SDRAM Controller Verification

Mousseau Long Wood Ferguson - Rev0

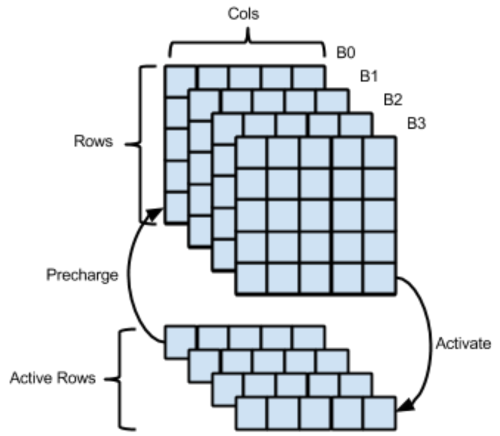
Intro

- We started with
    - “Fully working, Fully verified,” Wishbone compliant SDRAM Controller downloaded from www.OpenCores.org
    - Fully working SDRAM DIMM BFM
- IP is vanilla Verilog.

Intro...

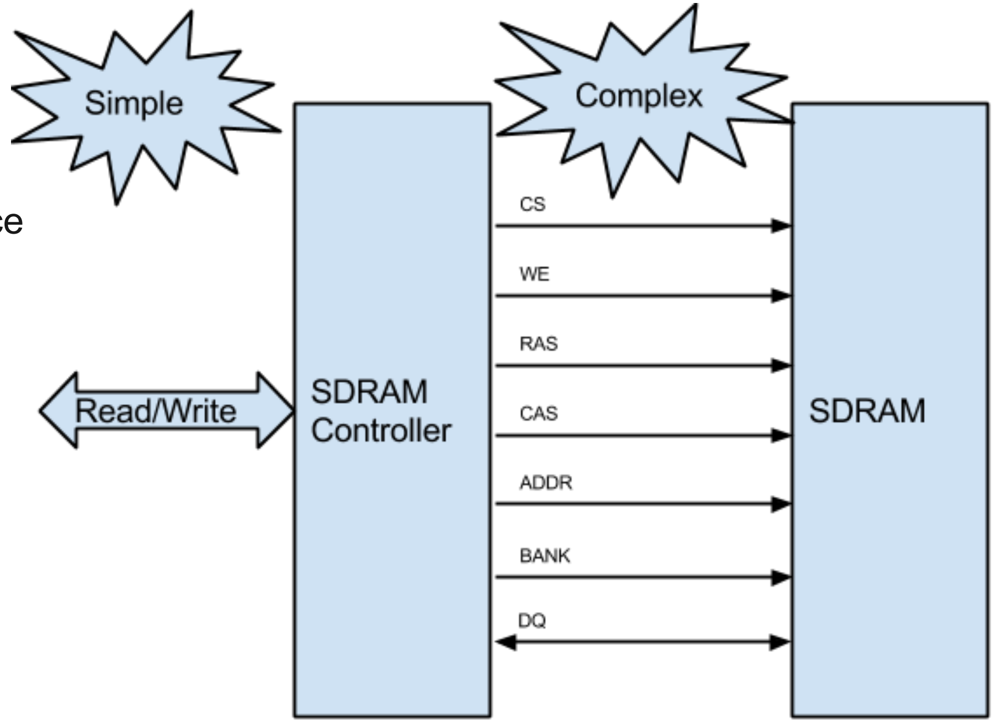
- SDRAM architecture
  - Rows, Columns, Banks
  - Activate, Precharge
  - Mux Col/Row Address(Act,Rd,Wr)
  - Timing…
- SDRAM Commands
  - Read,Write
- Only issue commands in certain states…
  - user must track current SDRAM state…
- Complexify with pin level

So…….

