PORTLAND STATE UNIVERSITY

Verification Plan - AHB LITE
ECE 571 : Fall 2022

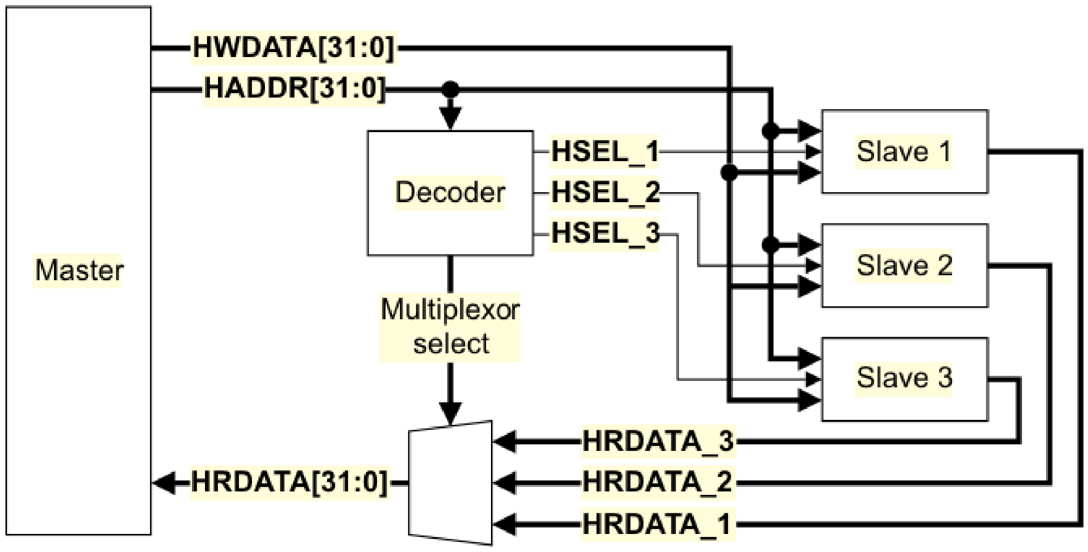
Jonathan Law <lawjon@pdx.edu> , Gagan Ganapathy Machiyanda Belliappa <gagan3@pdx.edu> , Balaji Rao Vavintaparthi <balaji2@pdx.edu>

Abstract : This report discusses the system level modeling of Advanced High performance Bus Lite (AHB-Lite), a subset of AHB that is part of the Advanced Microcontroller Bus Architecture (AMBA). Here we aim to create and implement a verification plan to verify the given design using SystemVerilog and test it using the QuestaSim simulator. We will be making use of oops, assertions, checkers, randomizations and other tools to perform this.

Introduction : AHB-Lite is a bus interface that supports a single bus master and offers high bandwidth operations. The most common slaves used for this protocol are internal memory devices, external memory interfaces, and high bandwidth peripherals. The main components of the AHB-Lite system are as follows:

- Master : An AHB-Lite master offers address and control information to initiate read and write operations.
- Slave : An AHB-Lite slave responds to transfers initiated by masters in the system.
- Decoder : This element decodes the address of each transfer and offers a select signal for the slave that is involved in the transfer. It also delivers a control signal to the multiplexor.
- Multiplexor : A slave-to-master multiplexor is required to multiplex the read data bus and response signals from the slaves to the master.

AHB Lite Block Diagram



Signal description:

- Global
  - HCLK - System clock.
  - HRESETn - System reset.
- Primary
  - HTRANS[1:0] - Transfer type.
  - HADDR[31:0] - Address.
  - HWRITE - Write high, read low.
  - HSIZE[2:0] - Transfer size.
  - HBURST[2:0] - Single, 4,8,16. Incrementing or wrapping.
  - HWDATA[31:0] - Primary write data.
  - HMASTLOCK - Lock transfer sequence.
  - HPROT[3:0] - Protection control documented in spec.
- Secondary
  - HRDATA[31:0] - Primary read data.
  - HREADYOUT - Read transfer complete.
  - HRESP - Low is okay, high is error
- Decoder
  - HSELx - Secondary selector bit for mux datapath.
- Multiplexor
  - HRDATA[31:0] - Primary read data.
  - HREADY - Transfer complete from primary and secondary.
  - HRESP - Transfer response.

