



Portland  
State  
UNIVERSITY

# **ECE-560, Fall 2024**

## **Assertion Based Verification**

Asst. Prof of Practice: Venkatesh Patil

Maseeh College of Engineering and Computer Science

# **Final Project Report**

## **AHB2APB BRIDGE**

**Team Number 3**

**Yashaswi Katne | [katne@pdx.edu](mailto:katne@pdx.edu)**

**Maheswar Reddy Kamalapuram | [maheswar@pdx.edu](mailto:maheswar@pdx.edu)**

**Naveen Kumar Reddy Thummala | [naveenkt@pdx.edu](mailto:naveenkt@pdx.edu)**

**Prasanna Kumar Panchada | [prapanch@pdx.edu](mailto:prapanch@pdx.edu)**

## Table of Contents

Introduction.....	3
Project Proposal/Overview.....	3
Bus Bridge Description .....	3
Project Collaboration and file structure .....	4
Specifications .....	6
AHB Master .....	7
AHB Slave Interface .....	9
Timing Diagrams .....	10
Verification Plan Execution .....	13
Assertions.....	14
Assumptions.....	15
Cover.....	15
State Machine for the AHB to APB interface. ....	16
Snapshots.....	17
Challenges.....	19
Summary.....	19
References.....	19

## Introduction:

The aim of this project and the associated verification plan is to outline the approaches, methodologies, and results derived from our efforts in assessing the effectiveness and performance attributes of the AHB-APB bridge design. This bridge serves as a vital intermediary, which enables seamless communication between the high-performance Advanced High-performance Bus (AHB) and the power-efficient Advanced Peripheral Bus (APB).

As semiconductor technology continues to evolve, there is a growing need for accuracy and reliable, the need for verification processes has become increasingly vital. In a design involving high-speed data transfer, memory mapping, and peripheral control, the verification of the bridge requires an integrated approach. This paper presents our step-by-step strategy for functional testing of the AHB-APB bridge for its conformance to the requirements specified by the ARM AMBA specification sheet. Our experience in this verification process has been challenging and enlightening. We tried to define assertions, assumptions, and coverage properties to check the design.

## Project Proposal:

The goal of our project, which involved the principles of assertion-based verification, was to create and apply a set of comprehensive validation for the functionality of the AHB-to-APB (AHB2APB) bridge design. This was inspired by our commitment to adhere to the set ARM AMBA standards that define the nature of this critical interface. During this project, our group engaged in crafting a set of properties derived directly from the bridge's design specifications. We developed an array of assertions, assumptions, and cover properties in parallel to perform extensive testing of various functionalities provided by the bridge. This effort was not easy in the beginning.

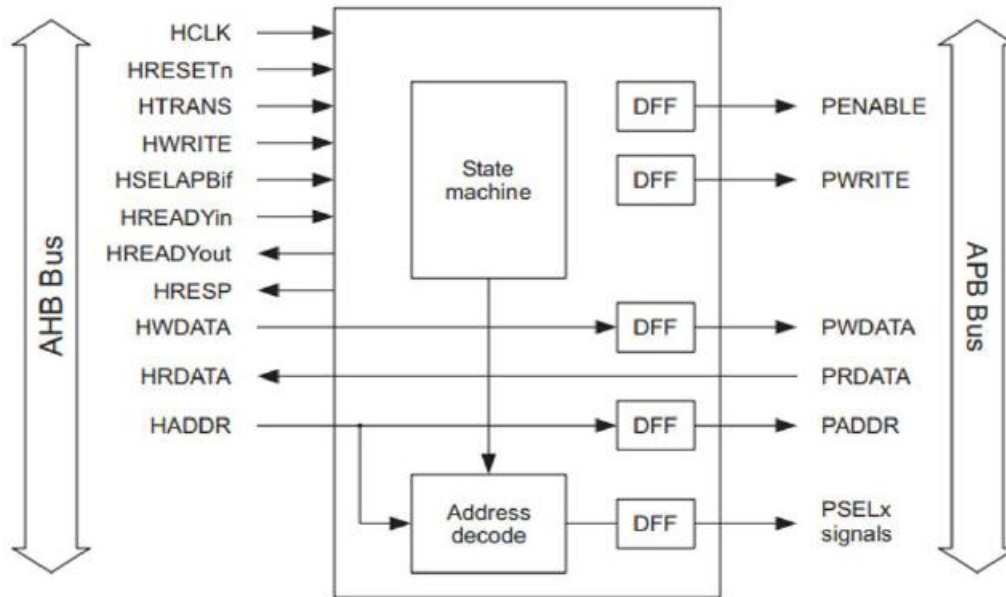
Specifically, we faced several major obstacles at the start of the above effort. The foremost of these issues involved obtaining a proper RTL (Register Transfer Level) code that met all our specified criteria. In other words, we were looking for RTL facilitated the burst read and write transactions as specified for the ARM AMBA standards. In the light of these challenges, we decided to proceed with the RTL code available for the AHB2APB bridge design, as an integral part of this process.

This has been presented as an annex to this report for quick reference. The RTL code selected can also be accessed from: <https://github.com/prajwalgekkouga/AHB-to-APB-Bridge>.

## BUS BRIDGE DESCRIPTION:

The AHB2APB bridge developed by ARM acts as a crucial interface between the fast-paced AHB and the power-efficient APB. In simpler words, this can be called a bridge that offers compatibility to a wide range of read-write transactions in both buses. Amongst the key ingredients for such a bridge are a slave bus interface, an APB transfer state machine that operates independently of the memory architecture of a device, and methods for providing APB output signals are provided.

The AHB2APB bridge is basically designed to preserve the addresses, control signals, and data originating from the AHB and then routes this to the APB peripherals while, at the same time, sends data along with a response signal back to the AHB. The bridge manages the APB data bus using two different channels: the read data path PRDATA and the write data path. It is designed to handle sequential and nonsequential data transfers of various sizes: PWDATA and HSIZE. This enables a runtime communication interface between the high-speed and low-speed buses.



## PROJECT COLLABORATION:

The team worked collaboratively, and the following file structure was set up to effectively manage the project. It was agreed that a GitHub repository would be used in order to manage our RTL, together with all associated properties, assertions, assumptions, and coverage points. This structure allowed each member of the team to work on their own properties independently yet still have access to.

Drawing on the work of colleagues, we created separate branches for each member of the team and populated these with template files.

We organized our repository into top-level directories for RTL, run, and SVA. Inside the run directory, we had specific templates that correspond to each RTL file of interest for our AEP, FPV, and FXP TCL files. Inside the SVA directory, we used templates to contain our SystemVerilog assertions, where each assertion is a single RTL file.