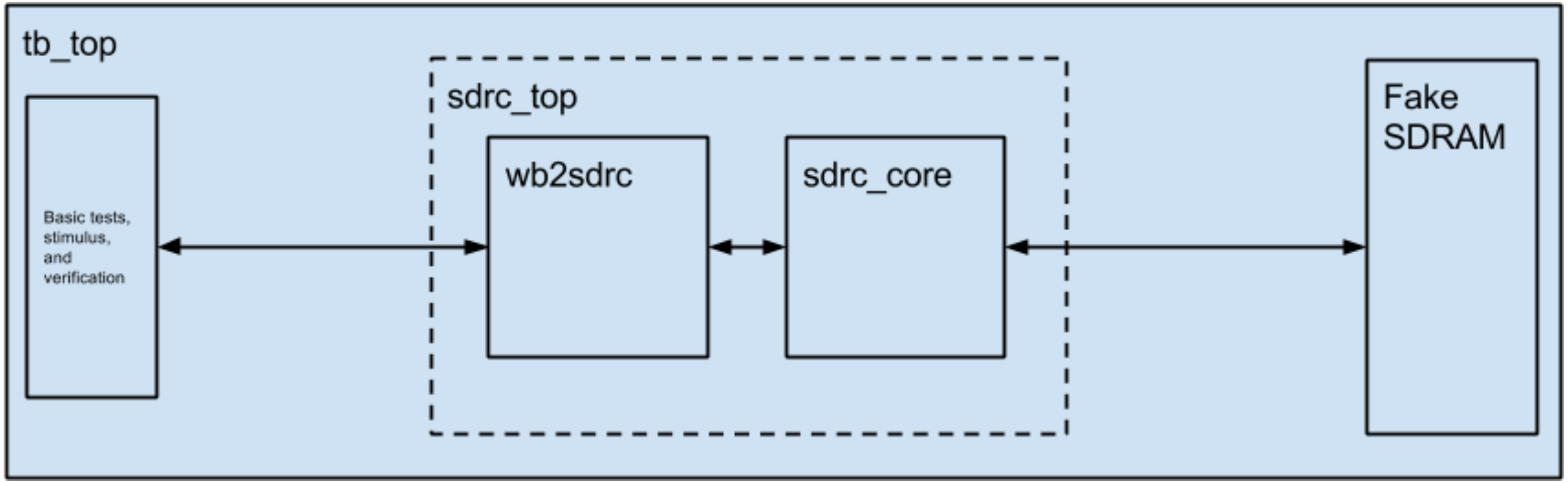


Intro...

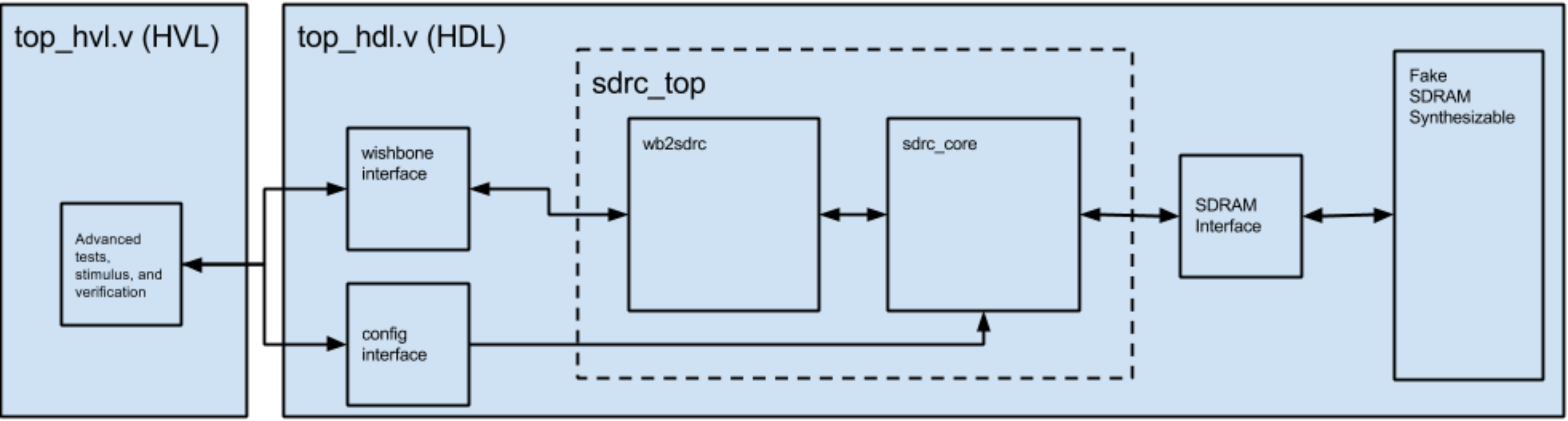
- SDRAM **Controller**
  - Keeps track of SDRAM state
  - Provides a simple user interface