Verification tasks

- Command packet
    - Transfer parameters
        - Size
        - Burst length
    - Write
    - Read
- Transfer types
    - IDLE
    - BUSY
    - NONSEQ
    - SEQ

Software/Tools :

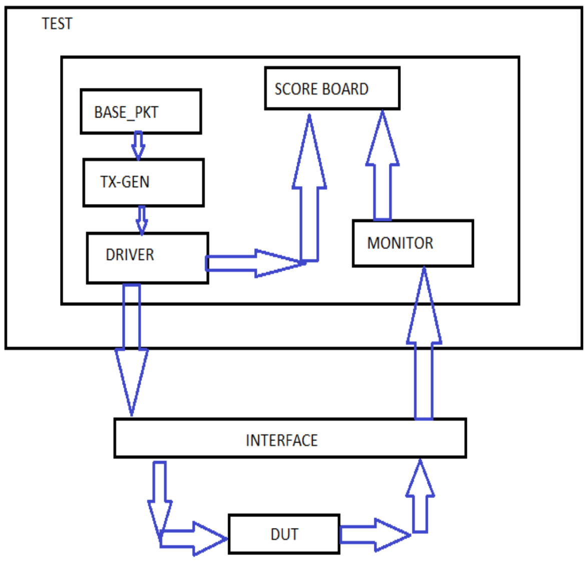
- QuestSim
- Github
- Google docs

Working Protocol : The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, width of the transfer, and indicate if the transfer forms part of a burst. The write data bus moves data from the master to a slave, and the read data bus moves data from a slave to the master.
Every transfer consists of two phases:
1) Address phase: one address and control cycle
2) Data phase: one or more cycles for the data.

A slave cannot request that the address phase is extended and therefore all slaves must be capable of sampling the address during this time. However, a slave can request that the master extends the data phase by using HREADY signal. This signal, when LOW, causes wait state to be inserted into the transfer and enables the slave to have extra time to provide or sample data. The slave uses a response signal to indicate the success or failure of a transfer.

Initial Verification Background : SystemVerilog supports many features for verification including oops, assertions, coverage, etc.. Object oriented programming feature of SystemVerilog makes it a powerful language to verify corner cases of logic designs. One more important feature is that SystemVerilog supports randomization using which we can randomize the test vectors instead of generating all the values. This is highly efficient and it also saves time.

Verification Architecture :



Functionality of each block in the verification architecture

- TX-GEN: The variables generated in BASE_PKT are given to TX-GEN. Here the variables are converted to test vectors and given to the driver.
- DRIVER: The driver drives the test vectors to the respective blocks, one to the interface and the other to the scoreboard.
- MONITOR: Monitor captures the DUT output and sends it to the scoreboard for verification.
- INTERFACE: Interface acts as communication medium between the DUT and the TEST.
- SCORE BOARD: Scoreboard compares the generated test vectors and the DUT to check if the designed circuit is correct.
- DUT: Device Under Test.

Setbacks : Since our group had minimal experience with SystemVerilog prior to this project, we were unable to implement oops SystemVerilog constructs successfully in designing the individual elements involved in the verification architecture shown above, however we did manage to implement a few blocks namely the interface block (which included the read, burst read, write and burst tasks), the driver block and in addition to this we also included a few assertions used for comparison in the scoreboard module.

The biggest challenge we faced was while integrating these modules together. We did not get the results we desired, hence we decided to go with writing a linear testbench which included tasks to perform read/write operations and assertion statements.

Timing diagrams referenced from the AMBA 3 AHB-Lite protocol specification v1.0.