## Snapshot of Filelist:

```
bind_bridge_top_sva.sva  AHB_Slave_Interface.sv  APB_Controller
1  ../RTL/APB_Controller.sv
2  ../RTL/AHB_Slave_Interface.sv
3  ../RTL/bridge_top.sv
4  ../RTL/bridge_top_sva.sva
5  ../RTL/bind_bridge_top_sva.sva
6  ../RTL/AHB_Slave_Interface_sva.sva
7  ../RTL/bind_AHB_Slave_Interface_sva.sva
8  ../RTL/APB_Controller_sva.sva
9  ../RTL/bind_APB_Controller_sva.sva
```

## Snapshot of Successful compilation:

The screenshot displays the QuestaSim IDE interface. The top-left pane shows a project tree with files: `APB_Controller.sv`, `run.do`, `bridge_top.sv`, `bridge_top_tb.sv`, and `AHB_Slave_Interface.sv`. The top-right pane shows a waveform viewer with a time scale from 0 ns to 100 ns. The bottom pane shows the Transcript window with the following text:

```
# // Questa Sim-64
# // Version 2021.3_1 linux_x86_64 Aug 15 2021
# //
# // Copyright 1991-2021 Mentor Graphics Corporation
# // All Rights Reserved.
# //
# // QuestaSim and its associated documentation contain trade
# // secrets and commercial or financial information that are the property of
# // Mentor Graphics Corporation and are privileged, confidential,
# // and exempt from disclosure under the Freedom of Information Act,
# // 5 U.S.C. Section 552. Furthermore, this information
# // is prohibited from disclosure under the Trade Secrets Act,
# // 18 U.S.C. Section 1905.
# //
# Loading project abvhw3
# reading /pkgs/mentor/questa/2021.3_1/questasim/linux_x86_64/./modelsim.ini
# Loading project abv_fp
# Compile of AHB_Slave_Interface.sv was successful.
# Compile of APB_Controller.sv was successful.
# Compile of bridge_top.sv was successful.
# Compile of bridge_top_tb.sv was successful.
# 4 compiles, 0 failed with no errors.
```

The transcript text is highlighted with a red box. The prompt `QuestaSim>` is visible at the bottom of the transcript window.

TCL:  
one of the tcl file

The screenshot shows a TCL script file in the QuestaSim IDE. The script contains the following commands:

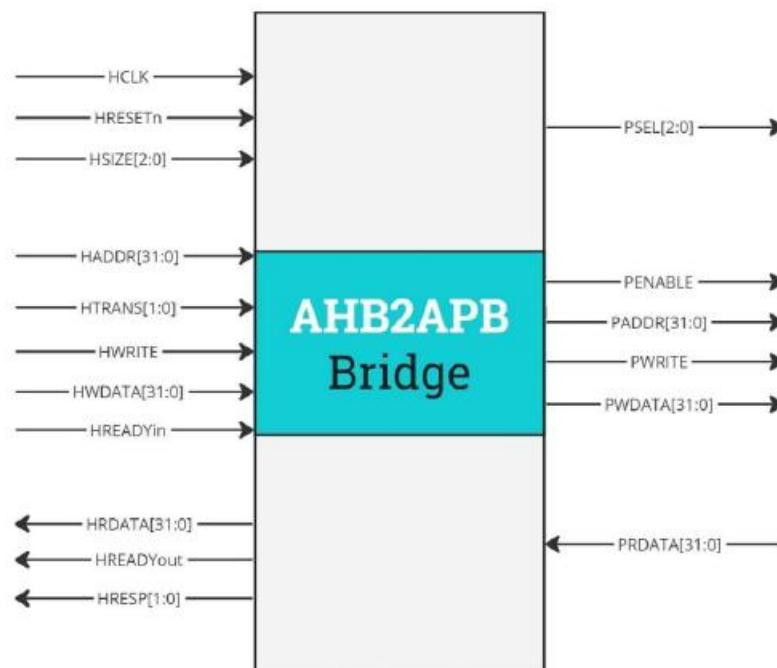
```
1
2 set_fml_apemode FPV
3 set design Bridge_Top
4
5
6 read_file -top $design -format sverilog -sva -vcs {-f ../RTL/filelist}
7
8 create_clock Hclk -period 100
9 create_reset Hresetn -sense low
10
11 sim_run -stable
12 sim_save_reset
13
```

## Specifications:

### Design Overview:

- The custom-designed AHB2APB bridge provides the interface between the Advanced High-performance Bus and the Advanced Peripheral Bus, which is designed to meet our needs.
- It allows for AHB masters and APB slaves to be supported in this configuration for smooth transactions between the two interfaces while ensuring that our design specifications are met.

The below diagram illustrates the input and output signals available in the AHB to APB Bridge, most (but not all of them) are exposed as input and output pins in the RTL design we are using. Since this is a bridge protocol, understanding the specifications of those signals is crucial for us to write good properties and to check the right functionality of the bridge.



### Design Input Signals (AHB side):

- HCLK: Clock signal for the AHB interface.
- HRESETn: Active-low reset signal for the AHB interface.
- HSIZE[2:0]: Size of the transfer on the AHB interface.
- HADDR[31:0]: Address for the AHB transfer.
- HTRANS[1:0]: Transfer type on the AHB interface.
- HWRITE: Write enable signal for the AHB interface.
- HWDATA[31:0]: Write data on the AHB interface.
- HREADYin: Ready signal indicating the availability of the AHB interface.

### Design Output Signals (AHB side):

- HRDATA[31:0]: Read data from the AHB interface.
- HREADYout: Ready signal indicating the readiness of the AHB interface.
- HRESP[1:0]: Response indicating the status of the AHB transfer.

Design Input Signals (APB side):

- PRDATA[31:0]: Read data on the APB interface.

Design Output Signals (APB side):

- PSEL[2:0]: Slave select signal on the APB interface.
- PENABLE: Enable signal for the APB interface.
- PADDR[31:0]: Address for the APB transfer.
- PWRITE: Write enable signal for the APB interface.
- PWDATA[31:0]: Write data on the APB interface.

These input and output signals form the communication interface between the AHB and APB sides of the bridge design. They facilitate the transfer of data and control signals between the two interfaces, ensuring the proper functioning of the AHB2APB bridge.

## AHB Master:

We have extracted the following specs from the AMBA reference manual.

<b>HTRANS[1:0]</b> Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
-------------------------------------	--------	---

<b>HTRANS[1:0]</b>	<b>Type</b>	<b>Description</b>
00	IDLE	Indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer.  Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be ignored by the slave.
01	BUSY	The BUSY transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfers. This transfer type indicates that the bus master is continuing with a burst of transfers, but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst.  The transfer should be ignored by the slave. Slaves must always provide a zero wait state OKAY response, in the same way that they respond to IDLE transfers.
10	NONSEQ	Indicates the first transfer of a burst or a single transfer. The address and control signals are unrelated to the previous transfer.  Single transfers on the bus are treated as bursts of one and therefore the transfer type is NONSEQUENTIAL.
11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the size (in bytes). In the case of a wrapping burst the address of the transfer wraps at the address boundary equal to the size (in bytes) multiplied by the number of beats in the transfer (either 4, 8 or 16).