Original IP Hierarchy



Final IP Hierarchy



Logical Validation

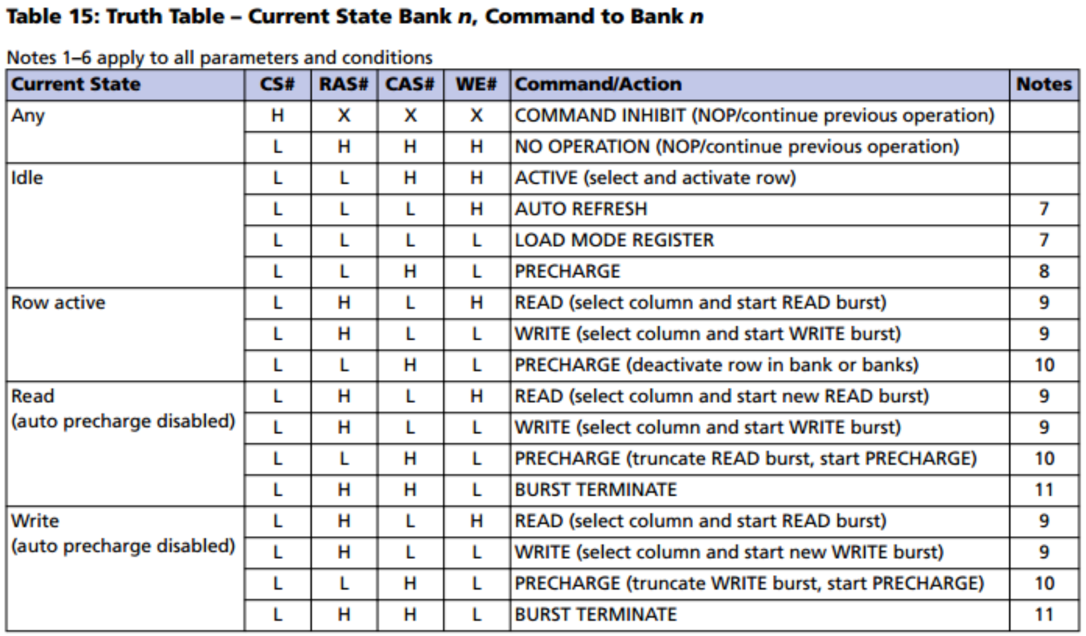


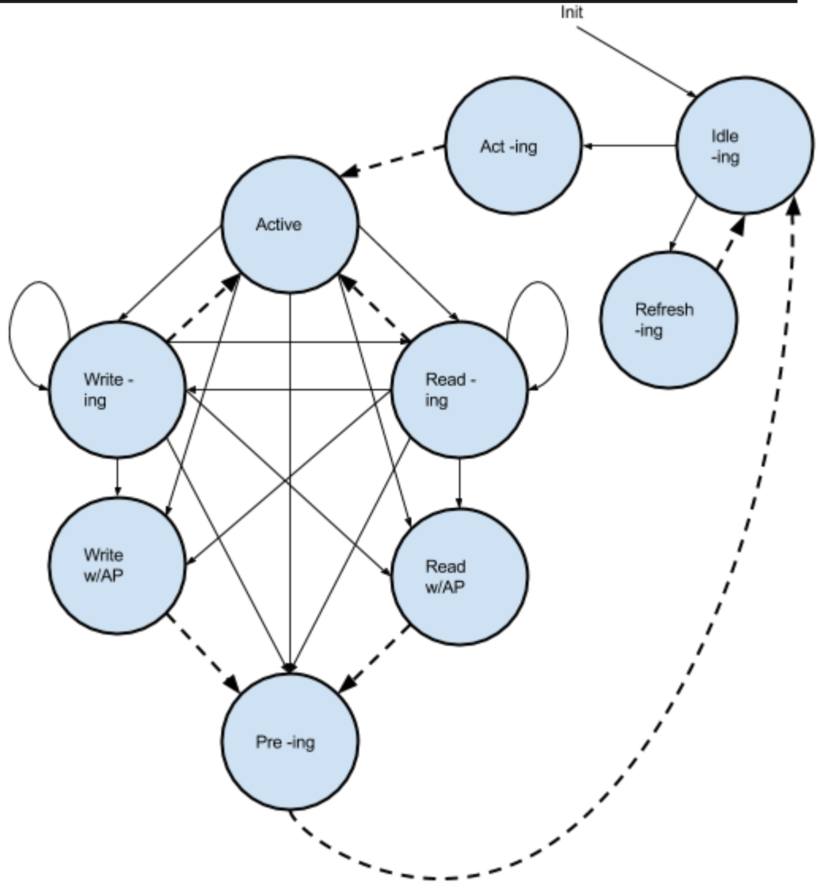
Table 15: Truth Table – Current State Bank *n*, Command to Bank *n*

