ECE 260C: SV-UVM based Verification Environment Development for YAPP Router

Mahima Rathore | Richa Pallavi

Project Overview:

This project aims at developing a verification environment for a router design. The design under test (DUT) is called yapp router, where yapp stands for "yet another packet protocol". Due to the limited time period of a month for this project, our focus was development of a well-structured and detailed UVM verification environment, instead of thorough testing and coverage. Mentioned below are the details of this design:

High level diagram of YAPP-Router:

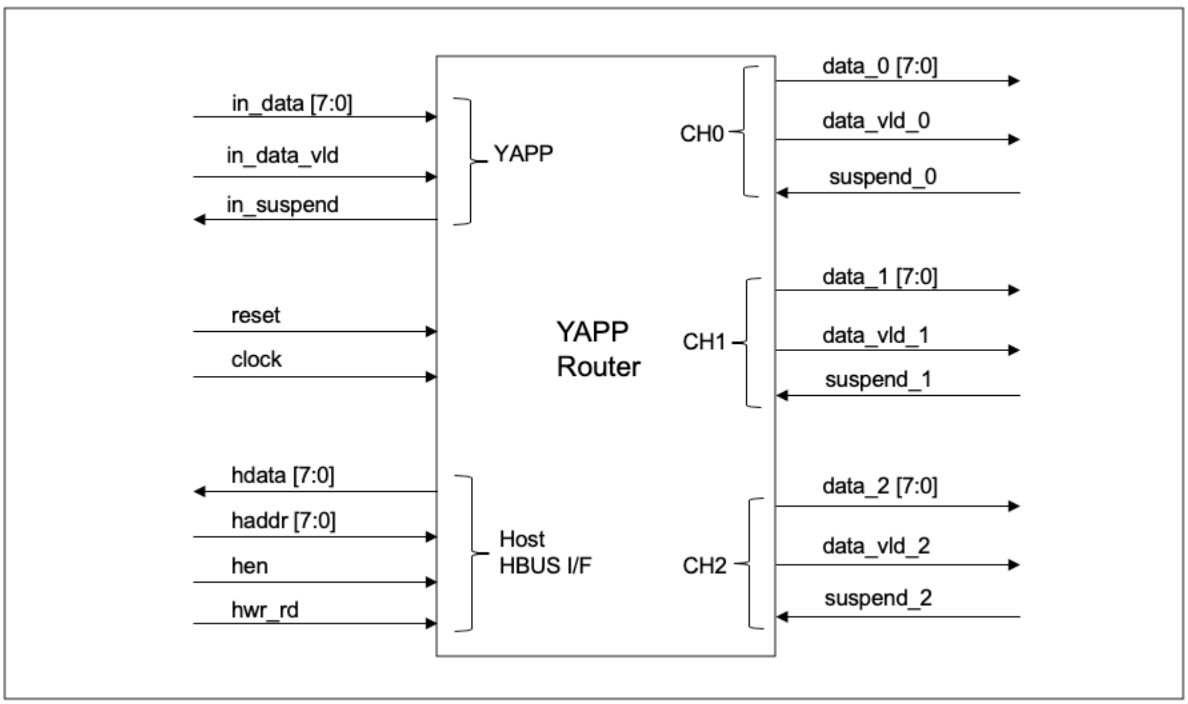


Fig1: YAPP Router DUT

Packet router description:

The packet router accepts data packets on a single input port, in_data, and routes the packets to one of three output channels: channel 0, channel 1 or channel 2. The input and output ports have slightly different signal protocols. The router also has a host interface for programming registers that are described next.

Packet data specification:

A packet is a sequence of bytes with the first byte containing a header, the next variable set of bytes containing payload, and the last byte containing parity.
The header consists of a two-bit address field and six-bit length field. The address field is used to determine which output channel the packet should be routed to, with the address 3 being illegal. The length field specifies the number of data bytes (payload). A packet can have a minimum payload size of 1 byte and maximum of 63 bytes.

The parity is a byte of even, bitwise parity, calculated over the header and payload bytes of the packet.