<b>HSIZE[2:0]</b> Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
------------------------------------	--------	--

### 3.7.1 Transfer direction

When **HWRITE** is HIGH, this signal indicates a write transfer and the master will broadcast data on the write data bus, **HWDATA[31:0]**. When LOW a read transfer will be performed and the slave must generate the data on the read data bus **HRDATA[31:0]**.

### 3.7.2 Transfer size

**HSIZE[2:0]** indicates the size of the transfer, as shown in Table 3-3.

**Table 3-3 Size encoding**

<b>HSIZE[2]</b>	<b>HSIZE[1]</b>	<b>HSIZE[0]</b>	<b>Size</b>	<b>Description</b>
0	0	0	8 bits	Byte
0	0	1	16 bits	Halfword
0	1	0	32 bits	Word
0	1	1	64 bits	-
1	0	0	128 bits	4-word line
1	0	1	256 bits	8-word line
1	1	0	512 bits	-
1	1	1	1024 bits	-

The size is used in conjunction with the **HBURST[2:0]** signals to determine the address boundary for wrapping bursts.

<b>HBURST[2:0]</b> Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
----------------------------------	--------	--



**Table 3-2 Burst signal encoding**

<b>HBURST[2:0]</b>	<b>Type</b>	<b>Description</b>
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

## AHB SLAVE INTERFACING:

<b>HRESP[1:0]</b>	Slave	The transfer response provides additional information on the status of a transfer.
Transfer response		Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

During a transfer the slave shows the status using the response signals, **HRESP[1:0]**:

**OKAY** The OKAY response is used to indicate that the transfer is progressing normally and when **HREADY** goes HIGH this shows the transfer has completed successfully.

**ERROR** The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.

**RETRY and SPLIT** Both the RETRY and SPLIT transfer responses indicate that the transfer cannot complete immediately, but the bus master should continue to attempt the transfer.

In normal operation a master is allowed to complete all the transfers in a particular burst before the arbiter grants another master access to the bus. However, in order to avoid excessive arbitration latencies it is possible for the arbiter to break up a burst and in such cases the master must re-arbitrate for the bus in order to complete the remaining transfers in the burst.

Table 3-5 Response encoding

HRESP[1]	HRESP[0]	Response	Description
0	0	OKAY	When <b>HREADY</b> is HIGH this shows the transfer has completed successfully. The OKAY response is also used for any additional cycles that are inserted, with <b>HREADY</b> LOW, prior to giving one of the three other responses.
0	1	ERROR	This response shows an error has occurred. The error condition should be signalled to the bus master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition.
1	0	RETRY	The RETRY response shows the transfer has not yet completed, so the bus master should retry the transfer. The master should continue to retry the transfer until it completes. A two-cycle RETRY response is required.
1	1	SPLIT	The transfer has not yet completed successfully. The bus master must retry the transfer when it is next granted access to the bus. The slave will request access to the bus on behalf of the master when the transfer can complete. A two-cycle SPLIT response is required.

## TIMING DIAGRAMS:

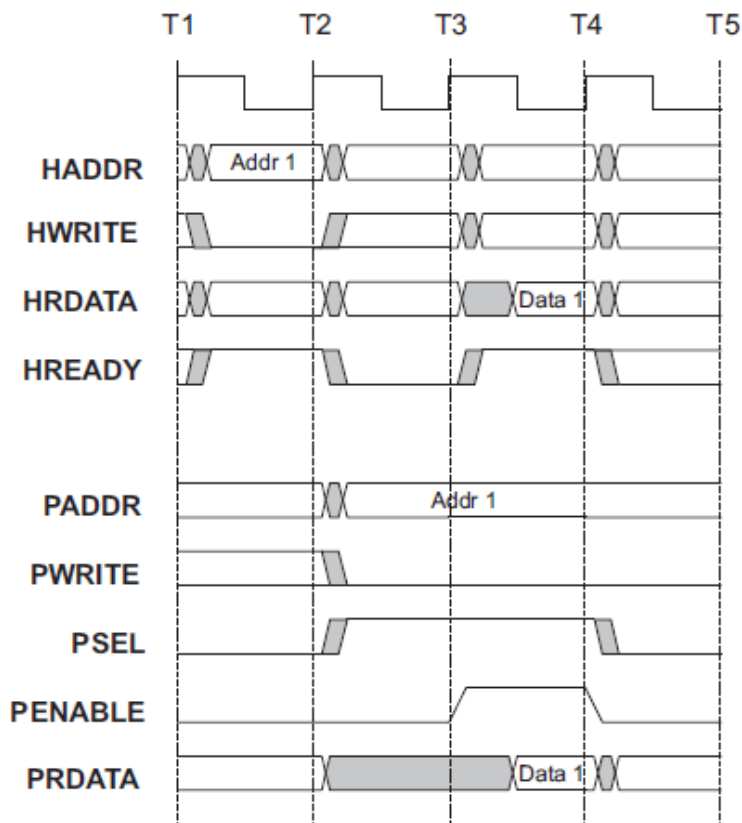


Figure 5-9 Read transfer to AHB

The APB samples the HADDR signal at time T2, as indicated by the PADDR signal being "addr 1" at that time. At T2, the PWRITE signal is driven low while PSEL is asserted high. It is important that PSEL should be high during the whole of the PENABLE signal which must also be asserted high during the transfer in order to make Data1 valid. Also, the HREADY signal should be high when DATA1 is on the HRDATA line. Each read needs a wait state for the transfer to happen.