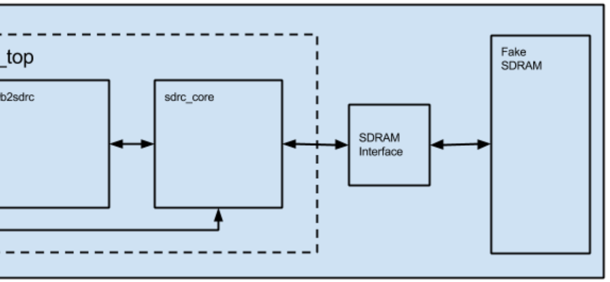
Notes 1–6 apply to all parameters and conditions

Current State	CS#	RAS#	CAS#	WE#	Command/Action	Notes
Any	H	X	X	X	COMMAND INHIBIT (NOP/continue previous operation)	
	L	H	H	H	NO OPERATION (NOP/continue previous operation)	
Idle	L	L	H	H	ACTIVE (select and activate row)	
	L	L	L	H	AUTO REFRESH	7
	L	L	L	L	LOAD MODE REGISTER	7
	L	L	H	L	PRECHARGE	8
Row active	L	H	L	H	READ (select column and start READ burst)	9
	L	H	L	L	WRITE (select column and start WRITE burst)	9
	L	L	H	L	PRECHARGE (deactivate row in bank or banks)	10
Read (auto precharge disabled)	L	H	L	H	READ (select column and start new READ burst)	9
	L	H	L	L	WRITE (select column and start WRITE burst)	9
	L	L	H	L	PRECHARGE (truncate READ burst, start PRECHARGE)	10
	L	H	H	L	BURST TERMINATE	11
Write (auto precharge disabled)	L	H	L	H	READ (select column and start READ burst)	9
	L	H	L	L	WRITE (select column and start new WRITE burst)	9
	L	L	H	L	PRECHARGE (truncate WRITE burst, start PRECHARGE)	10
	L	H	H	L	BURST TERMINATE	11

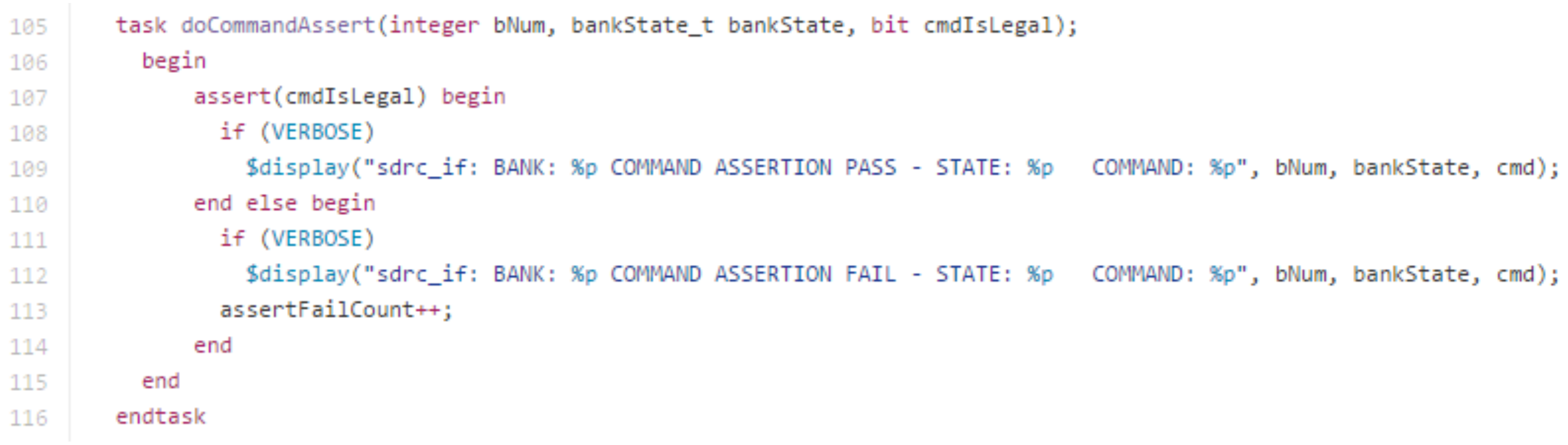
Logical Validation - SDRAM FSM

- Tracking DIMM state
  - snooping commands from SDRC.
  - Useful for logical verification of commands from the controller as well as indirect testing of some command timing requirements
- Instantiated in the interface, one state machine per bank in the SDRAM (4 total)





