Finding the Last Bug in a CNN DMA Unit

Bruno Lavigueur – Synopsys, Ottawa, Canada, brunola@synopsys.com
Dean Ke, Eric Rao – Synopsys, Wuhan, China, {Xinxiang.Ke, Jinli.Rao}@synopsys.com
Fergus Casey – Synopsys, Mountain View, USA, fcasey@synopsys.com
Achin Mittal, Pallavi Kumari – Oski Technology, Gurgaon, India, {amittal, pkumari}@oskitech.com
Michael Thompson, Roger Sabbagh – Oski Technology, Ottawa, Canada, {mthompson, rsabbagh}@oskitech.com

***Abstract*- Functional formal sign-off of design blocks provides the benefit of finding all bugs, no matter how resistant they are to discovery through traditional methods. The Direct Memory Access (DMA) unit of a Convolutional Neural Network (CNN) is a design that has very long memory volume transport latencies, which is a characteristic that would conventionally cause it to be considered incompatible with formal verification. However, in this paper, we present a repeatable process that enables the successful application of formal sign-off to such designs. We will show how we were able to find extreme corner case bugs and have confidence that we had found the last bug.**

I. Introduction

Design verification remains the biggest challenge in semiconductor design. To address this challenge, the industry has widely adopted specialized hardware verification languages, such as SystemVerilog, and advanced methodologies, such as UVM. However, these methodologies are probabilistic in nature and do not provide a way to guarantee that the last bug in a design has been found and fixed. In contrast, formal sign-off is exhaustive and can provide complete verification, equivalent to simulating all possible scenarios.

The DesignWare ARC Embedded Vision (EV) Processor [1] offers an optional Convolutional Neural Network (CNN) engine for object detection, classification and scene segmentation. A key component of the CNN is the DMA unit, as shown in Fig. 1, which performs bulk transfer of data between the CNN and main memory. This enables, for example, fast loading of convolutional layer weights and new data samples into the CNN.

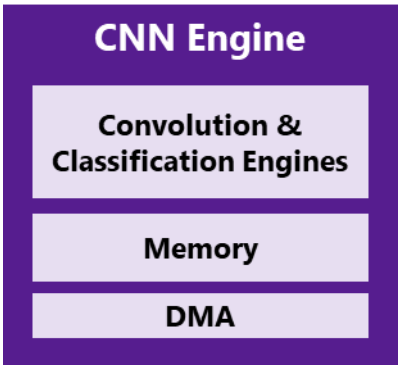

Figure 1. CNN with DMA

With the CNN, the EV processor can execute on-chip, deep learning tasks for artificial intelligence applications such as self-driving cars, facial recognition, video surveillance and virtual/augmented reality. Many of these applications have safety implications, thus ensuring that the last bug has been found is of primary importance.

II. Block-Level Formal Sign-Off

Block-level formal sign-off is Level 4 of the formal capability maturity model [2]. The methodology consists of the following primary components, known as the "4 Cs" [3]:

- End-to-end Checkers
  Level 4 formal includes end-to-end checkers, which differ significantly from the local checkers used in Levels 1 to 3 formal. End-to-end checkers model the end-to-end behavior of the block under test. They monitor the design inputs and predict the expected response at the output, much like a reference model or scoreboard. However, they must be coded using special techniques that make them "formal friendly"

so that they add only the minimal amount of complexity overhead. These techniques often exploit non-determinism in the formal environment, as in the case of the Wolper [4] method for checking data integrity.

