Application Acceleration with High-Level Synthesis

GitHub：https://github.com/ZheChen-Bill/Interface_Synthesis Lab_A 111061545 陳揚哲

1. Block-Level Protocol

The block-level I/O protocol allow us control RTL design by the control signal, which is independent of data I/O ports. The block-level control ports contain ap_start, ap_ready, ap_done, ap_idle, and ap_continue.

ap_start	Controls block execution
ap_ready	Shows the design is ready for new input
ap_done	Shows the design has complete all operation In current transaction
ap_idle	Shows the design is operating or idle
ap_continue	Only valid in ap_ctrl_chain. It shows the downstream block is ready for new data inputs. In other words, the ap_ready of downstream block can drive ap_continue port.

In this lab, we change the block level protocol from ap_ctrl_hs, ap_ctrl_chain and ap_ctrl_none. We use the Vitis_HLS to change the block level protocol. In the C synthesis report, we can observe the RTL ports and the block-level protocol. In ap_ctrl_hs, we have ap_start, ap_ready, ap_done, and ap_idle. In ap_ctrl_chain, there is additional port ap_continue. In ap_ctrl_none, there aren't these RTL ports.

ap_ctrl_hs :

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_start	in	1	ap_ctrl_hs	adders	return value
ap_done	out	1	ap_ctrl_hs	adders	return value
ap_idle	out	1	ap_ctrl_hs	adders	return value
ap_ready	out	1	ap_ctrl_hs	adders	return value
ap_return	out	32	ap_ctrl_hs	adders	return value
in1	in	32	ap_none	in1	scalar
in2	in	32	ap_none	in2	scalar
in3	in	32	ap_none	in3	scalar

ap_ctrl_chain：

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_chain	adders	return value
ap_rst	in	1	ap_ctrl_chain	adders	return value
ap_start	in	1	ap_ctrl_chain	adders	return value
ap_done	out	1	ap_ctrl_chain	adders	return value
ap_continue	in	1	ap_ctrl_chain	adders	return value
ap_idle	out	1	ap_ctrl_chain	adders	return value
ap_ready	out	1	ap_ctrl_chain	adders	return value
ap_return	out	32	ap_ctrl_chain	adders	return value
in1	in	32	ap_none	in1	scalar
in2	in	32	ap_none	in2	scalar
in3	in	32	ap_none	in3	scalar

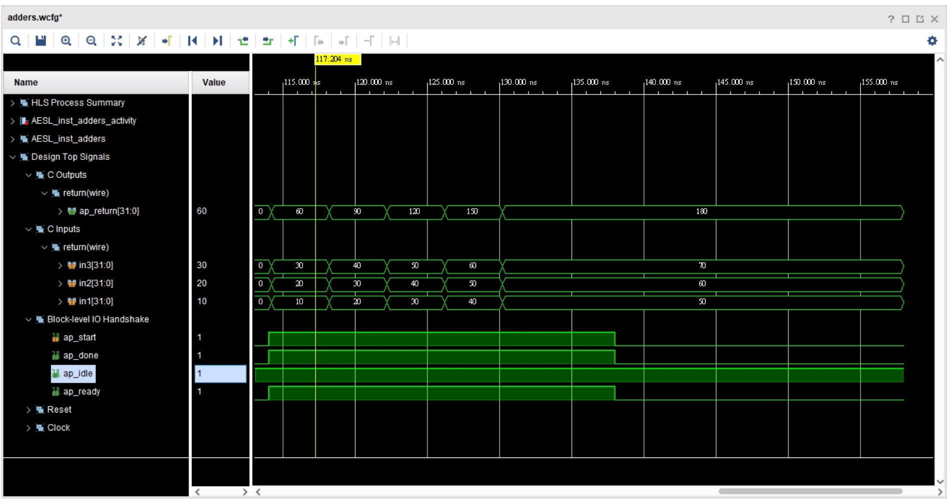
ap_ctrl_none：

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
in1	in	32	ap_none	in1	scalar
in2	in	32	ap_none	in2	scalar
in3	in	32	ap_none	in3	scalar
ap_return	out	32	ap_ctrl_none	adders	return value

The co-simulation waveform can demonstrate the relationship between block-level ports and the input and output port. If we use ap_ctrl_none, owing to co-simulation need block-level I/O protocol to sequence the test bench. However, in our case we can apply co-simulation. Since our design is viewed as combinational logic.