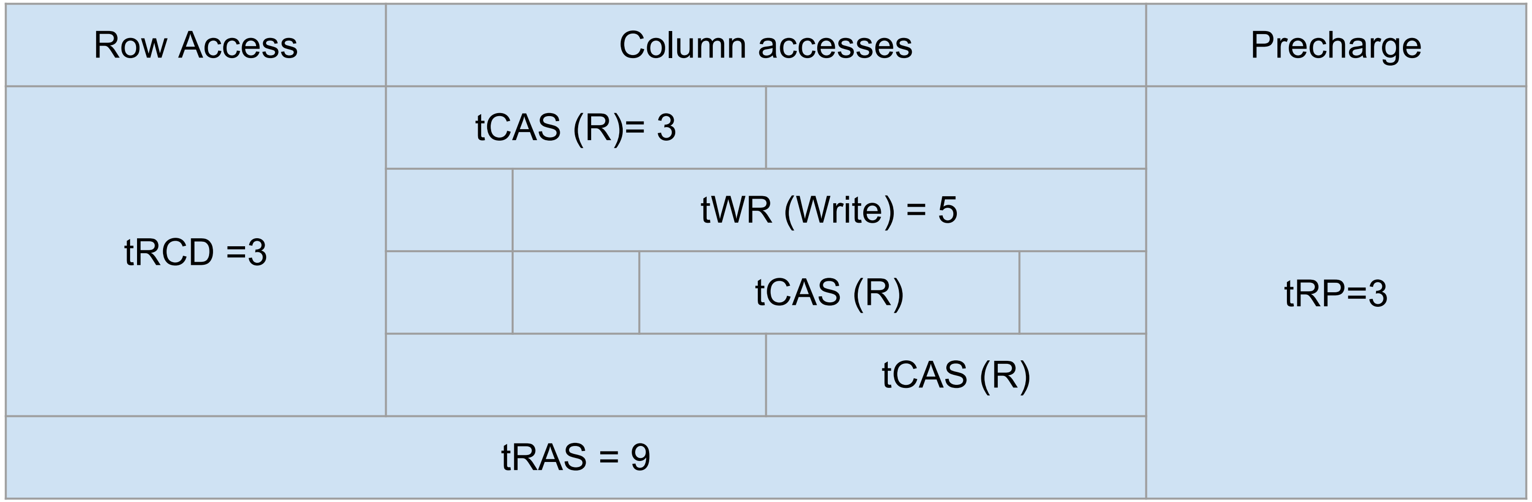
Logical Validation - Immediate Assertions



```
task doCommandAssert(integer bNum, bankState_t bankState, bit cmdIsLegal);
 begin
   assert(cmdIsLegal) begin
    if (VERBOSE)
     $display("sdrc_if: BANK: %p COMMAND ASSERTION PASS - STATE: %p   COMMAND: %p", bNum, bankState, cmd);
   end else begin
    if (VERBOSE)
     $display("sdrc_if: BANK: %p COMMAND ASSERTION FAIL - STATE: %p   COMMAND: %p", bNum, bankState, cmd);
    assertFailCount++;
   end
 end
endtask
```

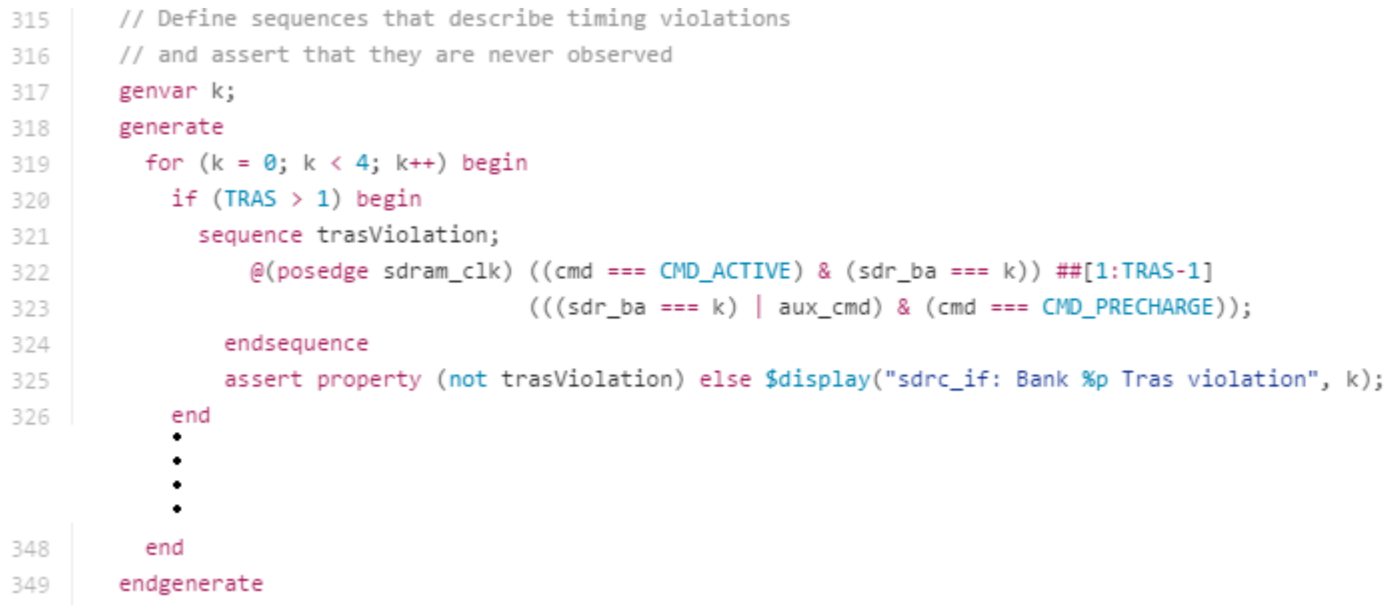
Timing Validation - Concurrent Assertions