- Constraints
  Constraints are required to filter out illegal input space combinations, however they must be validated to ensure bugs are not missed due to over-constraints. One of the most common strategies for constraint validation is simulation integration. With this approach, constraints are integrated into the simulation environment for the design under test as assertions, typically with the simulation running at a higher level of design hierarchy, such as the sub-system level. A failure of these assertions may be an indication that the constraints can be violated during normal operation of the design, which means they are too restrictive and should be reconsidered.
- Complexity Management
  Formal sign-off of end-to-end properties on designer-sized blocks will often require complexity management techniques. Decomposition is used wherever possible, to break up the problem into more tractable pieces. Abstraction models are also used to enable formal analysis to extend beyond the default complexity barriers [5]. Abstraction models reduce the number of cycles required to hit deep states of the design under test.
- Coverage
  Formal coverage provides a measure of both controllability and observability, making it a very strong sign-off metric.
  Controllability coverage reports the reachability of code and functional coverage points, which helps to gauge the depth of analysis that has been achieved in the formal run. Areas where the depth of analysis is insufficient to hit the coverage goals are an indication of the need for additional complexity management techniques.
  Observability metrics report the extent and quality of checking performed by the formal testbench. Coverage holes in this area are an indication that the checkers are incomplete.

This paper discusses how block-level formal sign-off was applied to the CNN DMA unit.

III. Case Study: CNN DMA

A block diagram of the DMA unit is shown in Fig. 2. The DMA has three primary subcomponents. The DMA Controller (DMAC) buffers DMA requests and issues them to the Inbound DMA (IDMA) and Outbound DMA (ODMA) blocks, which respectively control transfers to and from local memory. The DMA is a challenging design for formal verification because:

- The data volumes can be large and the transfer times for a single request can be in the tens of thousands of clock cycles. To reach all states of the design may require multiple transfers to be processed. Formal engines cannot analyze to such large cycle depths by default.
- The data volume may be defined in three or four dimensions, and may not be on contiguous system memory addresses, which complicates predicting the end-to-end behavior of the design for all possible cases.
- The DMA includes a compression mode to reduce the size of the data volumes transferred to the system memory. Algorithmic functions, such as compression, are generally too complex for property checking tools.

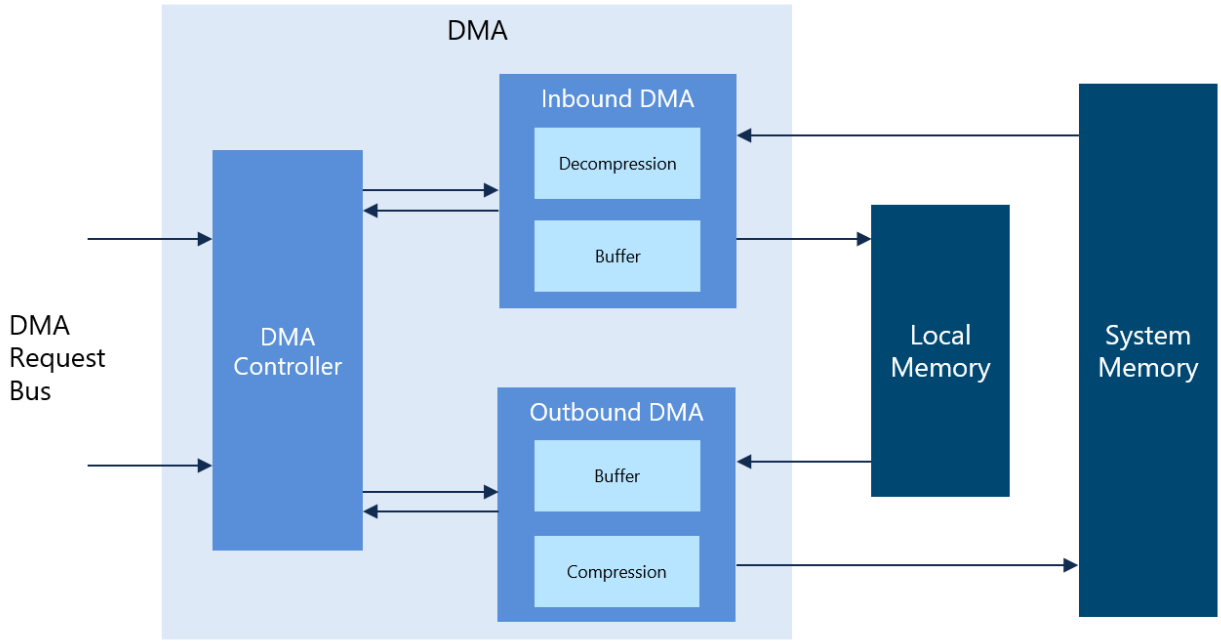

Figure 2. DMA Unit Block Diagram

The "4 Cs" sign-off methodology was applied to the DMA unit as follows:

Exhaustive List of Checkers

A key step in the formal sign-off process is the development of an exhaustive list of checkers. The list of checkers is exhaustive because, in addition to the interface protocols, it also captures the entire set of requirements for the end-to-end behavior of the design. It was carefully reviewed by verifiers and designers to ensure that any potential bug in the DMA unit would be caught by one of the checkers on the list.

Below, we share examples of the checkers for the DMA unit:

- Checkers for integrity of the transferred data
  Read data from system memory is buffered and formatted before being stored in local memory. To verify data integrity, we wrote the checker in two parts:
  - From read data input port to buffer
    - If the read data is accepted by IDMA, then it should be updated with correct formatting in the buffer.
  - From buffer to store output port
    - If there is sufficient data* available in the buffer after formatting, then store to local memory.
    - At the time of store, the byte enable port should correspond to the data transfer mode.
- Checkers for address correctness of the loads/stores
  The DMA unit supports a very large variety of data transfer modes, which in turn lead to many address calculation methods for different modes. The design uses pointers to keep track of the memory location to which a load or store needs to be issued. For each load and store issued by the design we predicted the updated value of the pointers and compare them against those calculated by the design.
- Checkers to verify the compression and decompression mode operation
  We isolated the logic that related strictly to the compression/decompression scheme and placed it within an abstract box. We identified the registers that fed data into this box and added cut-points on them. We then identified the wires that wrote the output data from this box into buffers and coded checkers on them. The reference model for these checkers was based on the compression scheme as described in the specification document.
- Performance checkers
  We developed performance checkers for inbound and outbound DMA transfers. Decomposition of the checkers was required to reduce complexity.

For example, we split the IDMA end-to-end performance checker into three smaller ones:

- IDMA should send a read request to system memory in each cycle if:
  - The external memory is ready to accept the request
  - The local memory is not stalling the store requests
  - The required data for transfer to local memory has not been completely read from system memory
- IDMA should send a store request to local memory in each cycle when the following conditions are met:
  - The local memory is ready to accept the request
  - Sufficient data* is available in the buffer after formatting
- IDMA should assert *fini* signal (to DMAC) to indicate that the data transfer is finished in the same cycle as the last store to local memory is accepted.

***** We calculate the "sufficient data" using the specification document for DMA, which describes the various transfer modes.

Complexity Management

To manage complexity, we used the strategies of decomposition and abstraction models.

Decomposition

We used the decomposition strategy in two ways:

1. We partitioned the DMA into three DUTs and developed formal test bench for each separately, as shown in Fig. 3.

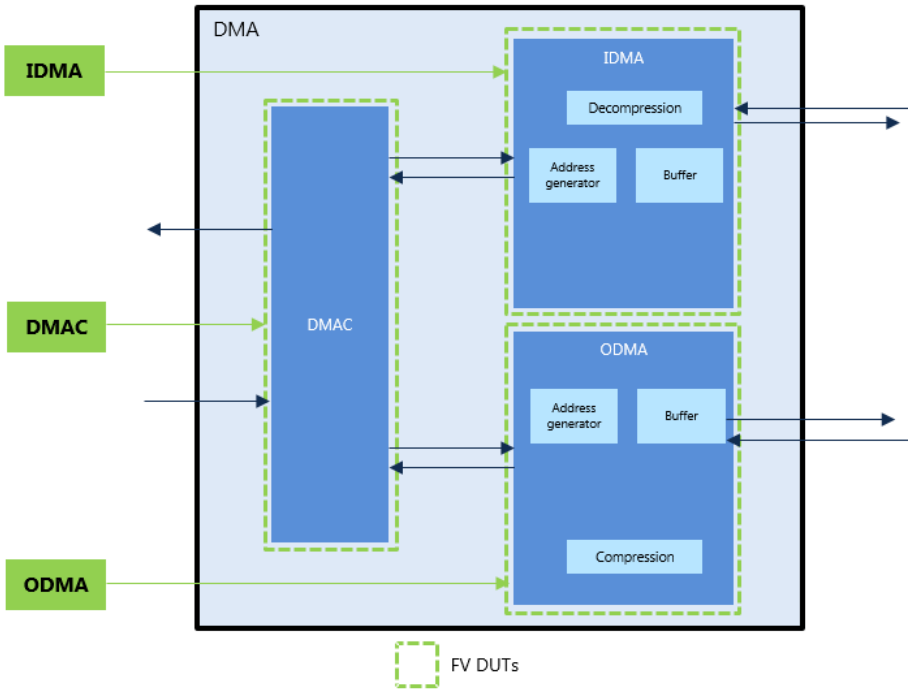

Figure 3. Formal testbench partitioning

This partitioning is possible because of the following reasons

- The three major functionality groups are contained in separate sub-units
- The interfaces between the sub-units are small and well defined. Due to this, the effort required to model the interface behavior was contained.

2. We verified the algorithmic portion of the compression and decompression functions separately from the control portion of the design.

Abstraction Models

Two main abstraction models worked in tandem:

1. Copy Volume Abstraction

   We abstracted the data transfer volumes to reduce the number of cycles required from start to finish of a transfer. To implement this abstraction, we added cut-points on the counters that keep track of the volume of data read from the source memory and the volume of data stored in the target memory.

   For example:

   - Assuming a copy volume – 40x50x80 (=20k units), and 3 cycles per unit of volume, it would take **60k+delta** cycles to copy the entire volume, as shown below:

   - With the abstraction model, the number of cycles required to copy the entire volume is reduced by several orders of magnitude. Multiple paths to completion can be covered in a small number of cycles, as shown below:

   To complete the abstraction model, we needed to add some constraints on the counters. For example, let us consider that the data volume is defined in three dimensions: *x*, *y* and *z*. To accomplish a data transfer in this case, we will have two sets of three counters, corresponding to *x*, *y* and *z* dimensions, for both the source and target memory interfaces. We needed the following constraints on these counters:

   - Constraints on individual counters
     - The *x*, *y* and *z* counters should never exceed their respective maximum values. The maximum values are given in the DMA transfer descriptor.
     - The *x*, *y* and *z* counters should be initialized to 0 for a new descriptor.
   - Relations between *x*, *y* and *z* counters
     - The *y* counter can toggle only when the *x* counter is at its maximum value.
     - The *z* counter can toggle only when the *x* and *y* counters are at their maximum value.
   - Relations between counters on the source and target memories
     - For IDMA, the counters for the internal memory should always lag those for the external memory
     - For ODMA, the counters for the external memory should always lag those for the internal memory

2. Address Abstraction

   The DMA unit supports multiple transfer modes (e.g. three or four dimensions), which enable a wide range of memory address space transfers. It also complicates the address generation logic and we thus faced complexity in verifying the correctness of the generated addresses for accesses to both the internal and external memories.

   To resolve this, we abstracted out a portion of the address generation logic by applying cut-points to the pointer registers that are used to calculate the addresses, as shown in Fig. 4. In so doing, we were able to separate the checking of the logic that calculated the pointers from the checking of the logic that generated the addresses. We were able to leave the cut-points unconstrained, since the address generation logic was completely independent from other design functions, such as the volume counters.