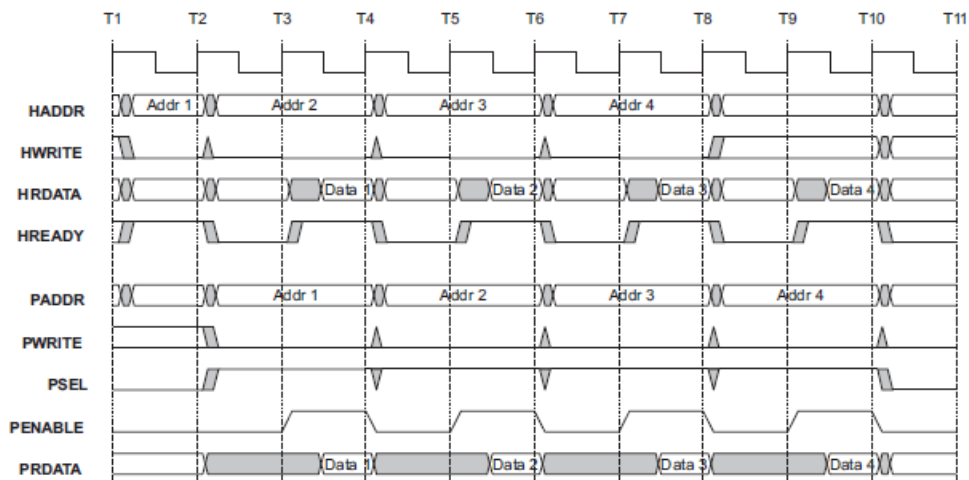


Figure 5-10 Burst of read transfers

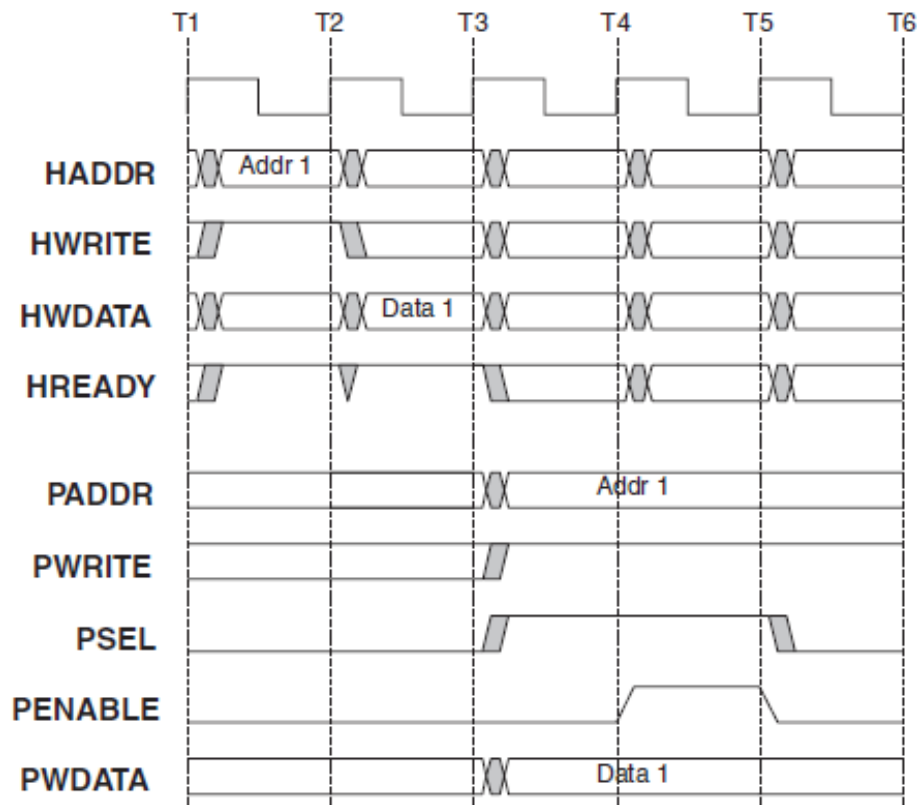


Figure 5-11 Write transfer from AHB

Write transfers can occur with zero transfers. The bridge must sample the address and data of the transfer and must hold these values for the duration of the write transfer.

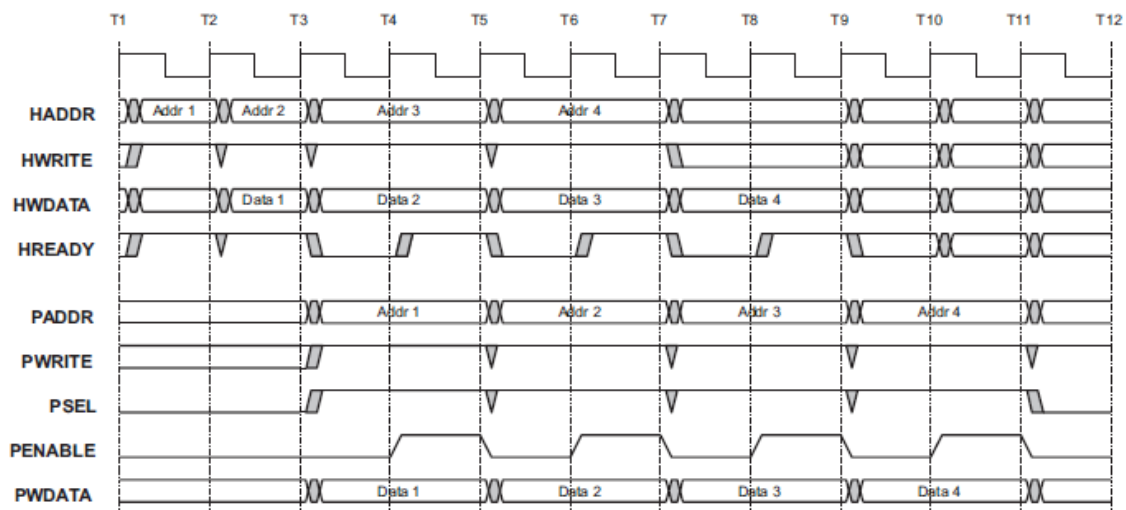


Figure 5-12 Burst of write transfers

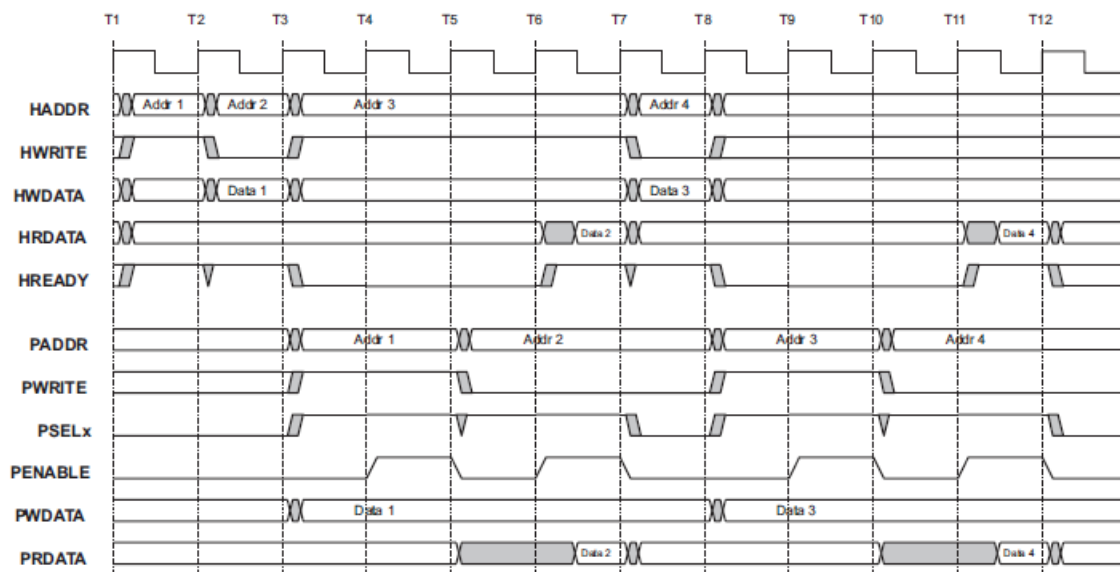


Figure 5-13 Back to back transfers

When a read follows a write, there must be 3 wait states to complete the read. The three wait states can be seen on HREADY and PWRITE.

