International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

Formal Verification of A Generic Hardware based Accelerator

Shrinivas B J
VLSI Design and Embedded Ssytem
BMS College Of Engineering
(Affiliated to VTU)
Bangalore, India

Dr. K. P. Lakshmi
VLSI Design and Embedded Ssytem
BMS College Of Engineering
(Affiliated to VTU)
Bangalore, India

***Abstract*— Formal verification being a State of the Art of Verification, gives any digital designer the confidence on a system designed by proving a set of properties which are derived by looking at the specification/behavior of the system. The advancement in the formal tools have catered to the state of art of formal verification. The ease of bringing up a design in formal verification environment results in early bugs being caught. These traits of formal give it an edge over the traditional verification. This project demonstrates how the power of formal verification and formal verification tools could be used to verify complex digital designs.**

Keywords—Formal Verification, Digital design, CNN

I. INTRODUCTION

As the semiconductor industry marches towards lower technology nodes for advancement in power, performance and area (PPA), the challenge for digital system design increases to meet the requirements. This challenge tends to increase the possibility of making mistakes by the designer, which leads to something called a bug. A bug is an error identified in a system. As any digital design has to go through fabrication process for practical use, finding bugs in a digital design at a later stage in the *Vlsi* design flow will lead to wastage of many resources. Therefore, the idea of using formal verification is explained and explored in the following sections.

Formal verification (FV) uses mathematical approach in order to check if a property adheres to the behaviour of system or the expectation of the system. The idea of formal verification has started to pick up its pace in the semiconductor industry as it could reveal the bugs in earlier stages of the development cycle as loading a design into formal verification takes very less time compared to traditional verification which usually takes weeks. This means it could save a lot of revenue.

A hardware accelerator, as the name suggests, aims at accelerating a certain activity. This is mainly used in GPU's. When a repetitive action has to be performed again and again, the load on the CPU would increase and the CPU will not be able to handle any other action. In order to avoid this, the CPU hands over the repetitive job to a GPU and fetches the result from GPU when it is available. Therefore, it can be seen as an off-loading task done by CPU for specialized computations in order to complete the task at the earliest.

Formal Verification (FV) is an approach where we exploit or look for convergence of properties of the design/specification. Thus, property development phase of a design is a very important/crucial aspect in Formal Verification. For incorporating FV, Formal Friendliness of RTL must be seen first [1]. This completely depends on the design. If the Design consists of high degree of concurrency/parallelism, implement control logic (e.g.: FSMs, arbiters, interrupt controllers, decoders, memory controllers, pipeline hazard detection units, etc.), then it is capable of end to end formal verification. If the Design is very complex as those including filters and signal processing units then end to end formal verification might be difficult and semi-formal verification could be used. Again, formal verification of complex designs are also possible but then it requires a lot of expertise and knowledge about the design. The authors of [1] have given a general methodology for performing formal verification in an iindustrial setup. The advantages of performing formal verification can be seen in [3]. In another work [3], a static formal verification method is used to verify the I/O multiplexing at the chip level using VC Formal Connectivity Checking (CC). In the early stage of the verification, the Input/Output reuse was verified for connectivity, and the toggle coverage was also collected. In [4] the authors give an Essential Toolkit for Modern VLSI Design and presents practical approaches for design and validation/verification, with hands-on advice to help working engineers integrate these techniques into their work. In [5] the author gives a brief overview of Simulation-Based Verification versus Formal Verification. According to the authors of [8], for aerospace FPGA software products, traditional simulation method faced severe challenges to verify product requirements under complicated scenarios. Given the increasing maturity of FV technology, this method could significantly improve verification work efficiency and product design quality, by expanding coverage on those "blind spots" in product design which were not easily identified previously. Taking UART communication as an example, the paper proposes several critical points to use formal for asynchronous communication protocol. Practices and experiments indicated that formal verification for asynchronous protocols can effectivelyreduce the time required, ensure a complete verification process and more importantly, achieve more accurate and intuitive results as it would cover few scenarios which are not possible to capture in simulation based verification. The contrast between the Formal and Functional verificationcan be understood by [9]. The authors of [10] have explored how formal verification could be used to reduce the area consumed by the design during the Synthesis process.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

With Machine Learning and Artificial Intelligence (ML and AI) taking over all the segments, the performance of any system now depends on the performance of the underlying ML processes. To accelerate the AI and ML processes, hardware accelerators are now being considered. Hardware acceleration is a process in which a CPU offloads a particular set of specialized repetitive computations to multiple programmable elements, such that the task is completed early[6].