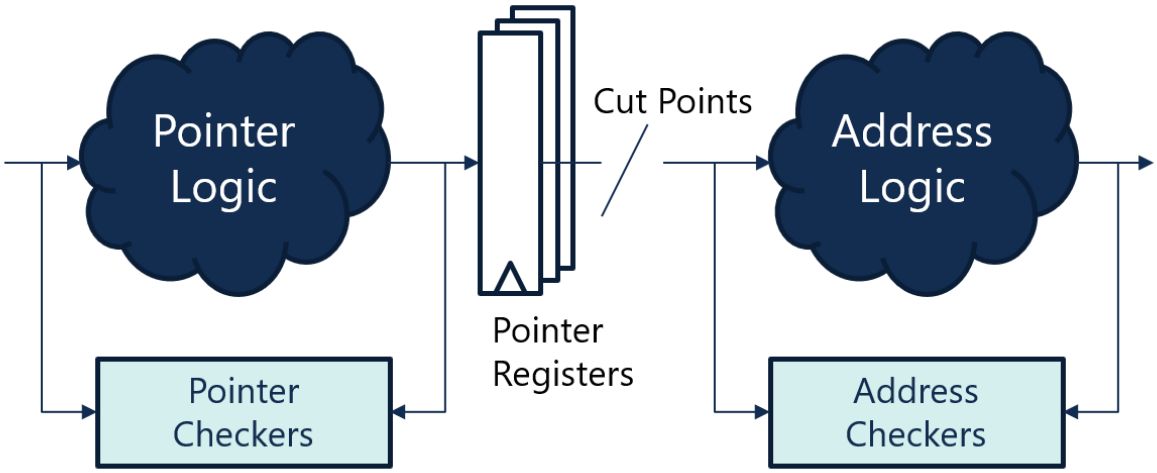

Figure 4. Address abstraction

Coverage

Formal coverage is a key component of the sign-off process, which helps to validate all three of the other primary elements of the methodology, as it can help identify:

1. Missing checkers
2. Over-constraints
3. Complexity barriers and insufficient bounded proof depths [6]

Controllability Coverage

There were two finite state machines (FSMs) in the design, one each in IDMA and ODMA, related to the compression and decompression functions. With the default setup for the IDMA FSM, we were unable to see a trace for the longest sequence of state transitions, as we were unable to achieve an analysis depth of beyond 59 cycles with a 6-hour run time. However, with abstraction models applied, we saw significant reductions in the number of cycles required to produce a trace for longest state transition as well as the runtime required. Table 1 illustrates the effect of abstraction models on the number of clock cycles required for the longest state transitions within each FSM.

FSM	Longest arc without abstraction models [clock cycles]	Longest arc with abstraction models [clock cycles]
IDMA FSM	63	17
ODMA FSM	77	19

Table 1. Effect of abstraction models on FSMs coverage

We saw a similar reduction in number of cycles for the depth of witness waveforms of the functional coverage points. We have described some of the representative examples in Table 2 below.

Scenario	Without abstraction models [clock cycles]	With abstraction models [clock cycles]
All IDMA counters updated twice	35	14
All ODMA counters updated twice	50	8
Register to track z-plane jump should be reset after being set	Not reached in 6 hr runtime (>63)	19

Compress data buffers are empty after getting full	Not reached in 6 hr runtime (>68)	7
Two compression metadata writes seen, one of which is due to buffers being full	Not reached in 6hr runtime (>103)	20

Table 2. Effect of abstraction models on functional coverage

We can observe that these functional cover points were especially hard for the formal tool to cover. We can thus conclude that the use of abstraction models was a key factor in improving the controllability coverage metrics from their default values to beyond the thresholds required for sign-off.

We also confirmed that the formal engines, with the aid of abstractions, had complete controllability of the design code coverage points. Table 3 contains the final reachability code coverage results after the application of abstraction models.