## VERIFICATION PLAN:

The assertions in these files are used for formal verification and dynamic simulation to ensure the correctness of the design under various conditions. Below is an outline of the verification methodology:

### 1. Functional Properties Verification

Each property in the file corresponds to a functional requirement of the design.

For example:

AHB\_slave\_interface\_sva:

Valid signal generation: Ensures the valid signal is asserted under the correct conditions (e.g., Hreadyin is high, Haddr is within range, and Htrans is valid). Address-based decoding (tempse1x): Checks if tempse1x is correctly set based on the Haddr range.

Invalid conditions handling: Ensures that the module behaves predictably when Haddr is out of range or invalid.

APB\_Controller\_sva:

State transitions: Verifies that the controller transitions between states (PRESENT\_STATE to NEXT\_STATE) correctly based on the design specification.

Read/Write sequencing: Ensures the APB controller follows the correct sequence for read and write transactions (e.g., enabling Penable after valid Pse1x).

Bridge\_Top\_sva:

Protocol compliance: Assumes and checks the behavior of AMBA AHB and APB protocols, such as: Hwrite and Hreadyin behavior. Valid address ranges for AHB transactions, Synchronization between AHB (Hwdata, Haddr) and APB (Pwdata, Paddr).

Data consistency: Ensures that Hrdata matches Prdata during enable cycles.

### 2. Temporal Behavior Verification

Assertions include temporal logic to validate timing relationships:

@posedge Hclk: Properties are verified at each clock cycle.

Delay operators (##, |->): Used to verify that signals and state transitions occur in the correct sequence and timing:

Example: write\_s ##1 read\_s ensures a read occurs one cycle after a write.

Example: PRESENT\_STATE == ST\_IDLE |-> NEXT\_STATE == ST\_READ ensures a specific state transition.

### 3. Protocol Compliance Verification

The files ensure compliance with the AHB and APB protocol requirements:

One-hot encoding: Checks that Pse1x is always one-hot or zero-hot.

Penable behavior: Ensures Penable is not high for consecutive cycles.

Address decoding: Verifies address mapping for peripherals in the APB subsystem.

Write and read waits: Ensures that the bridge respects wait states during read-after-write scenarios.

### 4. Corner Case Verification

The assertions also cover edge cases:

Reset conditions: Ensures all signals are reset to zero after a reset event.

Back-to-back transactions: Verifies behavior during bursts of reads or writes.

Read-after-write timing: Ensures correct timing and data consistency when a read transaction immediately follows a write.

## 5. Coverage Collection

Coverage properties (e.g., cover\_burst\_of\_reads, cover\_burst\_of\_writes) are used to:

Ensure the verification environment exercises key design scenarios, such as bursts of reads, back-to-back writes, and transitions between states. Identify untested portions of the design.

