Advanced Formal Verification Methodologies for a Read-Only Cache with Out-of-Order Support

Di Wu
Black Sesame Technologies

About Black Sesame Technologies

- Automobile solution provider since 2016
- Autonomous driving chip
  - Automotive Image Processing ISP
  - Automotive Deep NN Accelerator
- Teams in San Jose, CA and China

Outline

- Design overview
- Formal verification methodology
- Properties and models
- Coverage and sign-off
- Future work
- Q&A

Design overview

Design overview

Universal cache

- Address fields

Address		
Tag	Index	Offset
TAG_WIDTH-1:0	INDEX_WIDTH-1:0	OFFSET_WIDTH-1:0

INDEX_NUM = 2 ** INDEX_WIDTH

- Cacheline structure

Index	Way0
0	cacheline
1	cacheline
...	
INDEX_NUM-1	cacheline

Cacheline = valid + dirty + tag + data

- Interface

Design overview

Out-of-order read-only cache details

- Read-only
- Direct mapping cache line
- Support out-of-order on CPU interface
  - unique id for each transaction
  - up to 64 outstanding transactions
- MEM return in order data response for cache request
- Credit control on CPU and MEM side
- Fixed priority for respond to CPU interface, miss > hit

Formal verification methodology

Formal verification methodology

Why formal?

- Formal friendly design
  - Clear boundary with well-defined interface constraints
  - Functions can be summarized to key properties

Formal property verification	Traditional simulation
Easy to set up bench for certain designs	Complex bench setup
Easy to hit corner cases	Rely on random or targeted test
Reusable core checker modules	Re-developed scoreboard
Use machine-time to save man-time	Need experienced engineer's efforts

Properties and models

Models

Set-based transaction model

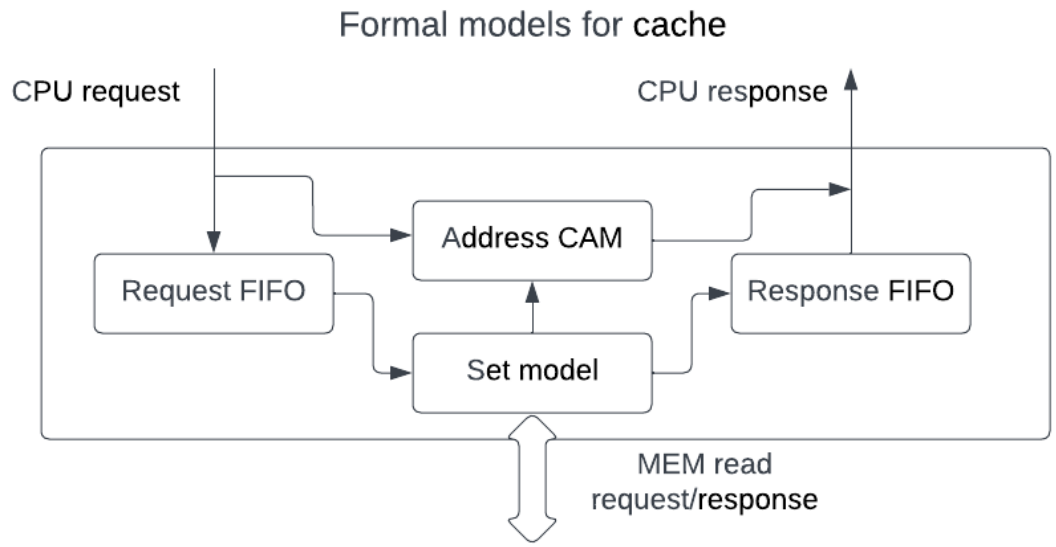
- Fully model one cache line, including control status and data
- Take RTL signals as input to transit between status
- Update data when getting ack from MEM interface

State	Trigger	Next state	Note
IDLE	CPU request miss	PENDING	Cacheline invalid
PENDING	MEM read quest	REQUEST	Miss and pend
REQUEST	MEM read response	VALID	MEM request send and waiting
VALID	replaced	IDLE	Cacheline valid

Models

Credit, request/respond model

- Credit model
  - Count credit issue and consume
  - Support credit checks and constraints
- Memory model
  - Tagram/dataram model for target set



- Request/respond model
  - Use content addressable memory (CAM) in model to store request ID and address mapping
  - Use one FIFO to store incoming CPU requests pending hit/miss check
  - Use one FIFO and priority model to store available response ready for CPU