2. Port-level Protocol

The port-level I/O protocol is the data flow I/O ports. There’re three different kinds port-level protocol. The ap_none specifies no I/O protocol be added to the port.

No Protocol	ap_none
Wire Handshakes	ap_hs (ap_ack, ap_vld, ap_ovld)
Memory Interface Protocol	ap_memory, bram ap_fifo

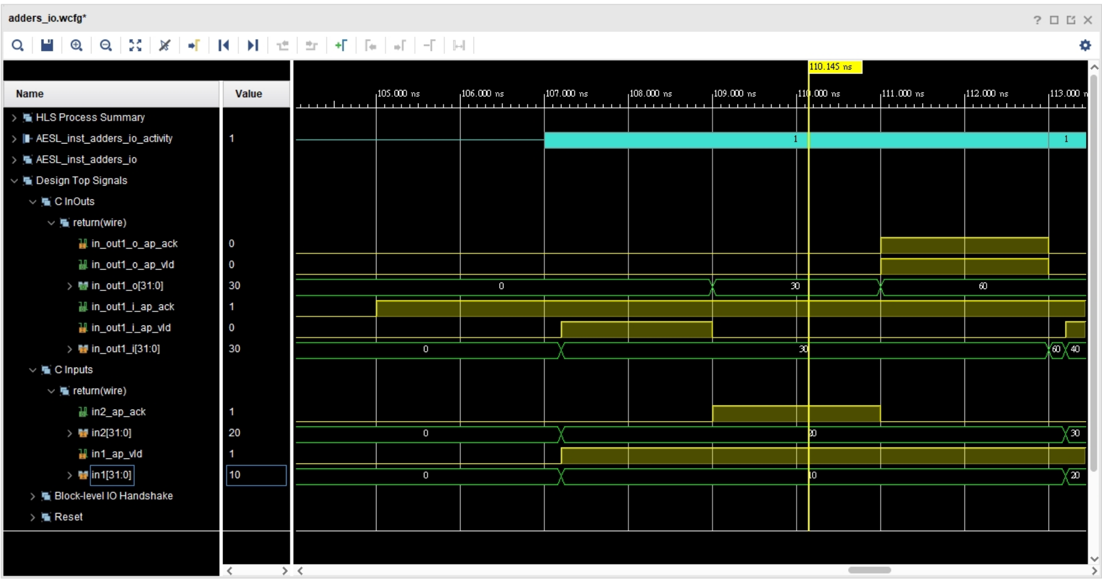
ap_none	The ap_none mode is the default for scalar inputs
ap_hs	Includes two-way handshake signal.
ap_ack	Only has a acknowledge port. Shows the data has been read. Input arguments： generates an output acknowledge. (the input is read.) Output arguments： implements an input acknowledge port to confirm the output was read.
ap_vld	Only has a valid port. Shows the data signal is valid and can be read. Input arguments： reads the data port as soon as the valid is active. Even if the design is not ready to read new data, the design samples the data port and holds the data internally until needed. Output arguments： implements an output valid port to indicate when the data on the output port is valid.
ap_ovld	Mode ap_none is applied to the input port and ap_vld applied to the output port. Input arguments： ap_none Output arguments： ap_vld

ap_memory, bram	Used to implement array arguments. This type of port-level I/O protocol can communicate with memory elements.
ap_fifo	Allows the port to be connected to a FIFO. Enables complete, two-way empty-full communication.