SDRAM access timing



(http://en.wikipedia.org/wiki/SDRAM_latency)

Timing Validation - Concurrent Assertions



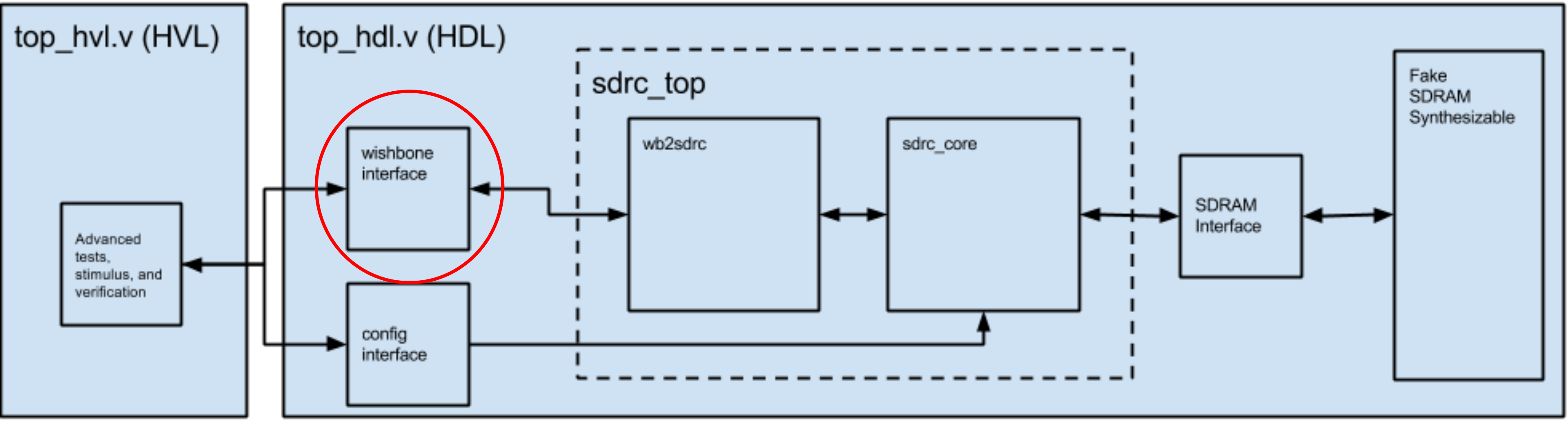
```
// Define sequences that describe timing violations
// and assert that they are never observed
genvar k;
generate
  for (k = 0; k < 4; k++) begin
    if (TRAS > 1) begin
      sequence trasViolation;
          @(posedge sdram_clk) ((cmd === CMD_ACTIVE) & (sdr_ba === k)) ##[1:TRAS-1]
                               (((sdr_ba === k) | aux_cmd) & (cmd === CMD_PRECHARGE));
        endsequence
        assert property (not trasViolation) else $display("sdrc_if: Bank %p Tras violation", k);
    end
    •
    •
    •
    •
  end
endgenerate
```