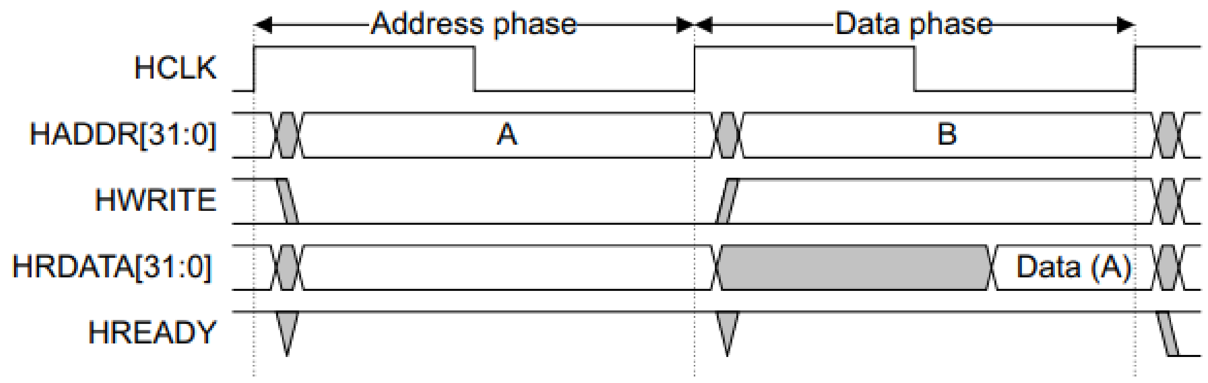


Figure 3-1 Read transfer

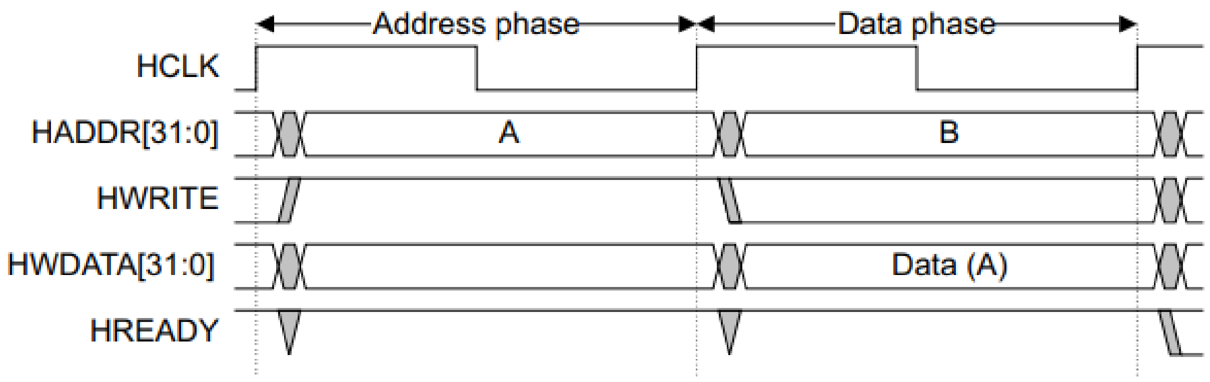


Figure 3-2 Write transfer

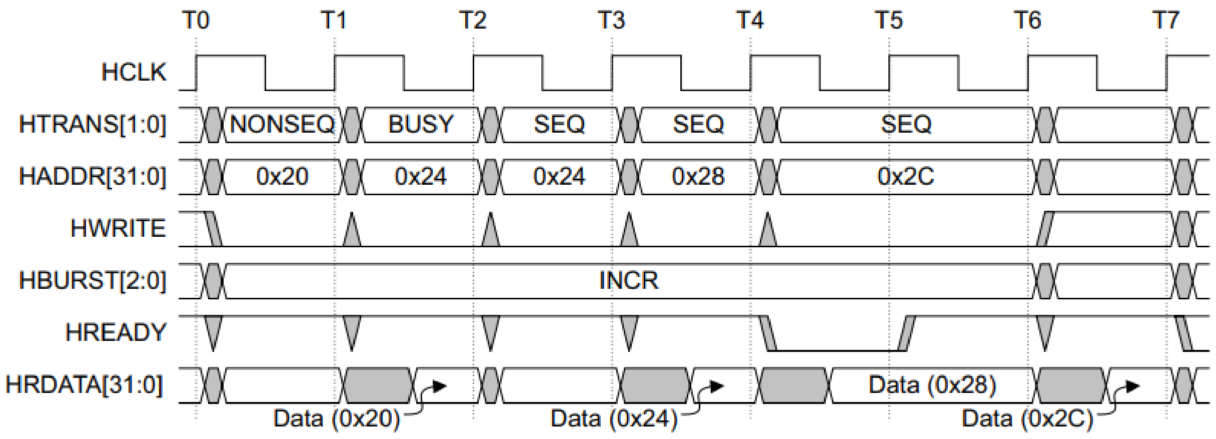


Figure 3-6 Transfer type examples

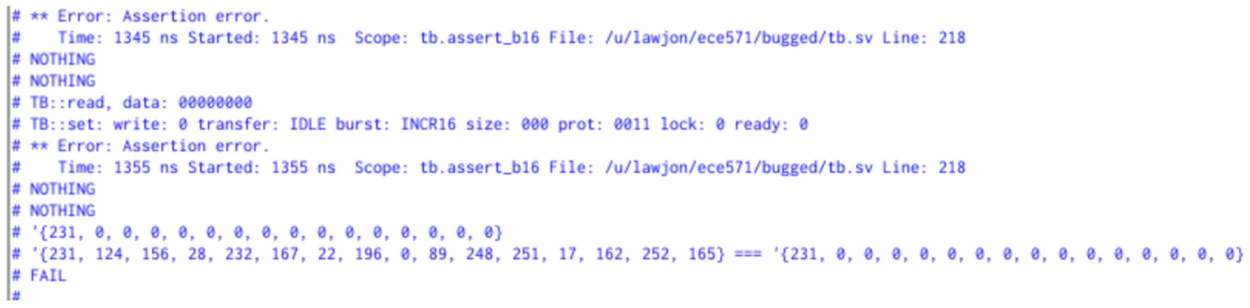
Starter code functionality is different from the AHB Lite spec.
We tried to follow the spec waveforms but we were seeing errors in the performance.
The sample testbench never read back any of the written memory values for comparison.
For multibyte operations, HADDR is only taken at the NONSEQ clock. Then it ignores the rest of the HADDR for the other SEQ clocks.

```
        else if ((HTRANS == NONSEQ)&&(HREADY!=1'b0))
                begin
                        Address <= HADDR[7:0];
```

Injected bugs found

