Graduate Institute of Electronics Engineering, NTU

HLS Lab A: Interface Synthesis

Speaker: 王景平

Date: 2022/10/20

Lab #1 – Block-Level Protocol

Lab #1 - Default protocol

- A simple adder function without assigning any pragma
- Default protocol
  - Block-level protocol: ap_ctrl_hs
  - Port-level protocol: ap_none

```
#include "adders.h"

int adders(int in1, int in2, int in3) {
    int sum;
    sum = in1 + in2 + in3;
    return sum;
}
```

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

Lab #1 – Specified Block-Level Protocol

- Specify block-level protocol
  - #pragma HLS INTERFACE <protocol> port=return

```
#include "adders.h"

int adders(int in1, int in2, int in3) {
#pragma HLS INTERFACE ap_ctrl_hs port=return
    int sum;
    sum = in1 + in2 + in3;
    return sum;
}
```

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

Lab #1 – Various Protocol

- Available block-level protocols
  - ap_ctrl_none
  - ap_ctrl_hs
  - ap_ctrl_chain
  - s_axilite

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

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

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

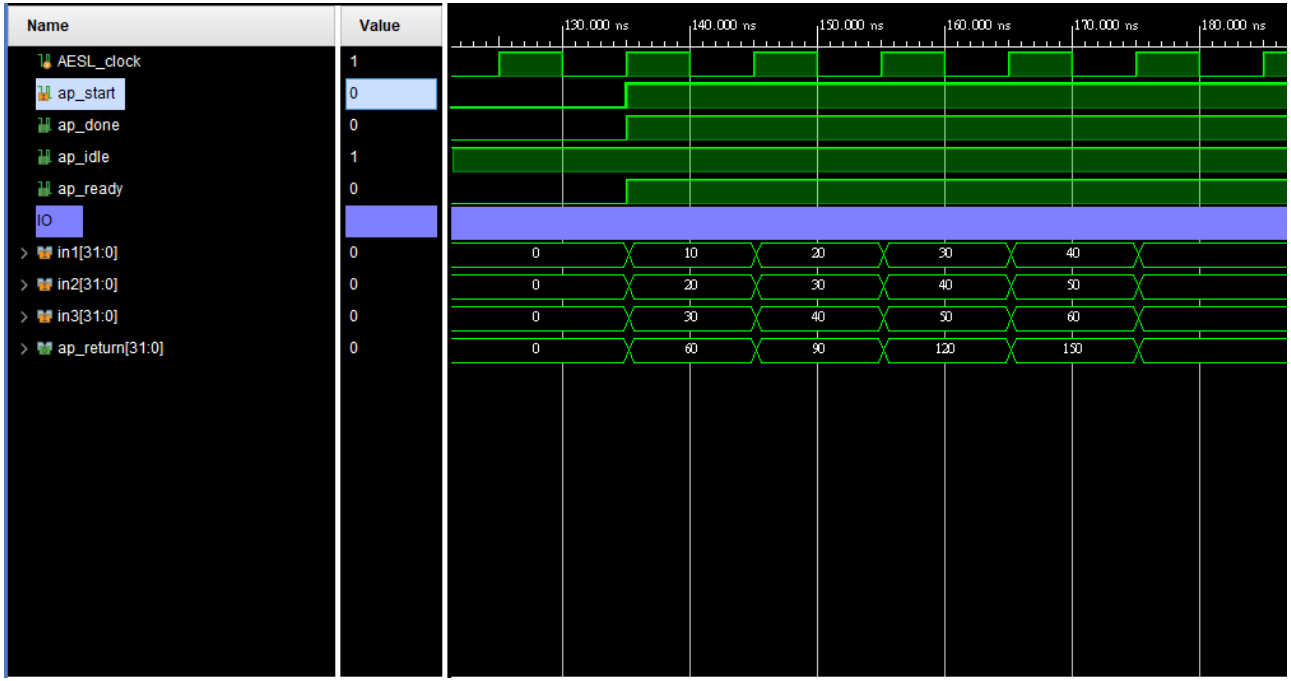
Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_control_AWVALID	in	1	s_axi	control	return value
s_axi_control_AWREADY	out	1	s_axi	control	return value
s_axi_control_AWADDR	in	5	s_axi	control	return value
s_axi_control_WVALID	in	1	s_axi	control	return value
s_axi_control_WREADY	out	1	s_axi	control	return value
s_axi_control_WDATA	in	32	s_axi	control	return value
s_axi_control_WSTRB	in	4	s_axi	control	return value
s_axi_control_ARVALID	in	1	s_axi	control	return value
s_axi_control_ARREADY	out	1	s_axi	control	return value
s_axi_control_ARADDR	in	5	s_axi	control	return value
s_axi_control_RVALID	out	1	s_axi	control	return value
s_axi_control_RREADY	in	1	s_axi	control	return value
s_axi_control_RDATA	out	32	s_axi	control	return value
s_axi_control_RRESP	out	2	s_axi	control	return value
s_axi_control_BVALID	out	1	s_axi	control	return value
s_axi_control_BREADY	in	1	s_axi	control	return value
s_axi_control_BRESP	out	2	s_axi	control	return value
in1	in	32	ap_none	in1	scalar
in2	in	32	ap_none	in2	scalar
in3	in	32	ap_none	in3	scalar
ap_clk	in	1	ap_ctrl_hs	adders	return value
ap_rst_n	in	1	ap_ctrl_hs	adders	return value
interrupt	out	1	ap_ctrl_hs	adders	return value