Testbench - Assertion Verification

One aspect of the testbench verifies the assertions do what they’re intended to

- Instantiate dummy interface and drive directly from testbench
  - Out of order commands
  - Logical Timing Violations
  - Signal Timing Violations

Final IP Hierarchy



Wishbone IF

- Standard SV interface
- Simplified pins (we tried moving it around a bunch till architecture was finalized)
- Put read & write methods into the IF itself
  - They were *not* synthesizable … BAD!
  - Split methods, moved non synth part to HVL … GOOD!
- Having this be the sole method of communication twixt HVL & HDL pointed us to using the BFM model for Veloce
  - Simple TBX model, only change needed to code-base was adding pragmas
  - Simulation and Emulation are the same
    - Using SCIMI pipes, for example, only works in emulation
- Mitigates risk that we might need major rework when we get to Emulation stage

Testbench

End to end verification ensures that the SDRC properly drives the DIMM

- Driving SDRC - wishbone interface function
    - `top_hdl.wbi.write(address, burstLength, data);`
- Types of tests
    - Directed
        - Write then read
        - Write burst, read burst
        - Write bursts with page boundary crossing, read burst
            - reading from a row, must close before moving to another row
        - Large numbers of writes/reads
            - SDRC handles ‘auto-refresh’
    - Randomized
        - Random address, random burst length
        - Random address, random burst length, random number of reads between writes

Veloce -- bringup

- Planned to be as close as possible to Simulation
- Ended up wanting to synthesize a RAM
  - Only time to make one (major loss was self-checking)
- Started by understanding Sameer’s BoothBFM example
- Puresim seemed easy and hopes were high
- Veloce mode … not so much
  - Inter assignment delays are verboten
    - RAM used ‘em
    - Core module used ‘em
  - Multiple clocks in always blocks verboten
    - wishbone IF used ‘em
  - Uncontrolled & Controlled nets merge within a clock tree
    - unfamiliar with Veloce HW

Results

- Assertion Tests
  - Immediate - states/commands
    - Of the 320 possible state/command commands, 192 will trigger assertion fails
    - Testbench triggers 192 assertions. Success!
  - Concurrent - timing assertions - table on next page
- End to End tests Tests
  - Approximately 1000 writes and 1000 reads, all pass
    - if we increase to 10,000+ random tests, we see some timing errors likely from Wishbone interface
- Veloce
  - not running yet, assume we will see same results
- Found ~3 strange bugs during testing
  - they were all introduced by us

Results - Concurrent Assertions

Testing two different values for each TRAS, TRCD, etc.

Timing Parameter	# failed	# expected	Timing Parameter	# failed	# expected
TRAS	3	3	TRAS	7	7
TRCD	2	2	TRCD	8	8
TRP	1	1	TRP	5	5
TWR	0	0	TWR	3	3

```
TRAS_D    = 4;
TRCD_D    = 2;
TRP_D     = 2;
TWR       = 1;
```

```
TRAS_D    = 8;
TRCD_D    = 5;
TRP_D     = 6;
TWR       = 4;
```

Results - State Counts Bank 0

State	Count
Idle	941
Activating	400
Active	3934
Refreshing	540
Write	1794
Write w/ Precharge	0
Read	1740
Read w/ Precharge	0
Precharging	500