Properties

Checks

- Set-based transition checks
  - When cache sends read response to CPU, the corresponding cacheline is valid in cache

    rd_resp_sym |->

    !rd_outqueue_empty && (rd_resp_addr_out == rd_outqueue_addr)
  - When cache sends read response to CPU, the data should match what received from MEM

    rd_resp_sym |-> *rd_resp_data == rd_outqueue_data*
  - When cache sends read request to MEM, a read request from CPU should be pending on this address and the requesting cacheline is currently missing

    mem_rd_req_sym |-> *(model_state == REQUEST)*

Properties

Checks

- Specified internal checks
  - Miss queue transaction and address checks
- Credit checks
  - Cache will only send read request to MEM when credit available
  - Cache will only send ack to CPU if request is previously received
- End-to-end forward progress
  - A read request from CPU should be replied by cache within certain cycles
- State-based forward progress
  - Counter based. Only count pending cycles if not blocked or stalled.
  - If a miss is detected, cache should send MEM request within certain cycles.
  - After MEM return data, cache should reply to CPU within certain cycles

Properties

Constraints

- CPU interface
  - Valid-ready protocol
  - CPU will only send read request when credit is available
  - No reuse of outstanding request id
- MEM interface
  - Valid-ready protocol
  - MEM will only send credit when current available credit does not exceed max
  - MEM will only respond data when cache is requesting
  - MEM will eventually respond to cache’s request

Coverage and sign-off

Design Abstraction

Method	Target
Symbolic variable	index
Blackbox	tagram, dataram
Shrink parameter	data width, queue sizes

- Purpose: reduce complexity in FPV run
- Symbolic variable
  - Defined as stable undriven logic within a range
  - Worked in FPV to represent ANY value in the range
  - Reduce model and check size to only serve the part matches symbolic variable
- Blackbox
  - Blackbox large-scaled storage structure, such as memory
  - Use partial-covered model on blackbox interface as replacement
- Shrink parameter
  - Manually set reduced parameter to shrink design, which won't affect control logic

Checker summary

- Properties
  - 90 total checks, 24 proven, 66 inconclusive
  - 5 covers, 4 covered, 1 inconclusive
  - 32 constraints
- Bugs found
  - 2 are medium level (random simulation could find)
  - 1 corner case (hard to find through random simulation)
    - Miss queue forwarding logic has condition issues, resulting wrong CPU response data
    - Found by CPU response data check, after design abstractions

Sign-off procedure

- Sign-off procedure
  - Check and constraint review
  - Proof-depth analysis
  - Deep bug hunting run
  - Line and toggle coverage
  - Formal core coverage
  - Fault injection

Proof-depth analysis

- Result
  - Core checks (set_based transition checks) reached 20-22 proof depth
- Proof-depth analysis
  - Minimum turnaround time from CPU request to response takes 14 cycles in this RTL design
  - MEM request to response takes at least one cycle and no upper limit by constraints
  - 20 proof depth covers at least 6 request full lifecycle, which can cover miss-on-miss, hit-on-miss, hit-on-hit cases thoroughly

Coverage

Formal core coverage shown in COV app

Name	Score	Line	Toggle	FSM	Condition
net_cache_top	59.32%	40.63%	42.57%	100.00%	54.09%
bst_lsu_cache_sva_checker	62.50%	54.89%	20.58%	100.00%	74.54%
net_cache_dataram_model_0					
net_cache_dataram_model_1					
net_cache_shmem_credit_counter	80.12%	66.67%	73.68%		100.00%
net_cache_shmem_os_counter	81.48%	66.67%	77.78%		100.00%
net_cache_shmem_os_pending_co...	80.12%	66.67%	73.68%		100.00%
net_cache_tagram_model_0					
net_cache_tagram_model_1					
u_dataarb	9.16%	0.00%	18.76%		8.72%
u_dataram					
u_drpipe	41.05%	30.10%	36.85%		56.19%
u_dwpipe	50.07%	9.94%	51.33%		88.95%
u_lst	86.31%	60.05%	98.89%		100.00%
u_lsuif	62.77%	56.07%	51.93%		80.31%
u_lup	30.58%	16.73%	40.72%		34.29%
u_lup_sym_sva	39.42%	25.71%	48.95%		43.59%
u_rfreq	77.55%	56.25%	76.40%		100.00%
u_rfrsp	74.83%		49.65%		100.00%
u_rfrsprmqidq	85.82%	76.74%	80.72%		100.00%
u_rhq	85.32%	74.36%	81.59%		100.00%
u_rmq	87.48%	73.58%	88.85%		100.00%
u_shmif	50.02%	83.33%	14.74%		52.00%

Future work

Future work

For other caches

- Read-write cache
  - Expand model states to include write-related
  - Write request/response and write-back checks
  - Invalidation/flush checks
  - Read/write arbitration checks
- Set-associative cache
  - Develop replacement model according to spec, such as LRU
  - Use model to determine expected hit/miss
- In order cache
  - No need to use CAM to store ID
  - Only need one FIFO in request/response model to store address

Future work

Formal tool features

- Proof assist
  - Analyze portion of time spent on each part of logics
  - Guidance further abstraction
- Fault injection
  - Use FTA to automatically inject faults to RTL to make sure that checks are working

Q & A


THANK YOU
YOUR
INNOVATION
YOUR
COMMUNITY