Lab #1 – ap_ctrl_hs Waveform

- ap_start: Input by previous stage, start the fuction
- ap_done: Indicate the result is valid
- ap_idle: Indicate the current status of the function
- ap_ready: Ready for the next input



Always idle due to combinational circuit

Lab #1 – ap_ctrl_none Co-Simulation

- ap_ctrl_none is valid for co-simulation
  - Combinational circuit
  - Pipeline with task interval=1
  - Array streaming or AXI4 stream ports

- Adder function is a combinational circuit
  - No co-simulation error

Lab #2 – Port-Level Protocol

Lab #2 – Port-Level Protocols

Pass-By-Value	Pass-By-Reference
ap_none	ap_none
ap_stable	ap_stable
ap_ack	ap_ack
ap_vld	ap_vld
ap_hs	ap_hs
	ap_ovld
	ap_fifo
	ap_bus

Lab #2 – Interfaces and Waveform

- Protocols
  - in1: ap_vld, inform next stage the data is valid
  - in2: ap_ack, inform previous stage the data is received
  - in_out1: ap_hs, both

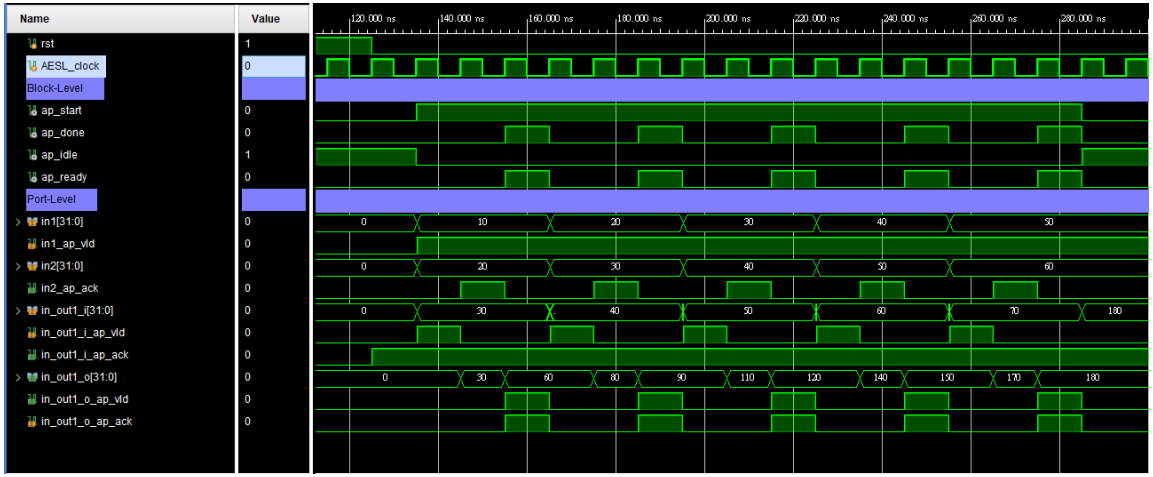
Interface

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

2 channels for reference type(in_out1)

```
#include "adders_io.h“
void adders_io(int in1, int in2, int *in_out1) {
    *in_out1 = in1 + in2 + *in_out1;
}
```