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	adders_io	return value
ap_rst	in	1	ap_ctrl_hs	adders_io	return value
ap_start	in	1	ap_ctrl_hs	adders_io	return value
ap_done	out	1	ap_ctrl_hs	adders_io	return value
ap_idle	out	1	ap_ctrl_hs	adders_io	return value
ap_ready	out	1	ap_ctrl_hs	adders_io	return value
in1	in	32	ap_vld	in1	scalar
in1_ap_vld	in	1	ap_vld	in1	scalar
in2	in	32	ap_ack	in2	scalar
in2_ap_ack	out	1	ap_ack	in2	scalar
in_out1_i	in	32	ap_hs	in_out1	pointer
in_out1_i_ap_vld	in	1	ap_hs	in_out1	pointer
in_out1_i_ap_ack	out	1	ap_hs	in_out1	pointer
in_out1_o	out	32	ap_hs	in_out1	pointer
in_out1_o_ap_vld	out	1	ap_hs	in_out1	pointer
in_out1_o_ap_ack	in	1	ap_hs	in_out1	pointer



1. The data on port in1 is only read when port in1_ap_vld is active-High.
2. Port in2_ap_ack will be active-High when data port in2 is read.
3. `inout_i` associated input valid port `inout1_i_ap_vld` and output acknowledge port `inout1_i_ap_ack`.
4. `inout_o` associated output valid port `inout1_o_ap_vld` and input acknowledge port inout1_o_ap_ack.

3. Array Interface

This lab specifies implement different type of RTL port for array arguments. We use single port, two-port RAM and FIFO array interface, array partition RAM and FIFO array interface and fully partition array interface.

If we use two-port RAM interface, this design can accept input data at twice rate of single port. Therefore, the estimated time of solution2(two port) are lower.

Performance Estimates

⊟ Timing

Clock		solution1	solution2
ap_clk	Target	4.00 ns	4.00 ns
	Estimated	2.602 ns	2.581 ns

⊟ Latency

		solution1	solution2
Latency (cycles)	min	34	33
	max	34	33
Latency (absolute)	min	0.136 us	0.132 us
	max	0.136 us	0.132 us
Interval (cycles)	min	35	34
	max	35	34

Solution 1: single port

Solution 2: two port unroll

Single port：

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	array_io	return value
ap_rst	in	1	ap_ctrl_hs	array_io	return value
ap_start	in	1	ap_ctrl_hs	array_io	return value
ap_done	out	1	ap_ctrl_hs	array_io	return value
ap_idle	out	1	ap_ctrl_hs	array_io	return value
ap_ready	out	1	ap_ctrl_hs	array_io	return value
d_o_address0	out	5	ap_memory	d_o	array
d_o_ce0	out	1	ap_memory	d_o	array
d_o_we0	out	1	ap_memory	d_o	array
d_o_d0	out	16	ap_memory	d_o	array
d_i_address0	out	5	ap_memory	d_i	array
d_i_ce0	out	1	ap_memory	d_i	array
d_i_q0	in	16	ap_memory	d_i	array

Two Port unroll：

Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	array_io	return value
ap_rst	in	1	ap_ctrl_hs	array_io	return value
ap_start	in	1	ap_ctrl_hs	array_io	return value
ap_done	out	1	ap_ctrl_hs	array_io	return value
ap_idle	out	1	ap_ctrl_hs	array_io	return value
ap_ready	out	1	ap_ctrl_hs	array_io	return value
d_o_din	out	16	ap_fifo	d_o	pointer
d_o_full_n	in	1	ap_fifo	d_o	pointer
d_o_write	out	1	ap_fifo	d_o	pointer
d_i_address0	out	5	ap_memory	d_i	array
d_i_ce0	out	1	ap_memory	d_i	array
d_i_q0	in	16	ap_memory	d_i	array
d_i_address1	out	5	ap_memory	d_i	array
d_i_ce1	out	1	ap_memory	d_i	array
d_i_q1	in	16	ap_memory	d_i	array

In rolled loop, loop is executed in turn. This implementation code limits the logic to one read on d_i in each iteration. In other word, reading d_i once in each loop. Thus, the estimated time wouldn't be faster (solution5).

Performance Estimates

⊟ Timing

Clock		solution1	solution2	solution5
ap_clk	Target	4.00 ns	4.00 ns	4.00 ns
	Estimated	2.602 ns	2.581 ns	2.602 ns

⊟ Latency

		solution1	solution2	solution5
Latency (cycles)	min	34	33	35
	max	34	33	35
Latency (absolute)	min	0.136 us	0.132 us	0.140 us
	max	0.136 us	0.132 us	0.140 us
Interval (cycles)	min	35	34	36
	max	35	34	36

Solution 1: single port

Solution 2: two port unroll

Solution 5: two port roll

In array partition, the input d_i will separate into the numbers of array partition factor RAM interfaces. The output d_o will be divided into the numbers of array partition FIFO interfaces. In this lab, we divide d_i into 2 seperated RAM interfaces and d_o into 4 FIFO interfaces.

