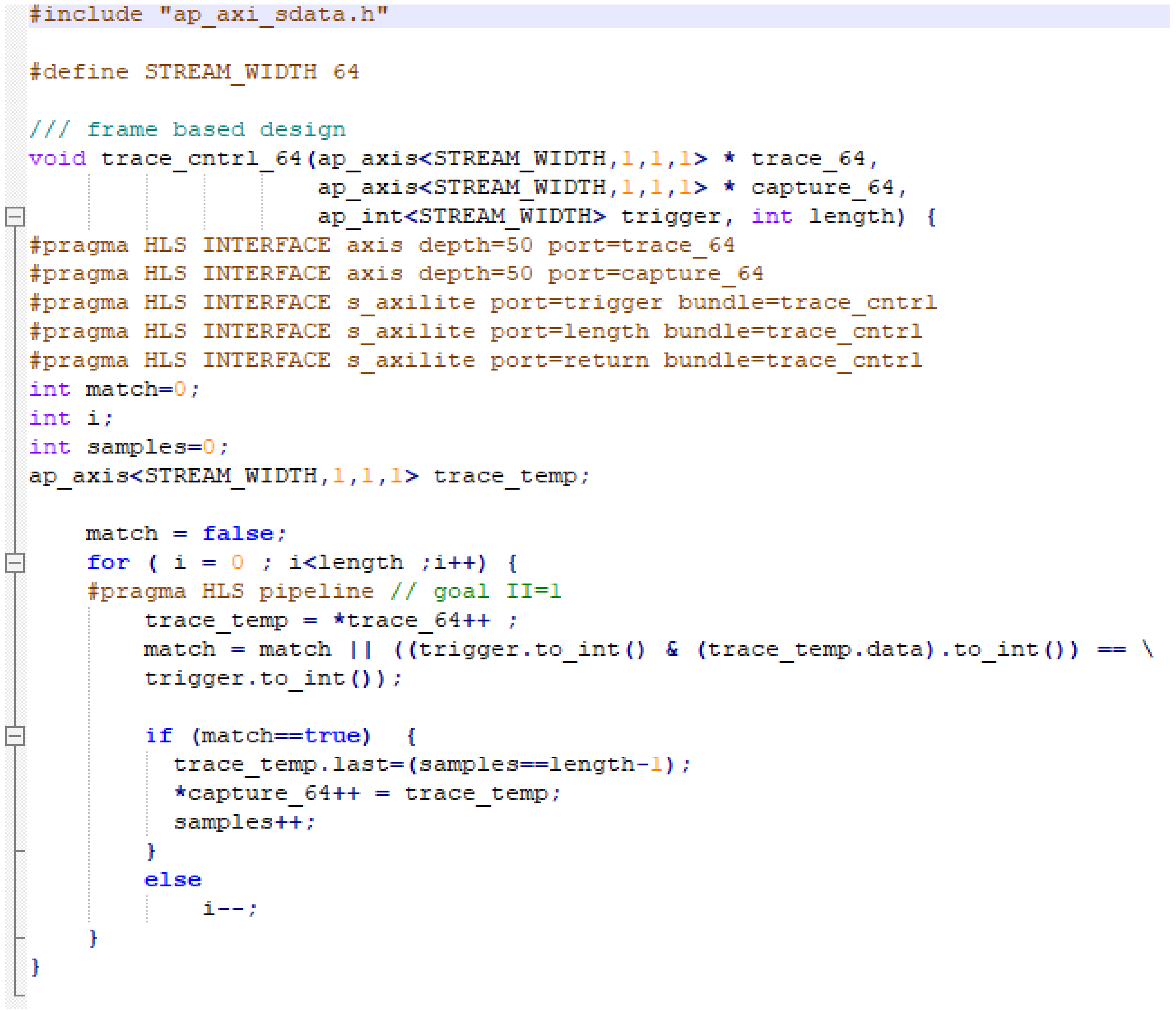


```
#include <ap_fixed.h>
#include <ap_int.h>
typedef ap_uint<8> pixel_type;
typedef ap_int<8> pixel_type_s;
typedef ap_ufixed<8,0, AP_RND, AP_SAT> comp_type;
typedef ap_fixed<10,2, AP_RND, AP_SAT> coeff_type;
struct video_stream {
    struct {
        pixel_type p1;
        pixel_type p2;
        pixel_type p3;
        pixel_type p4;
        pixel_type p5;
        pixel_type p6;
    } data;
    ap_uint<1> user;
    ap_uint<1> last;
};
struct coeffs {
    coeff_type c1;
    coeff_type c2;
    coeff_type c3;
};
void color_convert_2(video_stream* stream_in_48, video_stream* stream_out_48,
                        coeffs c1, coeffs c2, coeffs c3, coeffs bias) {
#pragma HLS CLOCK domain=default
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE s_axilite register port=c1 clock=control
#pragma HLS INTERFACE s_axilite register port=c2 clock=control
#pragma HLS INTERFACE s_axilite register port=c3 clock=control
#pragma HLS INTERFACE s_axilite register port=bias clock=control
#pragma HLS INTERFACE axis port=stream_in_48 register
#pragma HLS INTERFACE axis port=stream_out_48 register
#pragma HLS pipeline II=1
    stream_out_48->user = stream_in_48->user;
    stream_out_48->last = stream_in_48->last;
    comp_type in1, in2, in3, out1, out2, out3;
    comp_type in4, in5, in6, out4, out5, out6;
    in1.range() = stream_in_48->data.p1;
    in2.range() = stream_in_48->data.p2;
    in3.range() = stream_in_48->data.p3;
    in4.range() = stream_in_48->data.p4;
    in5.range() = stream_in_48->data.p5;
    in6.range() = stream_in_48->data.p6;

    out1 = in1 * c1.c1 + in2 * c1.c2 + in3 * c1.c3 + bias.c1;