Lab #3 – Array Interface

Lab #3 – Dual-Port with Rolled Loop

- Loop remains rolled
- Dual-port ram for input d_i
- Fifo for output d_o

Dual-port ram is available

```
set_directive_top -name array_io "array_io"
set_directive_interface -mode ap_fifo "array_io" d_o
set_directive_interface -mode ap_memory -storage_impl bram -storage_type ram_2p "array_io" d_i
```

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

```
void array_io (dout_t d_o[N], din_t d_i[N]) {
    int i, rem;
    static dacc_t acc[CHANNELS];
    dacc_t temp;
    For_Loop: for (i=0;i<N;i++) {
        rem=i%CHANNELS;
        temp = acc[rem] + d_i[i];
        acc[rem] = temp;
        d_o[i] = acc[rem];
    }
}
```

Only single-port is used in the design

Lab #3 – Single-Port vs. Dual-Port

- Loop unrolled
- Input/output are either single-port or dual-port

		Input	
		Single Port	**Dual Port**
Output	**Single Port**	320	330
	Dual Port	320	170

Any part in the design could be a bottleneck, and cause performance degradation

Lab #3 – Array Partition

- Loop unrolled
- Array partitioned (e.g. 13 elements with factor 4)
  - Block: (0, 1, 2), (3, 4, 5), (6, 7, 8), (9, 10, 11, 12)
  - Cyclic: (0, 4, 8, 12), (1, 5, 9), (2, 6, 10), (3, 7, 11)
  - Complete: (0), (1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11)

		Factor	
		2	**4**
Type	**Block**	170	100
	Cyclic	170	90
	Complete	10	

Each type has different latency according to the dataflow

Complete array partition leads to lowest latency,
but highest resource usage

Lab #4 – AXI Interface

Lab #4 - Stream Interface

- Loop unrolled
- Port-level protocols are set to axis

AXIS

Interface	Register Mode	TDATA	TREADY	TVALID	
d_i_0	both	16	1	1	
d_i_1	both	16	1	1	
d_i_2	both	16	1	1	
d_i_3	both	16	1	1	
d_i_4	both	16	1	1	
d_i_5	both	16	1	1	
d_i_6	both	16	1	1	
d_i_7	both	16	1	1	
d_o_0	both	16	1	1	
d_o_1	both	16	1	1	
d_o_2	both	16	1	1	
d_o_3	both	16	1	1	
d_o_4	both	16	1	1	
d_o_5	both	16	1	1	
d_o_6	both	16	1	1	
d_o_7	both	16	1	1	

```
void axi_interfaces (dout_t d_o[N], din_t d_i[N]) {
    int i, rem;
    // CHANNELS = 8
    static dacc_t acc[CHANNELS];
    For_Loop: for (i=0;i<N;i++) {
        rem=i%CHANNELS;
        acc[rem] = acc[rem] + d_i[i];
        d_o[i] = acc[rem];
    }
}
```

With the loop unrolled, 8 channels are created for both input and output

Lab #4 – Resource Usage

- Lab3 > Lab4 > Lab2 > Lab1
  - From Lab1 to Lab4, peripheral circuits are add
  - Lab3 use complete array partition, consume the most resource

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
adders				-	0	0.0	-	1	-	no	0	0	0	64	0

Lab 1

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
adders_io				-	2	20.000	-	3	-	no	0	0	36	144	0

Lab 2

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
array_io				-	1	10.000	-	2	-	no	0	0	530	1654	0

Lab 3

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
axi_interfaces				-	5	50.000	-	4	-	loop rewind(delay=0 clock cycles(s))	0	0	527	842	0
For_Loop				-	4	40.000	2	1	4	yes	-	-	-	-	-

Lab 4

Lab #4 – Content in hw.h File

- Memory mapping information about control signals

```
// ==========================================================
// 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
```

Conclusion

- Different protocols result in various number of ports
- The number of accesses to RAM in one cycle is the main bottleneck to the performance of a system
- To increase the number of accesses in one cycle, several methods can be applied:
  - Dual-port memory
  - Array partition
  - Streaming
- The overall throughput is determined by each parts in the system, and bottleneck can appear everywhere

Questions

- Why the adder function with ap_ctrl_none won't cause the co-simulation fail?
- Why doesn't the dual-port RAM be used when the loop is not unrolled?