## Assertions/Assumptions/Cover:

```
//For Address range 8400_0000 to 8800_0000
sequence psel_s1;
$onehot(Pselx) ##0 (Pselx[1] ##1 Pselx[1]);
endsequence

//For Address range 8800_0000 to 8C00_0000
sequence psel_s2;
$onehot(Pselx) ##0 (Pselx[2] ##1 Pselx[2]);
endsequence

// HRDATA should be same as PRDATA when PENABLE(Enable cycle)
property same_HR_PR_data;
@(posedge Hclk) disable iff(!Hresetn)
Penable |-> Hrddata == Prdata;
endproperty
assert_same_HR_PR_data: assert property (same_HR_PR_data);

//PWRITE should be same as HWRITE for Read transfer
property same_HPwrite_read;
@(posedge Hclk) disable iff(!Hresetn)
!Hwrite && Hreadyin && !($past(Hwrite) && $past(Hreadyin)) |=> (Pwrite == 0);
endproperty
assert_same_HPwrite_read: assert property (same_HPwrite_read);

////PWRITE should be same as HWRITE for Write transfer
property same_HPwrite_write;
@(posedge Hclk) disable iff(!Hresetn)
write_s |-> ##1 (Pwrite == 1);
endproperty
assert_same_HPwrite_write: assert property (same_HPwrite_write);

//HREADYOUT and PENABLE should be high at end of transaction for Read transfer
property Hreadyout_penable_read;
@(posedge Hclk) disable iff(!Hresetn)
read_s |-> ##2 Penable && Hreadyout;
endproperty
assert_Hreadyout_penable_read: assert property (Hreadyout_penable_read);

//HREADYOUT and PENABLE should be high at end of transaction for write transfer
property Hreadyout_penable_write;
@(posedge Hclk) disable iff(!Hresetn)
write_s |-> ##3 Penable && Hreadyout;
endproperty
assert_Hreadyout_penable_write: assert property (Hreadyout_penable_write);

//PENABLE shouldn't be high for 2 cycles continuously
property no_penable_2cycles;
@(posedge Hclk) disable iff(!Hresetn)
Penable |=> !Penable;
endproperty
assert_no_penable_2cycles: assert property (no_penable_2cycles);

// Valid PSEL
property valid_Psel;
@(posedge Hclk) disable iff(!Hresetn)
$onehot0(Pselx);
endproperty
assert_valid_Psel: assert property (valid_Psel);

// For Read transfer next cycle should have corresponding PSEL high for next 2 clocks and should be onehot
```



```

46
47 // Transition from ST_IDLE
48 property p_IDLE_to_WWAIT;
49   @(posedge Hclk) disable iff (!Hresetn)
50     (PRESENT_STATE == ST_IDLE && valid && Hwrite) |-> (NEXT_STATE == ST_WWAIT);
51 endproperty
52 assert_IDLE_to_WWAIT: assert property (p_IDLE_to_WWAIT);
53
54 property p_IDLE_to_READ;
55   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_IDLE && valid && !Hwrite |-> NEXT_STATE == ST_READ;
56 endproperty
57 assert_IDLE_to_READ: assert property (p_IDLE_to_READ);
58
59 property p_IDLE_to_IDLE;
60   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_IDLE && !valid |-> NEXT_STATE == ST_IDLE;
61 endproperty
62 assert_IDLE_to_IDLE: assert property (p_IDLE_to_IDLE);
63
64
65 // Transition from ST_WWAIT
66 property p_WWAIT_to_WRITE;
67   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_WWAIT && !valid |-> NEXT_STATE == ST_WRITE;
68 endproperty
69 assert_WWAIT_to_WRITE: assert property (p_WWAIT_to_WRITE);
70
71 property p_WWAIT_to_WRITEP;
72   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_WWAIT && valid |-> NEXT_STATE == ST_WRITEP;
73 endproperty
74 assert_WWAIT_to_WRITEP: assert property (p_WWAIT_to_WRITEP);
75
76
77 // Transition from ST_READ
78 property p_READ_to_RENABLE;
79   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_READ |-> NEXT_STATE == ST_RENABLE;
80 endproperty
81 assert_READ_to_RENABLE: assert property (p_READ_to_RENABLE);
82
83
84 // Transition from ST_WRITE
85 property p_WRITE_to_WENABLE;
86   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_WRITE && !valid |-> NEXT_STATE == ST_WENABLE;
87 endproperty
88 assert_WRITE_to_WENABLE: assert property (p_WRITE_to_WENABLE);
89
90 property p_WRITE_to_WENABLEP;
91   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_WRITE && valid |-> NEXT_STATE == ST_WENABLEP;
92 endproperty
93 assert_WRITE_to_WENABLEP: assert property (p_WRITE_to_WENABLEP);
94
95
96 // Transition from ST_WRITEP
97 property p_WRITEP_to_WENABLE;
98   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_WRITEP |-> NEXT_STATE == ST_WENABLE;
99 endproperty
100 assert_WRITEP_to_WENABLE: assert property (p_WRITEP_to_WENABLE);
101
102
103 // Transitions from ST_RENABLE
104 property p_RENABLE_to_IDLE;
105   @(posedge Hclk) disable iff (!Hresetn) PRESENT_STATE == ST_RENABLE && !valid |-> NEXT_STATE == ST_IDLE;
106 endproperty
107 assert_RENABLE_to_IDLE: assert property (p_RENABLE_to_IDLE);
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

// Assert valid signal generation logic
assert property (@(posedge Hclk)
  (Hresetn && Hreadyin && (Haddr >= 32'h8000_0000 && Haddr < 32'h8C00_0000) && (Htrans == 2'b10 || Htrans == 2'b11)) |-> (valid == 1'b1));

// Assert that valid should not be high when conditions are not met
assert property (@(posedge Hclk)
  (!(Hresetn && Hreadyin && (Haddr >= 32'h8000_0000 && Haddr < 32'h8C00_0000) && (Htrans == 2'b10 || Htrans == 2'b11))) |-> (valid == 1'b0));

// Assert tempse1x logic for different ranges of Haddr
assert property (@(posedge Hclk)
  (Hresetn && (Haddr >= 32'h8000_0000 && Haddr < 32'h8400_0000)) |-> (tempse1x == 3'b001));

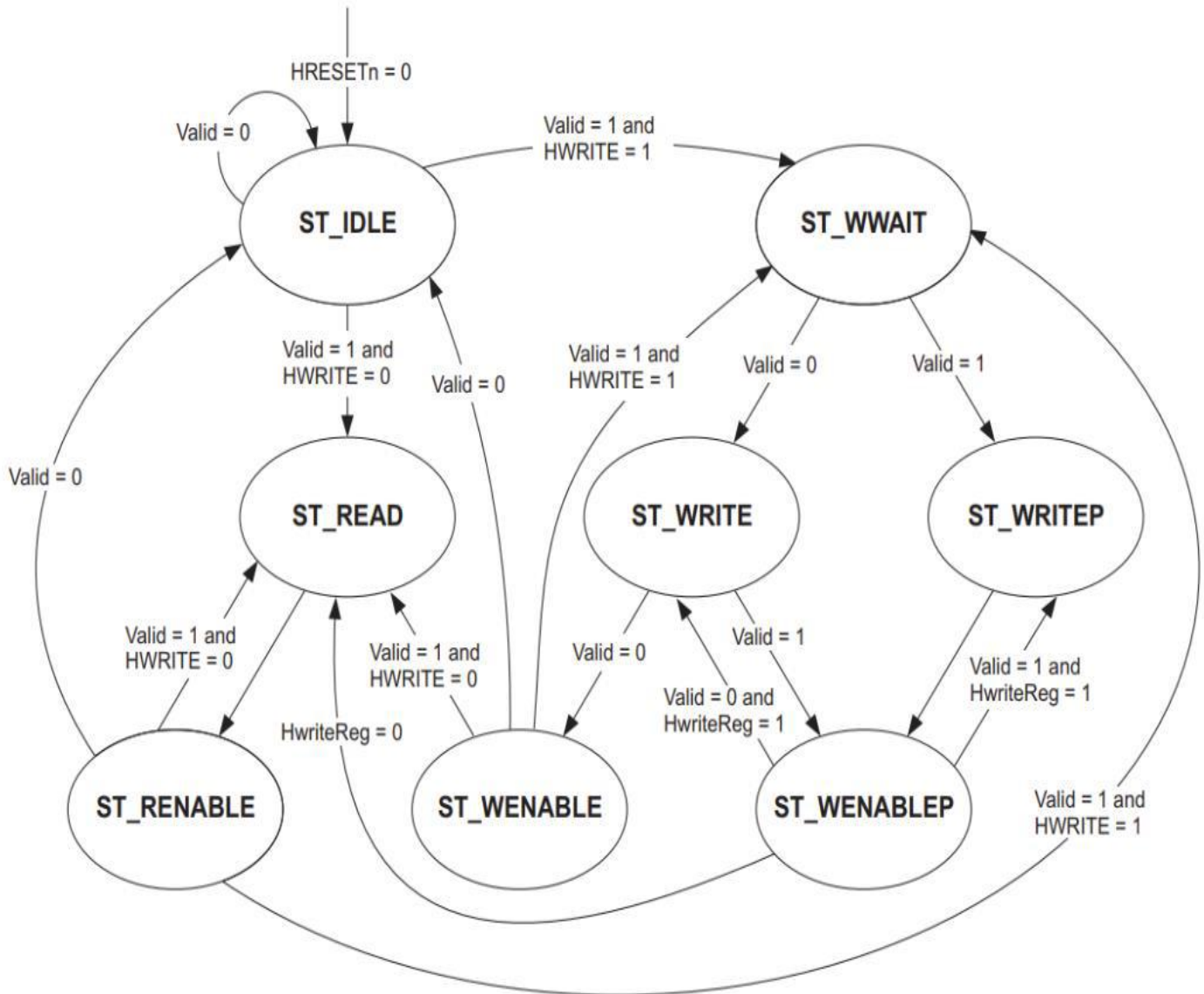
assert property (@(posedge Hclk)
  (Hresetn && (Haddr >= 32'h8400_0000 && Haddr < 32'h8800_0000)) |-> (tempse1x == 3'b010));

assert property (@(posedge Hclk)
  (Hresetn && (Haddr >= 32'h8800_0000 && Haddr < 32'h8C00_0000)) |-> (tempse1x == 3'b100));

// Assert tempse1x should be 000 for invalid ranges of Haddr
assert property (@(posedge Hclk)
  (Hresetn && !(Haddr >= 32'h8000_0000 && Haddr < 32'h8C00_0000)) |-> (tempse1x == 3'b000));