We found 1 of the injected bugs. The DUT only writes the first byte of the burst and does not write anymore. This happens with the HBURST = WRAP4, INCR4, WRAP8, INCR8, WRAP16, INCR16.

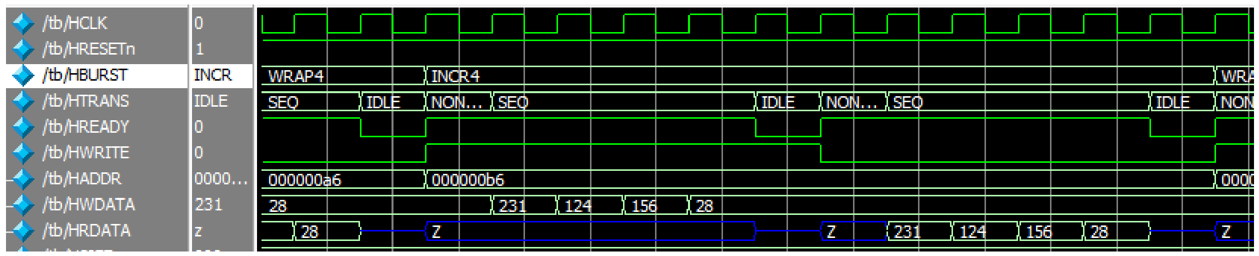
Screenshot of simulator transcript with the bug that we found:



```
# ** Error: Assertion error.
#    Time: 1345 ns Started: 1345 ns  Scope: tb.assert_b16 File: /u/lawjon/ece571/bugged/tb.sv Line: 218
# NOTHING
# NOTHING
# TB::read, data: 00000000
# TB::set: write: 0 transfer: IDLE burst: INCR16 size: 000 prot: 0011 lock: 0 ready: 0
# ** Error: Assertion error.
#    Time: 1355 ns Started: 1355 ns  Scope: tb.assert_b16 File: /u/lawjon/ece571/bugged/tb.sv Line: 218
# NOTHING
# NOTHING
# '{231, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
# '{231, 124, 156, 28, 232, 167, 22, 196, 0, 89, 248, 251, 17, 162, 252, 165} === '{231, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
# FAIL
#
```