If we change the factor of d_i and d_o. we should first know the d_i = 2, d_o =4 means there're two separate two port ram. If we change into d_i = 2, d_o

=2 or d_i = 4, d_o =4. These situations only have the single port interface. Further, if we set the d_o = 4, it means the output port only output 4 values at once. Thus, it's no benefit for reading inputs higher than 4.

Performance Estimates

Timing

Clock		solution3	solution6	solution8	solution9
ap_clk	Target	4.00 ns	4.00 ns	4.00 ns	4.00 ns
	Estimated	2.581 ns	2.759 ns	2.759 ns	2.759 ns

Latency

		solution3	solution6	solution8	solution9
Latency (cycles)	min	10	18	10	18
	max	10	18	10	18
Latency (absolute)	min	40.000 ns	72.000 ns	40.000 ns	72.000 ns
	max	40.000 ns	72.000 ns	40.000 ns	72.000 ns
Interval (cycles)	min	11	19	11	19
	max	11	19	11	19

Solution 3: d_i = 2, d_o = 4
Solution 6: d_i = 2, d_o = 2
Solution 8: d_i = 4, d_o = 4
Solution 9: d_i = 4, d_o = 2

If we choose complete, it will separate the array into 32 pieces. The d_o has been separated into 32 FIFO interfaces. Since the d_i is separated into 32 separate scalar ports. They use the I/O protocol ap_none for each scalar data. The estimated time is smaller.

d_i_0	in	16	ap_none	d_i_0	pointer
d_i_1	in	16	ap_none	d_i_1	pointer
d_i_2	in	16	ap_none	d_i_2	pointer
d_i_3	in	16	ap_none	d_i_3	pointer
d_i_4	in	16	ap_none	d_i_4	pointer
d_i_5	in	16	ap_none	d_i_5	pointer
d_i_6	in	16	ap_none	d_i_6	pointer
d_i_7	in	16	ap_none	d_i_7	pointer
d_i_8	in	16	ap_none	d_i_8	pointer
d_i_9	in	16	ap_none	d_i_9	pointer
d_i_10	in	16	ap_none	d_i_10	pointer
d_i_11	in	16	ap_none	d_i_11	pointer
d_i_12	in	16	ap_none	d_i_12	pointer
d_i_13	in	16	ap_none	d_i_13	pointer
d_i_14	in	16	ap_none	d_i_14	pointer
d_i_15	in	16	ap_none	d_i_15	pointer
d_i_16	in	16	ap_none	d_i_16	pointer
d_i_17	in	16	ap_none	d_i_17	pointer
d_i_18	in	16	ap_none	d_i_18	pointer
d_i_19	in	16	ap_none	d_i_19	pointer
d_i_20	in	16	ap_none	d_i_20	pointer
d_i_21	in	16	ap_none	d_i_21	pointer
d_i_22	in	16	ap_none	d_i_22	pointer

Performance Estimates

Timing

Clock		solution3	solution6	solution8	solution9	solution4
ap_clk	Target	4.00 ns	4.00 ns	4.00 ns	4.00 ns	4.00 ns
	Estimated	2.581 ns	2.759 ns	2.759 ns	2.759 ns	2.243 ns

Latency

		solution3	solution6	solution8	solution9	solution4
Latency (cycles)	min	10	18	10	18	1
	max	10	18	10	18	1
Latency (absolute)	min	40.000 ns	72.000 ns	40.000 ns	72.000 ns	4.000 ns
	max	40.000 ns	72.000 ns	40.000 ns	72.000 ns	4.000 ns
Interval (cycles)	min	11	19	11	19	2
	max	11	19	11	19	2

Solution 3: d_i = 2, d_o = 4 (block)

Solution 6: d_i = 2, d_o = 2 (block)

Solution 8: d_i = 4, d_o = 4 (block)

Solution 9: d_i = 4, d_o = 2 (block)

Solution 4: d_i = complete, d_o = complete

For cyclic type, the smaller arrays are generated by interleaving elements from original array. In this situation, the cyclic type is faster than block type.