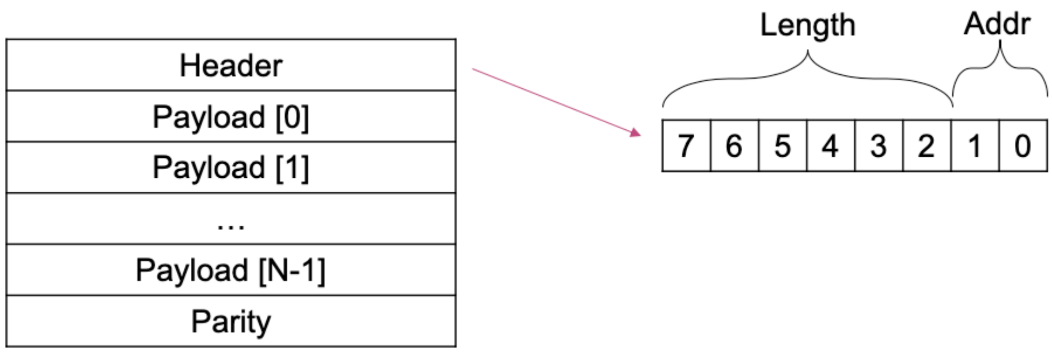
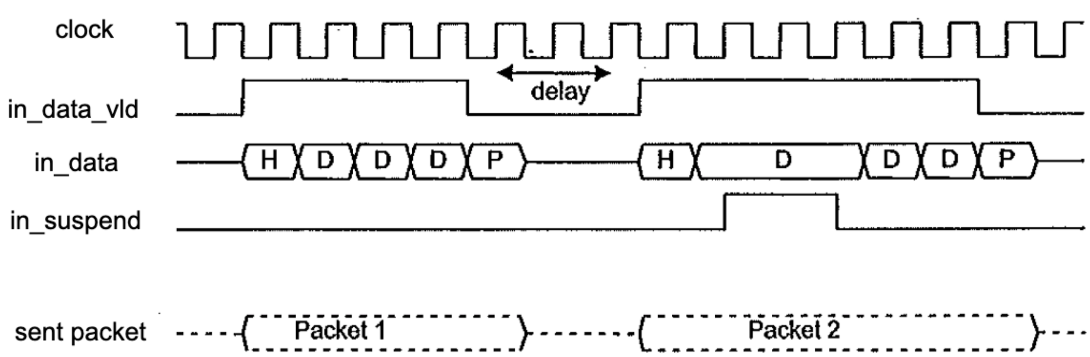


Fig2: YAPP Packet Structure

Input Port Protocol:

All input signals are active high and are to be driven on the falling edge of the clock. The `in_data_valid` signal must be asserted on the same clock when the first byte of the packet (the header byte), is driven onto the `in_data` bus. As the header byte contains the address, this tells the router to which output channel the packet needs to be routed. Each subsequent byte of data needs to be driven on the data bus with each falling clock.

After the last payload byte has been driven, on the next falling clock, the `in_data_valid` signal must be de-asserted, and the packet parity byte needs to be driven. The input data cannot change while `in_suspend` signal is active (indicating FIFO full).



H = Header, D = Data, P = Parity

Fig3: Input Port Protocol

Output Port Protocol (Channel Ports):

All output signals are active high and are to be sampled on the falling edge of the clock. Each output port is internally buffered by a FIFO of depth 16 and a width of 1 byte. The router asserts the `data_valid_x` signal when valid data appears on the `data_x` output bus. The `suspend_x` input signal must then be de-asserted on the falling clock edge in which data is read from the `data_x` bus. As long the `suspend_x` signal remains inactive, the `data_x` bus drives a new valid packet byte on each rising clock edge.