In this work, we have considered verification of a Convolution Neural Network (CNN). End-2-End verification of a CNN includes Data Path Verification (DPV) and Formal Property Verification (FPV). As a first step, we have performed FPV and DPV would be the future work of this project. The following section gives the design overview of CNN.

II. DESIGN OVERVIEW

The functional block diagram of the Accelerator chosen is shown below.

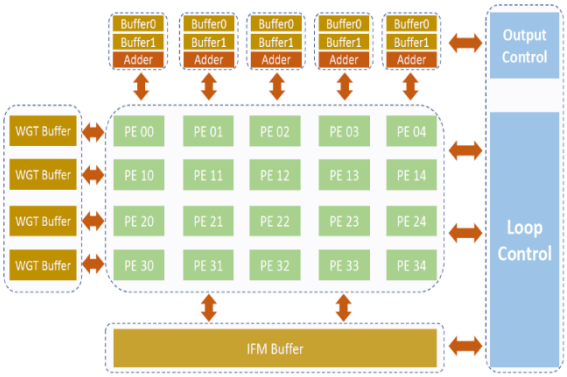

Fig 1. Design overview

As we can see in fig.1, there are multiple buffers. The buffers are used to store the input feature map (ifm) and weight values. These values are used by the programmable elements to compute the end result.

The *ifm* and *weight* values are divided into multiple sets and stored for the improvement in speed for calculations. To synchronize the activity between these buffers there are control logics designed using FSMs, counters and FIFOs. These are recognised as the potential matches in order to perform formal verification.

Fig 2. Interface overview

Fig.2. depicts the black box version of the top-level Accelerator design. At the input, interface signal *start_conv* is used to indicate the start of convolution. *Ifm* and *weight* are the inputs to the CNN. The signals *cfg_ci* and *cfg_co* indicate the number of channels in the CNN. As it's a 2-bit signal, the number of possible combinations are 4, where 2'b00 indicates 8 channels, 2'b01 indicates 16 channels and so on. Internally these channels configure the value of counters. Configuration of these counters will lead to the number of multiplications performed at module level. At the output, *ifm_read* and *wgt_read* indicates that the ifm and weight values are being fetched from the buffers. *Ofm_port0_v* and *ofm_port1_v* signifies if the output values on the ofm_port0 and ofm_port1 are valid or not. End_conv is asserted when the convolution is completed.

It was observed that, whenever any value is asserted on the output *ofm_port0* and *ofm_port1*, the signals *ofm_port0_v* and *ofm_port1_v* is asserted indicating that output is valid. Generally valid signals are asserted only after verifying the outputs are valid. With Data Path Verification (DPV), we can analyse if the design is asserting the valid signals only if the output is valid or just asserting the valid signals for any result computed by the module.

The detailed explanation of the design can be found in (https://github.com/lirui-shanghaitech/CNN-ccelerator-VLSI/tree/main/src).

The output ports ofm-port0 and ofm-port1 are the results of the convolution, which is result of data transformation blocks in the design (Data transformation block is explained in the next section).

III. FORMAL PLANNING

The Methodology adopted to perform Formal Verification ofhardware-based accelerator was taken from [1] and is as follows:

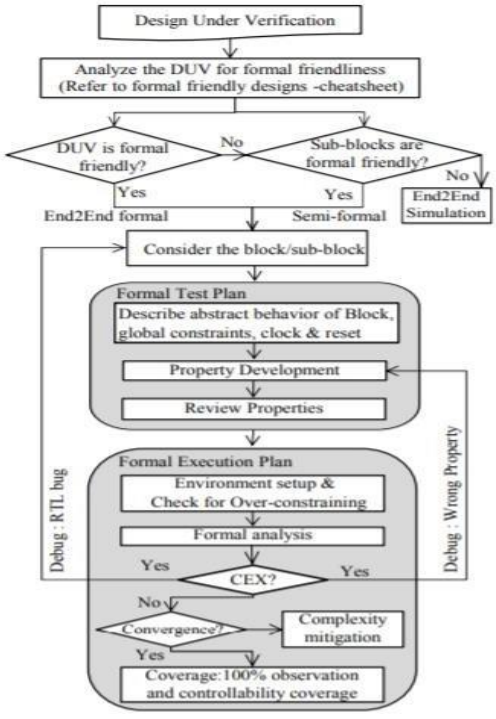

Fig 3. Formal test plan

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

While considering any block for formal property verification, it's important to have a high-level view of the block. The first step is to decide on the DUV. Then all the sub-blocks contained in it are observed from high level. The blocks are categorized into two types- Data transport block and Data transformation block. A block that allows the data to pass through it as it is, without any modification is referred to as data transport block. A well-known, common example of a data transport block would be a FIFO Buffer. A block that performs any operation on the input data before passing it to the output interface is referred to as data transformation block. All blocks categorized under data transformation block are considered as less formal friendly as it is not very straight forward to verify them.

In the following project all the sub-blocks were analyzed from high level and categorized as either belonging to data transport block or data transformation block. Four data transport or control modules were identified to which formal property verification [FPV] was applied.

Upon observation of the design at module level, there were four modules considered for formal verification PE_FSM, WRITE_BACK controller, PSUM_BUFFER and SYNCH_FIFO as they are data control block. Properties related to these modules were coded and implemented in Jasper Gold tool. As it is an open-source design, due to lack of proper documentation we have started with a few common assumptions in the formal environment.

There are two generic issues in the design we would like to point at after our formal analysis which would help the designers to get exposure and make use of the formal tool to ensure that the design meets the specification.

Scenario 1:

Scenario 1 depicting a bug in the code: Cover properties are properties that ensure a particular state/port in a design is asserted or not. During verification, a cover property is said to be hit if the combination of the signal/signals is possible to be asserted in the design. For example, let us consider a general FIFO buffer. In a FIFO buffer there are two output signals full and empty. In order to verify if the FIFO is able to assert the signals full and empty, we add a cover property in formal environment. If the cover property passes, it means the FIFO is able to assert the full and empty signal and the cover property related to full and empty is hit. The formal tool verifies if full and empty is asserted by continuously writing to FIFO and then by continuously reading from the FIFO. In general formal tries to prove a cover property by trying to assert all the signals in the design. Similarly in fig.4, the cover property is trying to see if there is any scenario possible in the design to see if the signal end_conv is asserted or not. If a cover property passes in a formal tool, it means the signal/state in the design is possible to be asserted. If a cover property fails in a formal tool, it means the signal/port is not asserted/hit by any means. This could be because of multiple reasons. The simple one being dead lock in an FSM or an incorrectly coded conditional statement. These can be very simple to implement.

Below (fig. 5) is a cover property that is written. The cover property intends to validate the behavior of port end_conv in fig.2. This property is basically asking the formal tool to check if the port end_conv is asserted at any positive edge of the clock by the design. Generally, these cover properties which are quite vague in nature flags an alert to the designer in early stages of the design if there are issues in design.

```
end_conv_must_be_asserted_after_convolution_is_completed : cover property(
@(posedge clk) disable iff (~rst_n)
        end_conv
);
```

Fig 4. Cover property

As we can observe in fig.4, there is a red cross mark next to the cover property. This basically means the cover property has failed.

