Systemverilog Assertions and Formal Verification (Winter 2021)
UCSC Extension
VLSI.X411.(7)
6h January 2021

Develop Assertion IP for H2A protocol

OVERVIEW

As a project for the course - Systemverilog Assertions and Formal Verification (Winter 2021), we are going to build an assertion IP for a communication protocol - H2A (host to accelerator). This document provides the specifications for the protocol.

GOALS

1. Learn the process of building Assertion IP.
2. Write assertion code for H2A protocol.
3. Debug and run the assertions in Formal Verification tool.

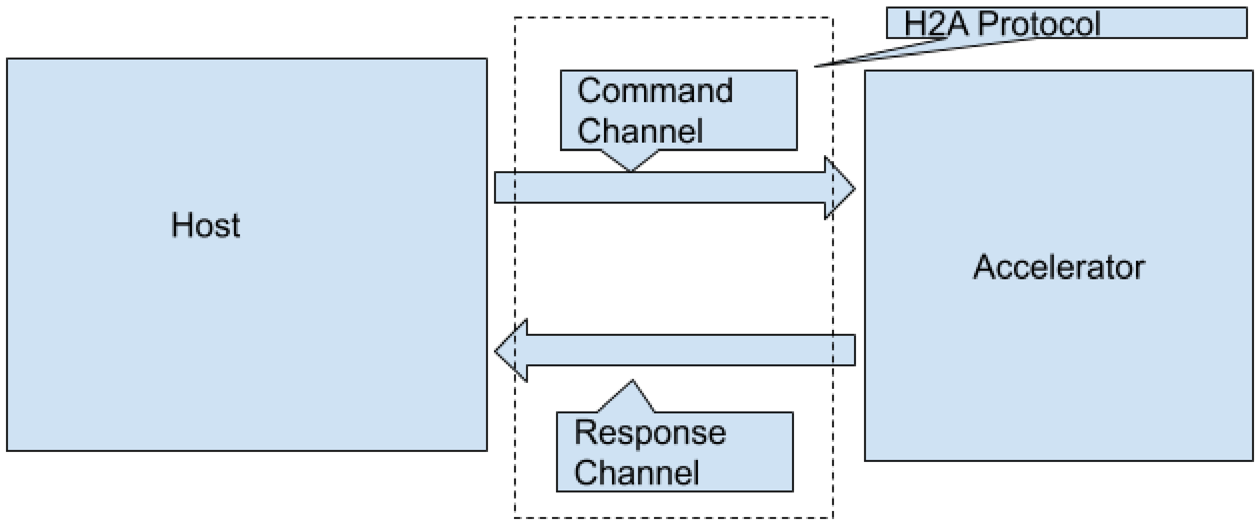
SPECIFICATIONS

H2A protocol is envisioned to be used between a host device (IP) and an accelerator device.

This is a point to point communication protocol. Host is assumed to be the master and the accelerator is a client device. This protocol assumes that the blocks interfacing with this protocol are using a single clock and reset scheme across both the master and client devices.

As shown in the following diagram, H2A protocol consists of two channels.

- Command Channel
- Response Channel



Command Channel

Command channel is used by the host to transfer commands and data to the accelerator. It follows the conventions of ready/valid flow control protocol.

In a ready/valid protocol, “vld” signal is asserted by the agent initiating transaction. If “rdy” is not asserted, “vld” if asserted should keep asserted and “cmd” and “operand” signals should remain stable. The transaction completes when both “vld” and “rdy” are asserted in the same cycle.

Following table describes the signals involved in the command channel.

Signal Name	Bit Width	Direction (host point of view)	Description
vld	1	output	When host wants to initiate a command to accelerator it activates this signal
cmd	3	output	
operand1	64	output	
operand2	64	output	
rdy	1	input	When accelerator is ready to accept the command it asserts ready

Response Channel

Accelerator communicates with the host through the response channel. Whenever a command is processed to completion, it asserts the “done” pulse. There is an optional done_cmd signal which may be used when the accelerator is capable of out-of-order processing to indicate which command was completed.

Signal Name	Bit Width	Direction (host point of view)	Description
done	1	input	When command is completed, accelerator with assert this signal as a pulse
done_cmd	3	input	Optional signal to indicate which command was completed

Cmd

There are 8 possible commands

RST, INIT, ADD, SUB, MULT, DIV, REM, HLT

Note : ADD, SUB, MULT, DIV, REM are collectively called OP commands because they operate on the operands provided. RST, INIT, HLT are called control commands as they control the accelerator itself.

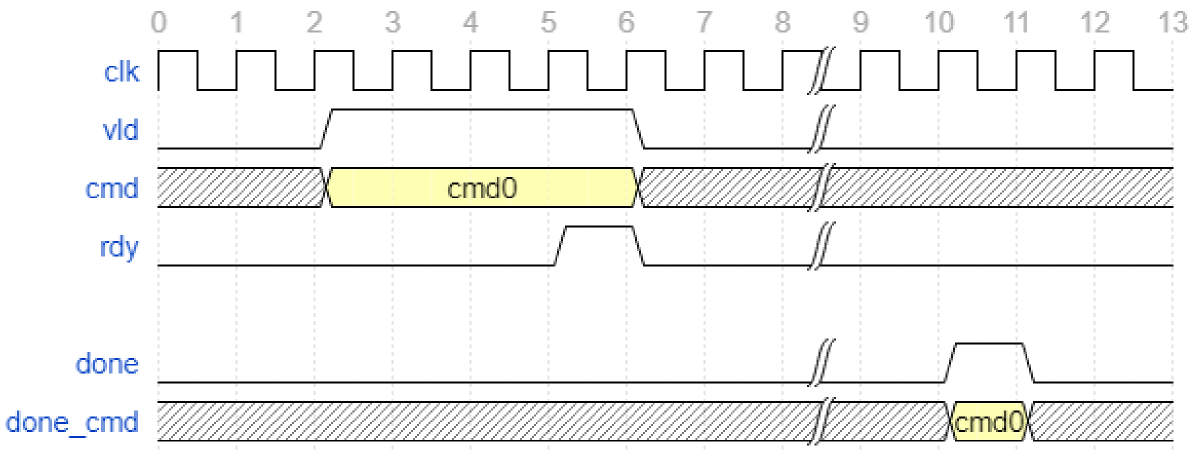
Theory of operation

- When design comes out of reset, the host must issue an INIT command before issuing any of the OP or HLT commands.
- After issuing an HLT command, the host must issue an RST followed by an INIT command before issuing any OP commands.
- HLT commands can only be issued when there are no outstanding or pending commands in the accelerator.
- There cannot be more than one command of the same type, be outstanding.

- The accelerator must assert a “ready” (rdy) signal in response to “valid” (vld) within a certain predetermined LATENCY number of cycles, otherwise it will be treated as a timeout.
- Signal “done” indicates that the accelerator has finished processing a command. There is no requirement for - when the “done” signal is asserted, but it must be asserted eventually. Depending on whether the accelerator handles commands “inorder or out-of-order”, there is additional signal on the interface as described below.
    - If the accelerator processes commands “inorder”, then there is no need for done_cmd signal
    - If the accelerator processes commands “out-of-order”, then the signal “done_cmd” accompanies the “done” signal and indicates which command was completed.

Timing Diagram

Following waveforms describe a sample transaction.



MILESTONES

M1. Prepare Assertion Plan

M2. Write SVA code

M3. Debug and run Formal Verification