Performance Estimates

Timing

Clock		solution3	solution6	solution8	solution9	solution4
ap_clk	Target	4.00 ns	4.00 ns	4.00 ns	4.00 ns	4.00 ns
	Estimated	2.581 ns	2.581 ns	2.581 ns	2.581 ns	2.243 ns

Latency

		solution3	solution6	solution8	solution9	solution4
Latency (cycles)	min	9	17	9	17	1
	max	9	17	9	17	1
Latency (absolute)	min	36.000 ns	68.000 ns	36.000 ns	68.000 ns	4.000 ns
	max	36.000 ns	68.000 ns	36.000 ns	68.000 ns	4.000 ns
Interval (cycles)	min	10	18	10	18	2
	max	10	18	10	18	2

Solution 3: d_i = 2, d_o = 4 (cyclic)

Solution 6: d_i = 2, d_o = 2 (cyclic)

Solution 8: d_i = 4, d_o = 4 (cyclic)

Solution 9: d_i = 4, d_o = 2 (cyclic)

Solution 4: d_i = complete, d_o = complete

4. AXI Interface

We separate the into d_o = 8 and d_i = 8 in axis protocol (AXI4-stream). Compared to lab2, we can observe the data separation.

d_i_0_TVALID	in	1	axis	d_i_0	pointer
d_i_0_TDATA	in	16	axis	d_i_0	pointer
d_i_0_TREADY	out	1	axis	d_i_0	pointer
d_o_0_TREADY	in	1	axis	d_o_0	pointer
d_o_0_TDATA	out	16	axis	d_o_0	pointer
d_o_0_TVALID	out	1	axis	d_o_0	pointer
d_i_1_TVALID	in	1	axis	d_i_1	pointer
d_i_1_TDATA	in	16	axis	d_i_1	pointer
d_i_1_TREADY	out	1	axis	d_i_1	pointer
d_o_1_TREADY	in	1	axis	d_o_1	pointer
d_o_1_TDATA	out	16	axis	d_o_1	pointer
d_o_1_TVALID	out	1	axis	d_o_1	pointer
d_i_2_TVALID	in	1	axis	d_i_2	pointer
d_i_2_TDATA	in	16	axis	d_i_2	pointer
d_i_2_TREADY	out	1	axis	d_i_2	pointer
d_o_2_TREADY	in	1	axis	d_o_2	pointer
d_o_2_TDATA	out	16	axis	d_o_2	pointer
d_o_2_TVALID	out	1	axis	d_o_2	pointer
d_i_3_TVALID	in	1	axis	d_i_3	pointer
d_i_3_TDATA	in	16	axis	d_i_3	pointer
d_i_3_TREADY	out	1	axis	d_i_3	pointer
d_o_3_TREADY	in	1	axis	d_o_3	pointer
d_o_3_TDATA	out	16	axis	d_o_3	pointer
d_o_3_TVALID	out	1	axis	d_o_3	pointer
d_i_4_TVALID	in	1	axis	d_i_4	pointer
d_i_4_TDATA	in	16	axis	d_i_4	pointer
d_i_4_TREADY	out	1	axis	d_i_4	pointer
d_o_4_TREADY	in	1	axis	d_o_4	pointer
d_o_4_TDATA	out	16	axis	d_o_4	pointer
d_o_4_TVALID	out	1	axis	d_o_4	pointer
d_i_5_TVALID	in	1	axis	d_i_5	pointer
d_i_5_TDATA	in	16	axis	d_i_5	pointer
d_i_5_TREADY	out	1	axis	d_i_5	pointer
d_o_5_TREADY	in	1	axis	d_o_5	pointer
d_o_5_TDATA	out	16	axis	d_o_5	pointer
d_o_5_TVALID	out	1	axis	d_o_5	pointer
d_i_6_TVALID	in	1	axis	d_i_6	pointer