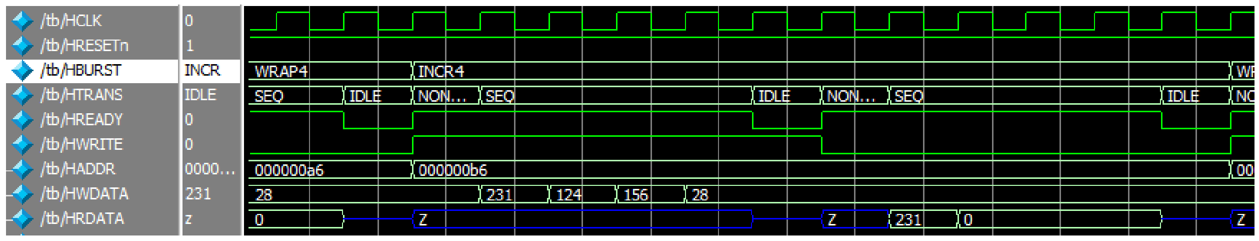
Screenshot of simulated timing diagram from the good starter code:

HBURST = INCR4
Write then read back
Package flow is Addr > Data x4 > IDLE



Screenshot of simulated timing diagram with the injected bug:

HRDATA is missing after the first byte.



Assertions Written :

1. **Assertion for basic write :** ***hwrite*** is detected high at second clock and at the third clock ***hready*** goes high.

```
property basic_write;
     @(posedge   Bus.HCLK)   disable   iff   ((Bus.HTRANS==IDLE   ||
Bus.HTRANS==BUSY) && Bus.HBURST>'0) $rose(Bus.HWRITE) |=> Bus.HREADY;
endproperty
```

2. **Assertion for basic read :** ***hwrite*** is detected low at second clock and at the third clock ***hready*** goes high.

```
property basic_read;
     @(posedge   Bus.HCLK)   disable   iff   ((Bus.HTRANS==IDLE   ||
Bus.HTRANS==BUSY)   &&   Bus.HBURST>'0   ||   (Bus.HADDR   >   ((2**10)*
`NoOfSlaves))) $fell(Bus.HWRITE) |=> Bus.HREADY;
endproperty
```

3. **Assertion for basic burst write :** If ***hwrite*** is detected as high and ***htrans*** is in non sequential state at the second clock, at the third clock hready goes high. It is disabled if it's in a busy state.

```
property basic_burst_write;
@(posedge Bus.HCLK) disable iff (Bus.HTRANS == BUSY)
((Bus.HWRITE==1)&&(Bus.HTRANS == NON_SEQ) )|=>     (Bus.HREADY=='1) ;
endproperty
```

4. **Assertion for basic burst read :** If ***hwrite*** is detected as low and ***htrans*** is in non sequential state at the second clock, at the third clock hready goes high. It is disabled if it's in a busy state.

```
property basic_burst_read;
@(posedge Bus.HCLK) disable iff (Bus.HTRANS == BUSY || (Bus.HADDR >
((2**10)* `NoOfSlaves)))
((Bus.HWRITE=='0)&&(Bus.HTRANS == NON_SEQ) )|=>     (Bus.HREADY=='1) ;
endproperty
```