```

## FSM FOR AHB TO APB INTERFACE:



First, we started exploring the design by running the completely automated apps, AEP and FXP, we ran them on the top module with the filelist, as well as on the individual modules for better signal visibility. In both cases, we didn't get much information using those automated techniques.



# AEP:

File View Source OneTrace Tools Window Help

VCF:TaskList

Task List

Name	Progress	Result
AEP	<div><div></div></div>	1:10:00

VCF:GoalList

Time: 12H Max Cycle: -1 <Filter Target & Constraint Tables by Name>

Targets: ALL

status	depth	name	type	location	expression	state_req	state_name	state_val	engine	elapsed time
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_IDLE_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_IDLE	'h0	I5	00:00:50
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_READ_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_READ	'h2	I5	00:00:49
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_READABLE_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_READABLE	'h5	I5	00:00:49
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_WENABLE_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_WENABLE	'h7	I5	00:00:49
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_WENABLE_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_WENABLE	'h6	I5	00:00:53
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_WRITE_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_WRITE	'h4	I5	00:00:53
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_WRITE_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_WRITE	'h3	I5	00:00:53
✓		APB_Controller.i_fsm_checks_AEP.Prop.fsm.PRESENT_STATE_ST_WWAIT_deadlock_safe	fsm_deadlock	...controller.sv:41		PRESENT_STATE	ST_WWAIT	'h1	I5	00:00:54

Task Summary

Constraints: ALL

Task	Property Summary	Assertion	Cover	Constraint	Vacuity	Witness
AEP	Number of Properties	8	0	4	11	0
	Proven (P) / Covered (C)	8	0	-	11	0
	Falsified (F) / Uncoverable (U)	0	0	-	0	-
	Vacuous (V)	0	0	-	0	0
	Inconclusive (I)	0	0	-	0	0
	Checking (CH)	0	0	-	0	0
	Not run (N)	0	0	-	0	0
	Disabled	0	0	0	-	-

Total Properties: 8 - passed[8] - failed[0] - disabled[0] ; Constraints Enabled: 4 ; Run Time: 0-01:10

\*Src1 APB\_Controller.sv VCF:GoalList(AEP)

VC Formal Console

File View Tools Window Help

Line: 76

# FPV\_Bridge:

File View Source OneTrace Tools Window Help

VCF:TaskList

Task List

Name	Progress	Result
FPV	<div><div></div></div>	21:00:00

VCF:GoalList

Time: 12H Max Cycle: -1 <Filter Target & Constraint Tables by Name>

Targets: ALL

status	depth	name	vacuity	witness	type	engine	elapsed time
✓		Bridge_Top.chk_bridge_top.assert_hreadyout_penable_read	✓ 2		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_hreadyout_penable_write	✓ 1		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_no_penable_2cycles	✓ 4		assert	rp1	00:00:03
✓		Bridge_Top.chk_bridge_top.assert_read_addr0	✓ 1		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_read_addr1	✓ 1		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_read_addr2	✓ 1		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_read_after_write_3waitSates	✓ 2		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_same_HPwrite_read	✓ 1		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_same_HPwrite_write	✓ 1		assert	e2	00:00:10
✓		Bridge_Top.chk_bridge_top.assert_same_HR_PR_data	✓ 4		assert	t1	00:00:02
✓		Bridge_Top.chk_bridge_top.assert_valid_Pse1			assert	rp1	00:00:03
✓ 2	2	Bridge_Top.chk_bridge_top.cover_back_to_back			cover	rf1	00:00:01
✓ 3	3	Bridge_Top.chk_bridge_top.cover_burst_of_reads			cover	rf1	00:00:02
✓ 2	2	Bridge_Top.chk_bridge_top.cover_burst_of_writes			cover	rf1	00:00:02
✓ 2	2	Bridge_Top.chk_bridge_top.cover_write_2_addrs			cover	rf1	00:00:02
✓		Bridge_Top.chk_bridge_top.read_after_write1	✓ 6		assert	e2	00:00:09
✓		Bridge_Top.chk_bridge_top.reset_check_Pse1x1	✓ 1		assert	t1	00:00:02
✓		Bridge_Top.chk_bridge_top.reset_check_Pse1x1	✓ 1		assert	t1	00:00:02

Task Summary

Constraints: ALL

Task	Property Summary	Assertion	Cover	Constraint	Vacuity	Witness
FPV	Number of Properties	17	4	8	20	0
	Proven (P) / Covered (C)	17	4	-	20	0
	Falsified (F) / Uncoverable (U)	0	0	-	0	0
	Vacuous (V)	0	0	-	0	0

Total Properties: 21 - passed[21] - failed[0] - disabled[0] ; Constraints Enabled: 8 ; Mn depth: 2 ; Max depth: 3 ; Run Time: 0-00:20

\*Src1 bridge\_top.sv VCF:GoalList(FPV)

VC Formal Console

File View Tools Window Help

Line: 56

56 [Info] LIC RT CHECKOUT: VC Formal run time license checkout. Base1: FPV:1.  
57 [Info] RETRIEVE LEARN DATA: Retrieved learned data from (local directory) Bridge\_Top\_learn\_dir.