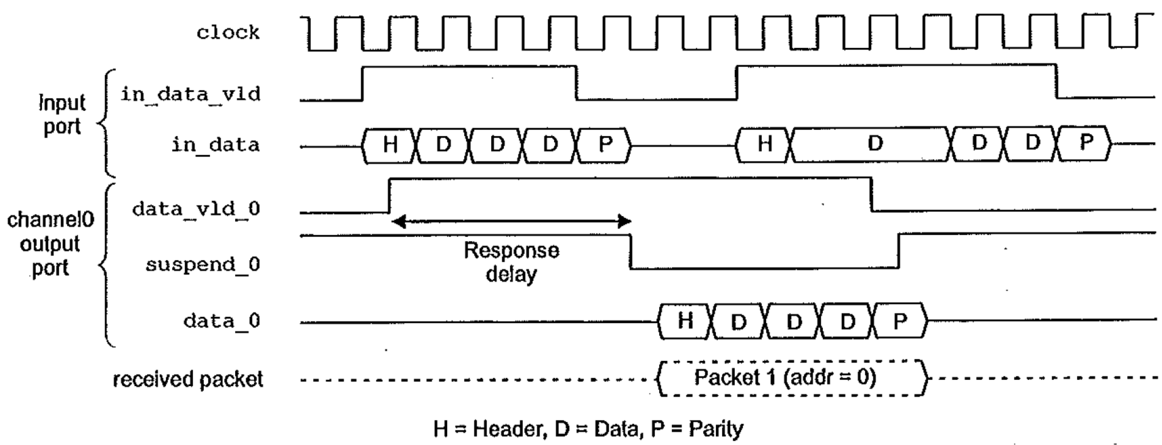


Fig4: Output Port Protocol

Host Interface Port Protocol (HBUS):

All input signals are active high and are to be driven on the falling edge of the clock. The host port provides synchronous read/write access to program the router.

A WRITE operation takes one cycle as follows:

- `hwr_rd` and `hen` must be 1. Data on `hdata` is then clocked on next rising clock edge in to the register based on `haddr` decode.
- `hen` is driven to 0 in the next cycle.

A READ operation takes two cycles as follows:

- `hwr_rd` must be 0 and `hen` must be 1. In the first clock cycle, `haddr` is sampled and `hdata` is driven by the design (DUT) in the second clock cycle.
- `hen` is then driven to 0 after the cycle 2 ends.

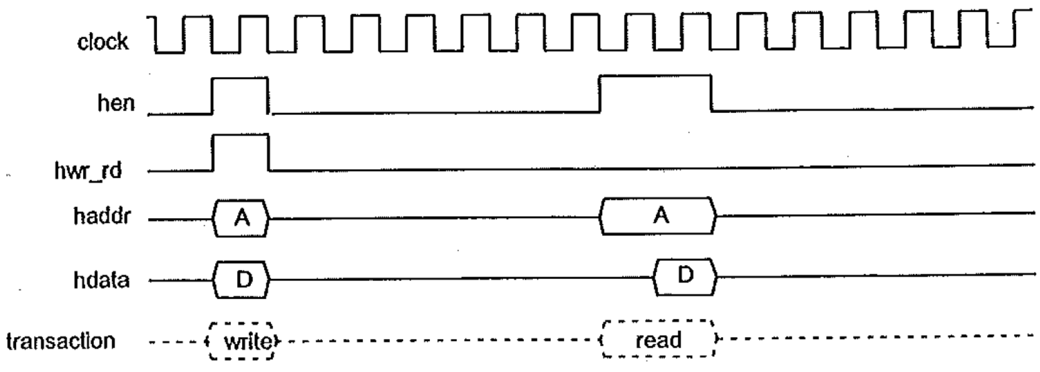


Fig5: HBUS Protocol

Verification Environment Plan:

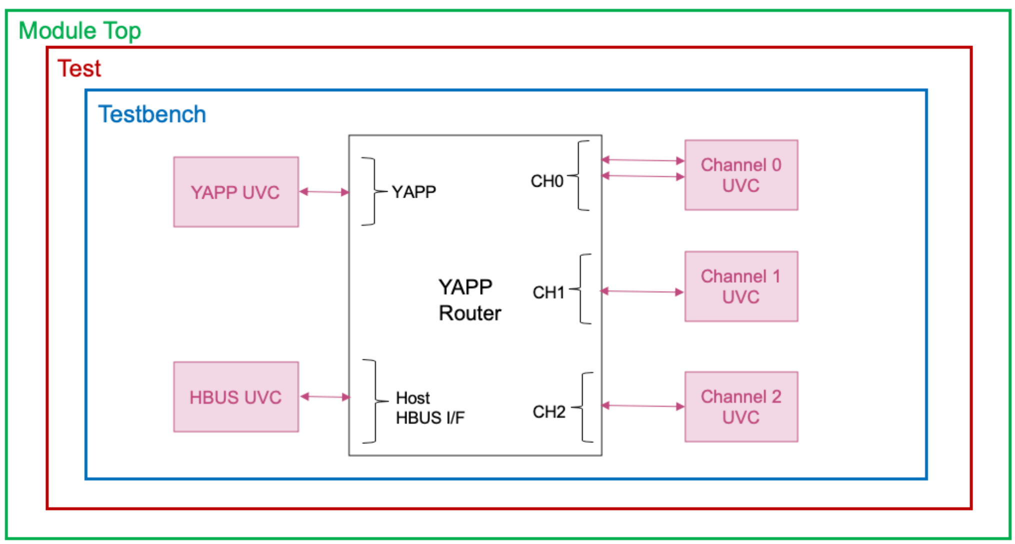


Fig6: Verification Environment for YAPP Router DUT

UVM Verification Component:

UVM Verification Component (UVC) emulates a design. It is an abstraction of the stimulus and monitoring needed to verify a design component, interface or protocols. It is defined by a set of classes and methods in the UVM library. It is mainly developed by uvm_components, uvm_env & uvm_test.

Env instantiates and configures ***agent. Agent*** contains three subcomponents: a ***driver, sequencer***, and ***monitor.*** If the agent is active, subtypes should contain all three subcomponents. If the agent is passive, subtypes should contain only the monitor.

Sequence is a series of ***transactions/packets*** which is of the type uvm_sequence_item. ***Sequencer*** is responsible for the coordination between sequence and driver. Sequencer sends the transaction to driver and gets the response from the driver.

Driver drives stimulus into the signals of DUT and ***Monitor*** observes/samples the signal interface of DUT.

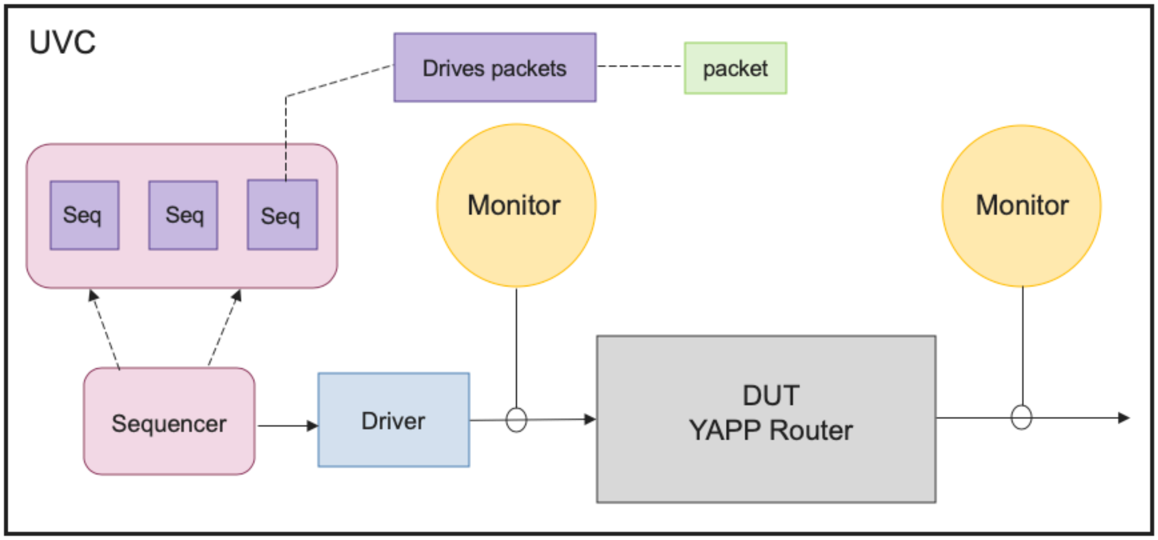


Fig7: Generic UVM Verification Component

Milestones:

Milestone 1	Understanding the design under test and building a plan for development of verification environment from it.
Milestone 2	Development of UVC for YAPP to drive & monitor the input channel of router
Milestone 3	Development of UVC for Channels, to monitor & drive the output channels of the router
Milestone 4	Development of UVC for HBUS Interface
Milestone 5	Connecting the UVCs to the DUT & development of testbench
Milestone 6	Development of top module
Milestone 7	Development of test which calls uvm sequences of different UVCs

Grading:

Assigned grades for milestone:

Grade	Milestones
B+	1, 2
A-	1, 2, 3, 4
A	1, 2, 3, 4, 5, 6
A+	1, 2, 3, 4, 5, 6, 7