5. **Assertion for transition :** This is an assertion for non sequential to sequential transition.

```
property seq_Check;
@(posedge Bus.HCLK) (( (Bus.HWRITE=='1) || (Bus.HWRITE=='0) ) &&
(Bus.HTRANS==NON_SEQ) ) |=> (Bus.HTRANS == SEQ);
endproperty
```

6. **Assertion for hready :** If ***hready*** is low then ***HADDR***, ***HWRITE*** and ***HDATA*** should remain in the same state until the ***HREADY*** signal goes high.

```
property HREADY_Check;
@(posedge Bus.HCLK) (Bus.HREADY == 1'b0) |=> $stable (Bus.HADDR &&
Bus.HWRITE && Bus.HWDATA) ;
endproperty
```

7. **Assertion for burst count :** This assertion checks for 4, 8, 16 incrementing bursts when the state transition is going. That is when a non sequential state is followed by 3, 7, 15 sequential states.

```
property burst_count_check4;
@(posedge Bus.HCLK) disable iff(Bus.HTRANS == BUSY || Bus.HBURST !=3'b011)
(Bus.HTRANS == 2'b10) |=> (Bus.HTRANS == SEQ)|=> (Bus.HTRANS == SEQ)[*2];
endproperty
```

```
property burst_count_check8;
@(posedge Bus.HCLK)disable iff(Bus.HTRANS == BUSY || Bus.HBURST !=3'b101)
(Bus.HTRANS  ==  NON_SEQ)  |=>  (Bus.HTRANS  ==  SEQ)|=>  (Bus.HTRANS  ==
SEQ)[*7];
endproperty

property burst_count_check16;
@(posedge Bus.HCLK)disable iff(Bus.HTRANS == BUSY  || Bus.HBURST !=3'b111)
(Bus.HTRANS  ==  NON_SEQ)  |=>  (Bus.HTRANS  ==  SEQ)  |=>  (Bus.HTRANS  ==
SEQ)[*14];
endproperty
```

8. **Assertion for address change :** This assertion checks for 4, 8, 16 incrementing bursts whether the address is changing. That is when a non sequential state is followed by 3, 7, 15 sequential states.

```
property address_change4;
@(posedge Bus.HCLK) disable iff (Bus.HBURST!=3'b011)
(Bus.HTRANS == NON_SEQ) |=> not ($stable(Bus.HADDR)[*3]);
endproperty

property address_change8;
@(posedge Bus.HCLK) disable iff (Bus.HBURST!=3'b101)
(Bus.HTRANS == NON_SEQ) |=> not ($stable(Bus.HADDR)[*7]);
endproperty


property address_change16;
@(posedge Bus.HCLK) disable iff (Bus.HBURST!=3'b111)
(Bus.HTRANS == NON_SEQ) |=> not ($stable(Bus.HADDR)[*15]);
endproperty
```

Conclusion: This project helped us develop new skills and also taught us about various approaches to arrive at a verification plan which can be applied towards future projects. We would have liked to build a proper working verification environment and write more assertions to improve coverage if we had more time. The first time around, it required some time to fully comprehend each phase of the process, but we are optimistic that future testing would go quicker and more smoothly. We are happy that we did find one of the injected bugs. One other bug may have been hidden behind the bug that we did find. And the last bug may have had a work around that we bypassed in our test sequence.

Team members responsibilities :

1. Jonathan Law - Verification Environment Development, Framing Assertions, Simulation in QuestaSim, Documentation.

2. Gagan Ganapathy Machiyanda Belliappa -Verification Environment Development, Top Level Integration, Framing Assertions, Documentation.

3. Balaji Rao Vavintaparthi - RTL Development & Packaging, Writing Test Cases, Debugging and Documentation.

References

[1] Prof Mark Faust Prof Roy Kravitz. Lecture notes for ECE 571: Introduction to SystemVerilog.

[2] ARM. Amba 3 ahb-lite protocol specification v1.0.

[3] https://issuu.com/ijraset/docs/33120

[4]https://www.researchgate.net/publication/304943129_Design_and_verification_of_AMBA_AHB-lite_protocol_using_verilog_HDL