pstrmInput_TDATA	in	32	axis	pstrmInput_V_data_V	pointer
pstrmInput_TVALID	in	1	axis	pstrmInput_V_dest_V	pointer
pstrmInput_TREADY	out	1	axis	pstrmInput_V_dest_V	pointer
pstrmInput_TDEST	in	1	axis	pstrmInput_V_dest_V	pointer
pstrmInput_TKEEP	in	4	axis	pstrmInput_V_keep_V	pointer
pstrmInput_TSTRB	in	4	axis	pstrmInput_V_strb_V	pointer
pstrmInput_TUSER	in	1	axis	pstrmInput_V_user_V	pointer
pstrmInput_TLAST	in	1	axis	pstrmInput_V_last_V	pointer
pstrmInput_TID	in	1	axis	pstrmInput_V_id_V	pointer
pstrmOutput_TDATA	out	32	axis	pstrmOutput_V_data_V	pointer
pstrmOutput_TVALID	out	1	axis	pstrmOutput_V_dest_V	pointer
pstrmOutput_TREADY	in	1	axis	pstrmOutput_V_dest_V	pointer
pstrmOutput_TDEST	out	1	axis	pstrmOutput_V_dest_V	pointer
pstrmOutput_TKEEP	out	4	axis	pstrmOutput_V_keep_V	pointer
pstrmOutput_TSTRB	out	4	axis	pstrmOutput_V_strb_V	pointer
pstrmOutput_TUSER	out	1	axis	pstrmOutput_V_user_V	pointer
pstrmOutput_TLAST	out	1	axis	pstrmOutput_V_last_V	pointer
pstrmOutput_TID	out	1	axis	pstrmOutput_V_id_V	pointer

To compare the design area, we should look up the information related to the utilization of different FPGA resources such as LUTs (lookup tables), DSPs (digital signal processors), and BRAM (block RAM).

Lab 4:

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	523	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	316	-
Register	-	-	527	-	-
Total	0	0	527	839	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	0	~0	~0	0
Utilization SLR (%)	0	0	~0	~0	0

Lab 3:

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1640	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	478	-
Register	-	-	1274	-	-
Total	0	0	1274	2118	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	0	~0	~0	0
Utilization SLR (%)	0	0	~0	~0	0

For the same code, the synthesis will generate the same system. If we compare the area, the Lab 4 design generate better efficiency. Cyclic partitioning of the array interfaces and partial for-loop unrolling has allowed implementation of this code as eight separate channels in the hardware. Pipelining the for-loop allows the logic in each channel to process 1 sample per clock. Varying the partitioning and loop unrolling create a design which is the optimal balance of area and performance to satisfy this requirements.

If we open the hw.h file. This file shows the addresses to access and control the block-level interface signals. We can set the bit 0 to value 1, the ap_start will be enabled. It shows how host program control the Axilite.

```
// ==============================================================
// Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.1 (64-bit)
// Tool Version Limit: 2022.04
// Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
// ==============================================================
// control
// 0x0 : Control signals
//       bit 0  - ap_start (Read/Write/COH)
//       bit 1  - ap_done (Read/COR)
//       bit 2  - ap_idle (Read)
//       bit 3  - ap_ready (Read/COR)
//       bit 7  - auto_restart (Read/Write)
//       bit 9  - interrupt (Read)
//       others - reserved
// 0x4 : Global Interrupt Enable Register
//       bit 0  - Global Interrupt Enable (Read/Write)
//       others - reserved
// 0x8 : IP Interrupt Enable Register (Read/Write)
//       bit 0 - enable ap_done interrupt (Read/Write)
//       bit 1 - enable ap_ready interrupt (Read/Write)
//       others - reserved
// 0xc : IP Interrupt Status Register (Read/COR)
//       bit 0 - ap_done (Read/COR)
//       bit 1 - ap_ready (Read/COR)
//       others - reserved
// (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)

#define XAXI_INTERFACES_CONTROL_ADDR_AP_CTRL 0x0
#define XAXI_INTERFACES_CONTROL_ADDR_GIE     0x4
#define XAXI_INTERFACES_CONTROL_ADDR_IER     0x8
#define XAXI_INTERFACES_CONTROL_ADDR_ISR     0xc
```