Code Coverage Type	Total Points	Reachable Points	Dead Code	Waived	% Covered
Line Coverage	2,261	2,238	16	7	100%
Conditional Coverage	1,055	984	27	44	100%

Table 3. Formal Reachability Code Coverage

Observability Coverage

Cone-of-Influence (COI) coverage helps to confirm the exhaustiveness of the list of checkers. It reports the flip-flops in the design which are not in the structural fan-in of any checker, thus indicating where checkers may need to be added. Table 4 contains the final report of out-of-COI coverage results for each formal DUT.

Formal DUT	Out of COI [flop bits]
DMAC	3
IDMA	8
ODMA	0

Table 4. Formal COI Coverage

We were able to waive each out-of-COI flop bit report, as they were not being used as part of the feature set for the current version of the DMA unit.

Constraints Validation

DMA constraints were run as assertions in simulation regressions at the CNN core level. A failure of these assertions is an indication of a bug in one of the following three areas:

1. A bug in the constraints
   The constraints are too tight. There is a bug in the constraints that disallows some input behavior in the formal run, but that input behavior is possible in the real world. This problem might mask real bugs from being discovered. The constraints need to be updated and the formal runs repeated.
2. A bug in a neighboring block
   A neighboring block is violating the interface protocol and driving illegal stimulus in simulation to the inputs of the block we have formally signed-off. The bug in the block that is driving the input to the formal DUT needs to be fixed.
3. A bug in the interface specification
   The interface protocol is mis-specified or is ambiguous. The constraints have been designed according to the spec, but the interface operates differently. The specification and the constraints need to be updated and the formal runs repeated.

With the DMA design, we discovered two failures during simulation regression testing:

- The first case was due to a bug in a neighboring block
- The second case was due to an outdated specification

IV. Results

Bugs Found

The formal verification work was done at a later stage of the project, after simulation was completed. Six corner-case bugs were found. Multiple preconditions had to occur with specific timing in order to reach the states which would trigger these bugs. This made the bugs very resistant to discovery through traditional verification methods.

For example, we found a deadlock bug in IDMA when data transfer happened in decompression mode. The bug is triggered when each of three IDMA FSMs reach specific states at the same time. For the deadlock to happen, the following preconditions must be in place concurrently:

1. Main memory is slow to respond to read requests from IDMA
    - One of the IDMA FSMs is in a wait state waiting for a read response
2. IDMA has run out of metadata (used to decompress the data stream coming from main memory)
    - A second IDMA FSM in the state where it is requesting new metadata
3. IDMA moves to the next step in the z-plane (3rd dimension) for the data volume transfer
    - A third IDMA FSM triggers another request for metadata

If all the above conditions occur in the same cycle, then:

→ IDMA spuriously sets a register, which renders it unable to accept metadata
→ To complete the data transfer to local memory, IDMA is waiting for decompressed data
→ Data in the buffer cannot be decompressed because there is no metadata available
→ Compressed data from the system memory fills up the buffer
→ IDMA cannot accept any more data from the system memory since its buffer is full
→ Deadlock!

V. Conclusions

The DMA controller is a type of design that has very long sequences of behavior, which makes it challenging to exhaustively test with formal tools. We have shown how block-level formal sign-off methodology, with the use of abstraction models, can explore all possible design behaviors and give us confidence that we have found the last bug.

References

[1] DesignWare EV6x Vision Processors, www.synopsys.com/ev6x

[2] B. Bailey, “Adding Order And Structure To Verification”, semiengineering.com/adding-order-and-structure-to-verification

[3] I. Tripathi, A. Saxena, A. Verma, P. Aggarwal, "The Process and Proof for Formal Sign-off: A Live Case Study.", DVCon 2016

[4] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In Proc. POPL ’86, pp. 184–193, 1986

[5] P. Aggarwal, D. Chu, V. Kadamby, V. Singhal, “End-to-End Formal using Abstractions to Maximize Coverage”, FMCAD 2011

[6] N. Kim, J. Park, H. Singh, and V. Singhal. “Sign-off with Bounded Formal Verification Proofs”, DVCon 2014