    out2 = in1 * c2.c1 + in2 * c2.c2 + in3 * c2.c3 + bias.c2;
    out3 = in1 * c3.c1 + in2 * c3.c2 + in3 * c3.c3 + bias.c3;
    out4 = in4 * c1.c1 + in5 * c1.c2 + in6 * c1.c3 + bias.c1;
    out5 = in4 * c2.c1 + in5 * c2.c2 + in6 * c2.c3 + bias.c2;
    out6 = in4 * c3.c1 + in5 * c3.c2 + in6 * c3.c3 + bias.c3;

    stream_out_48->data.p1 = out1.range();
    stream_out_48->data.p2 = out2.range();
    stream_out_48->data.p3 = out3.range();
    stream_out_48->data.p4 = out4.range();
    stream_out_48->data.p5 = out5.range();
    stream_out_48->data.p6 = out6.range();
}
```





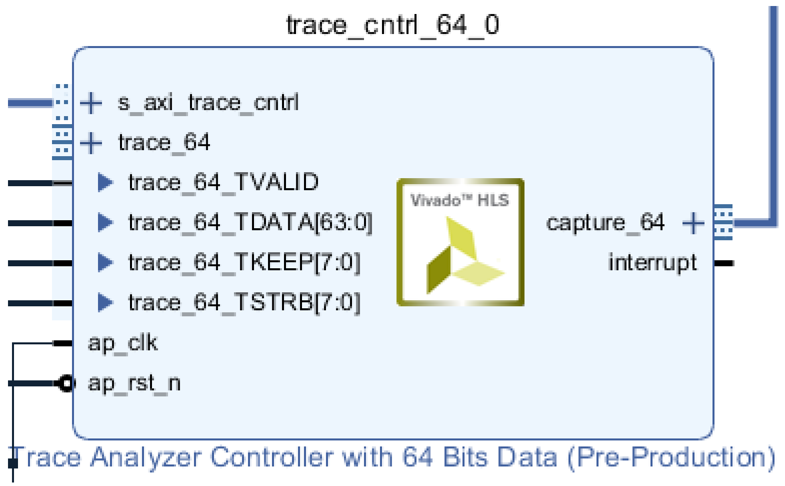
```
#include "ap_axi_sdata.h"

#define STREAM_WIDTH 64

/// frame based design
void trace_cntrl_64(ap_axis<STREAM_WIDTH,1,1,1> * trace_64,
                    ap_axis<STREAM_WIDTH,1,1,1> * capture_64,
                    ap_int<STREAM_WIDTH> trigger, int length) {
#pragma HLS INTERFACE axis depth=50 port=trace_64
#pragma HLS INTERFACE axis depth=50 port=capture_64
#pragma HLS INTERFACE s_axilite port=trigger bundle=trace_cntrl
#pragma HLS INTERFACE s_axilite port=length bundle=trace_cntrl
#pragma HLS INTERFACE s_axilite port=return bundle=trace_cntrl
int match=0;
int i;
int samples=0;
ap_axis<STREAM_WIDTH,1,1,1> trace_temp;

    match = false;
    for ( i = 0 ; i<length ;i++) {
    #pragma HLS pipeline // goal II=1
        trace_temp = *trace_64++ ;
        match = match || ((trigger.to_int() & (trace_temp.data).to_int()) == \
        trigger.to_int());

        if (match==true)  {
          trace_temp.last=(samples==length-1);
          *capture_64++ = trace_temp;
          samples++;
        }
        else
            i--;
    }
}
```



XILINX