# FPV\_APB\_CONTROLLER:

File View Source OneTrace Tools Window Help

VCF TaskList

Task List

Name	Progress	Result
FPV	19:00:16	0

VCF GoalList

Time: 12H Max Cycle: -1 <Filter Target & Constraint Tables by Name>

Targets: ALL

status	depth	name	vacuity	witness	type	engine	elapsed time
✓	1	APB_ControllerAPB_Controller_chk.assert_IDLE_to_IDLE	✓1		assert	e2	00:00:03
✓	2	APB_ControllerAPB_Controller_chk.assert_IDLE_to_READ	✓1		assert	e2	00:00:03
✓	3	APB_ControllerAPB_Controller_chk.assert_IDLE_to_WWAIT	✓1		assert	e2	00:00:03
✓	4	APB_ControllerAPB_Controller_chk.assert_READ_to_RENABLE	✓2		assert	e2	00:00:03
✓	5	APB_ControllerAPB_Controller_chk.assert_RENABLE_to_IDLE	✓3		assert	e2	00:00:03
✓	6	APB_ControllerAPB_Controller_chk.assert_RENABLE_to_READ	✓3		assert	e2	00:00:03
✓	7	APB_ControllerAPB_Controller_chk.assert_RENABLE_to_WWAIT	✓3		assert	e2	00:00:03
✓	8	APB_ControllerAPB_Controller_chk.assert_RENABLE_to_DEAD	✓4		assert	e2	00:00:03

Task Summary

Constraints: ALL

Task	Property Summary	Assertion	Cover	Constraint	Vacuity	Witness
FPV	Number of Properties	19	0	1	19	0
	Proven (P) / Covered (C)	19	0	-	19	0
	Falsified (F) / Uncoverable (U)	0	0	-	0	0
	Vacuous (V)	0	-	-	-	-
	Inconclusive (I)	0	-	-	0	0
	Checking (CH)	0	-	-	0	0
	Not run (N)	0	0	-	0	0
	Disabled	0	0	-	-	-

Total Properties: 19 - passed[19] - failed[0] - disabled[0] ; Constraints Enabled: 1 ; Run Time: 0:00:16

\*Src1APB\_Controller.sv VCF:GoalList(FPV)

VC Formal Console

Rule View Tools Window Help

Line: 54

```
54 sim_save_reset
55 1
56 1
57 1
58 #check fv
59 [Info] FORMAL_I CREATE: Create Formal Model:APB_Controller.
60 [Info] FORMAL_I RUN: Starting formal verification for check fv
61   Id: 0 Goals: 38 Constraints: 0 Block Mode: false
62 [Info] LIC UNUSED WORKERS: 0 unused worker(s) based 1 licenses needed to support 4 workers requested.
63   Use "set_grid_usage" to maximize worker usage and improve performance, if there are sufficient compute resources to support more workers.
64   Each runtime license supports 12 workers.
65 [Info] LIC RT CHECKOUT: VC Formal run time license checkout. Base:1 FPV:1.
66 [Warning] LEARNED DATA NOT FOUND: No existing learned data found in APB_Controller.learn.dir
67 [Info] BITLEVEL_MODEL_STATS: Generated model with 209 gates, 3 inputs, 47 registers, 0 initial constraints, 0 constraints.
68 [Info] BITLEVEL_MODEL_STATS: Generated model with 135 gates, 3 inputs, 44 registers, 0 initial constraints, 0 constraints.
```

# FPV\_AHB:

File View Source OneTrace Tools Window Help

VCF TaskList

Task List

Name	Progress	Result
FPV	6:00:12	0

VCF GoalList

Time: 12H Max Cycle: -1 <Filter Target & Constraint Tables by Name>

Targets: ALL

status	depth	name	vacuity	witness	type	engine	elapsed time
✓	1	AHB_slave_interface.chk_ahb_slave_interface.unnamed\$5_0	✓1		assert	e1	00:00:02
✓	2	AHB_slave_interface.chk_ahb_slave_interface.unnamed\$5_1	✓1		assert	rp1	00:00:02
✓	3	AHB_slave_interface.chk_ahb_slave_interface.unnamed\$5_2	✓1		assert	t1	00:00:01
✓	4	AHB_slave_interface.chk_ahb_slave_interface.unnamed\$5_3	✓1		assert	rp1	00:00:02
✓	5	AHB_slave_interface.chk_ahb_slave_interface.unnamed\$5_4	✓1		assert	rp1	00:00:02
✓	6	AHB_slave_interface.chk_ahb_slave_interface.unnamed\$5_5	✓1		assert	rp1	00:00:02

Task Summary

Constraints: ALL

Task	Property Summary	Assertion	Cover	Constraint	Vacuity	Witness
FPV	Number of Properties	6	0	1	6	0
	Proven (P) / Covered (C)	6	0	-	6	0
	Falsified (F) / Uncoverable (U)	0	0	-	0	0
	Vacuous (V)	0	-	-	-	-
	Inconclusive (I)	0	-	-	0	0
	Checking (CH)	0	0	-	0	0
	Not run (N)	0	0	-	0	0
	Disabled	0	0	0	-	-

Total Properties: 6 - passed[6] - failed[0] - disabled[0] ; Constraints Enabled: 1 ; Run Time: 0:00:12

\*Src1AHB\_Slave\_Interface.sv VCF:GoalList(FPV)

VC Formal Console

Rule View Tools Window Help

Line: 52

```
52 1
53 1
54 1
55 #check fv
56 [Info] FORMAL_I CREATE: Create Formal Model:AHB_slave_interface.
57 [Info] FORMAL_I RUN: Starting formal verification for check fv
58   Id: 0 Goals: 12 Constraints: 0 Block Mode: false
59 [Info] LIC UNUSED WORKERS: 0 unused worker(s) based 1 licenses needed to support 4 workers requested.
60   Use "set_grid_usage" to maximize worker usage and improve performance, if there are sufficient compute resources to support more workers.
61   Each runtime license supports 12 workers.
62 [Info] LIC RT CHECKOUT: VC Formal run time license checkout. Base:1 FPV:1.
63 [Info] RETRIEVE LEARN DATA: Retrieved learned data from (local directory) AHB_slave_interface.learn.dir.
64 [Info] BITLEVEL_MODEL_STATS: Generated model with 169 gates, 35 inputs, 13 registers, 0 initial constraints, 0 constraints.
65 [Info] BITLEVEL_MODEL_STATS: Generated model with 37 gates, 8 inputs, 9 registers, 0 initial constraints, 0 constraints.
```

## **Challenges:**

Specifically, we faced several major obstacles at the start of the above effort. The foremost of these issues involved obtaining a proper RTL (Register Transfer Level) code that met all our specified criteria. In other words, we were looking for RTL facilitated the burst read and write transactions as specified for the ARM AMBA standards. In the light of these challenges, we decided to proceed with the RTL code available for the AHB2APB bridge design, as an integral part of this process.

## **Summary:**

This paper outlines our verification efforts on the AHB-APB bridge, which is a critical link between AHB masters and slave interfaces. Our focus was on read and write transfers, the AHB to APB state machine, handling Read After Write issues, and reset signal handling. We analyzed advanced verification methodologies and shared our experience on the PSELX signal. We used the AMBA specifications to create meaningful assumptions, assertions, and cover properties to cover the given RTL design as much as the RTL design and time allowed us.

## **References:**

- 1) Github
- 2) Lecture recordings
- 3) ARM SPEC DOCUMENT
- 4) Udemy