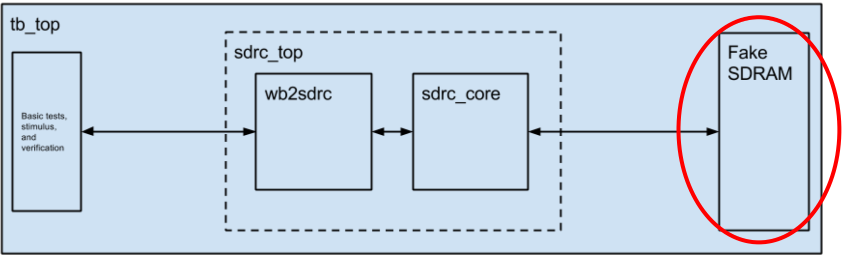
Q & A

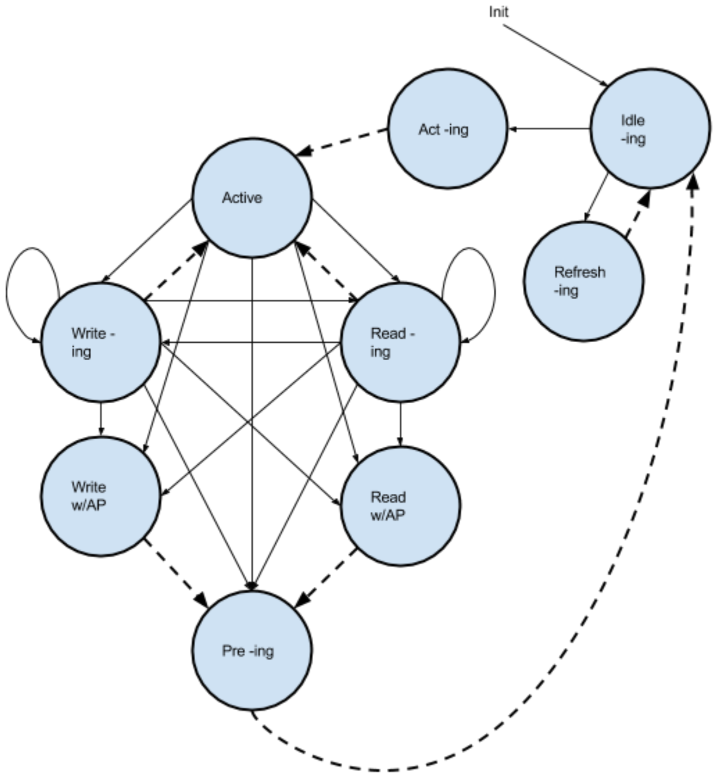
Opens & Unknowns

- Back to back reads to the wishbone interface without proper delay seem to produce faulty controller output
    - Did not root cause.

Research

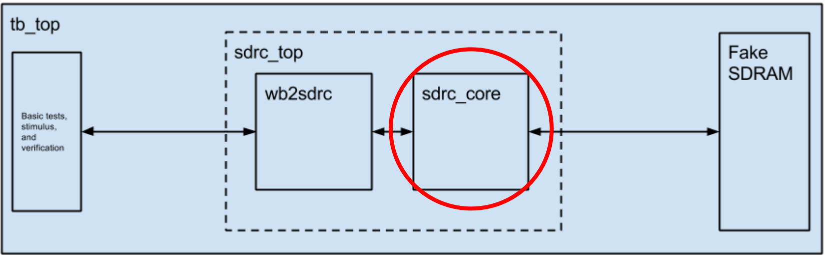
- If this were REAL
- In reality was much simpler

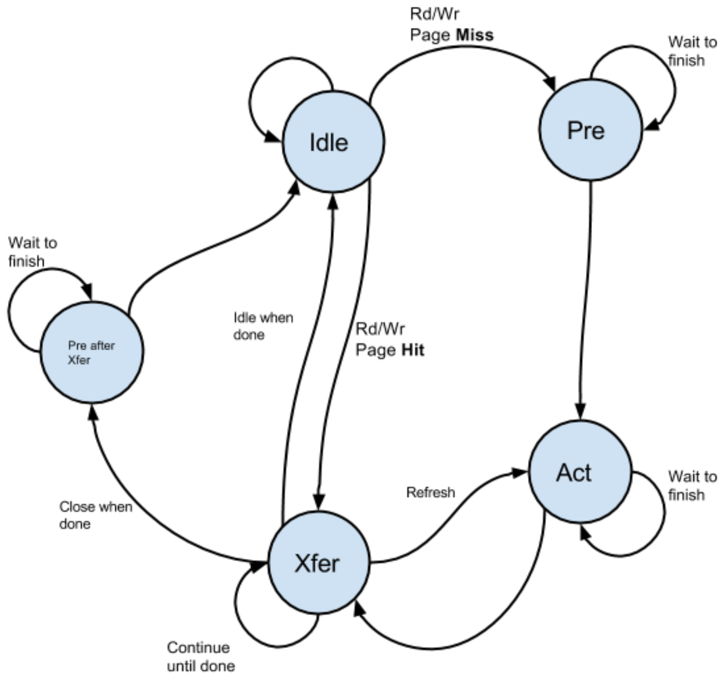




Research

- Two things
  - Initialization management
  - Only activates a row if necessary





Story Arc

- Introduction
  - SDRAM
  - Controller
  - Starting architecture
- Getting Started
  - Research
- Validation Strategy
  - Interface
    - wbinterface
    - cfginterface
    - sdrc_if
      - SDRAM FSM
      - State Tracking
      - Assertions
        - Immediate
        - Concurrent
    -
  - Testbench
    - End-to-end
    - Assertion Triggering
  - Veloce
    -
    - HVL

Wishbone Interface

Originally just wrapper for opencores SoC

Made into SV interface

Added read and write functions

This allows use of TBX BFM model