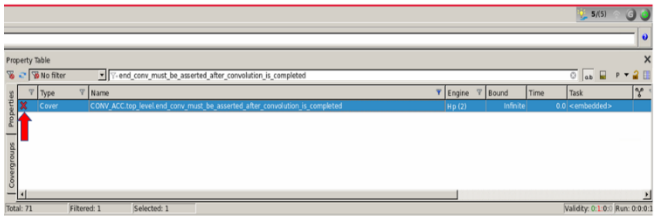

Fig 5. Cover property failure

Upon debug, it was found that because of loosely combined set of conditional statement the FSM was not able to reach/hit the finish state which was causing the design not to assert the end_conv port.

```
always @ (current_state or start_conv or start_again or cnt1 or cnt2)
begin
    next_state = 3'bx;
    case(current_state)
        IDLE:
            if(start_again)
                next_state = S1;
            else if(start_again && cnt2 == 0 && cnt3 == 0)
                next_state = FINISH;
            else
                next_state = IDLE;
        S1:
            next_state = (cnt1 == 4) ? S2 : S1;
        S2:
            if(cnt2 == 0 && cnt1 == 0)
                next_state = IDLE;
            else if(cnt1 == 0)
                next_state = S1;
            // else if(cnt1 == 0)
            //     next_state = S1;
            else
                next_state = S2;
        default:
            next_state = IDLE;
    endcase
end
```

Fig 6. Loosely coupled conditional block

As we can see in the above code, if start_again is asserted the next_state in the fsm is S1. If start_again is asserted, cnt2 is 0 and cnt3 is 0, then the next_state in FSM is FINNISH. As start_again is common in both the conditions, whenever start_again is asserted the fsm will always go to S1 and there is no possibility for the fsm to go to FINISH state. This is referred to as loosely combined conditional statement.

When the design is used as standalone module, this might not seem to be a big issue. In real world, a design always interacts with other modules. When the neighboring module is

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

waiting for the end_conv to be asserted in order to start processing the result obtained from this design, then there would be an overwrite of the result which would lead to the missing of previous result.

From a verification point of view, this would be a road block. According to the design, end_conv signal must be asserted after the completion of 1st convolution. This will be an indicator to assert the start_conv signal for the next convolution. Now as the end_conv signal is not being asserted; it would be tough to make the tool understand when the next set of inputs can be asserted.

Scenario 2:

Here is another scenario we came across in our debug.

A Scoreboard is an entity that verifies data integrity of a model. A model is said to be in alignment with data integrity if the data that entered the model first is sent out on the model first. A jasper scoreboard was implemented for the FIFOs in the design. As there were ten such FIFO buffers, a single test bench of FIFO was written to verify all the FIFO buffers.

It was observed that data integrity check was failing at a depth of 47. This basically means, at 47th cycle of the design, formal found a possible scenario where data integrity was not holding true.

It is quite common in formal that few properties expose other issues in a design. This is one such scenario. As we can see in the below figure, the output of the adder is pushed to FIFO.

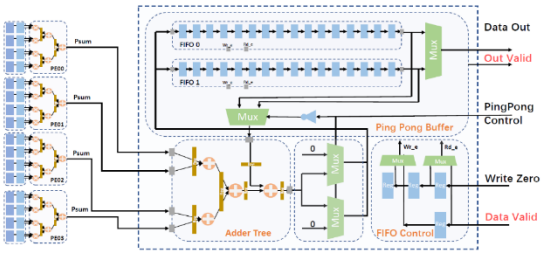

Fig 7. FIFO interface with neighboring module

Upon observing the waveform, it was found that the output of the adder was not pushed to the FIFO. Though this was not the reason for the failure of the property, it was observed during the debug. The reason for the failure of property was because though the read enable of the FIFO was enabled, the data at the output was not the available.

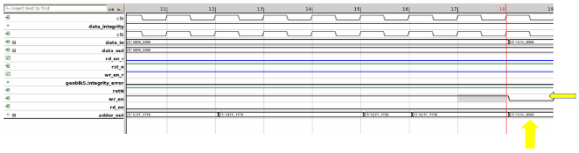

Fig 8. Adder output discarded

IV. RESULT

In this phase we have discussed about two generic scenarios that lead to a bug in the design. This will help the designers to think of different scenarios during the design phase of any module and explore formal verification domain in the early stages of the design that could help them resolve issues. An ideal design must consider all the possible scenarios and give the output accordingly even when an error is occurred. With increasing complexity in design, it is hard to think of all scenarios by a designer. Therefore, any designer with the aid of formal technology can verify the design by writing properties. Though it is impractical for one to expect the designer to perform verification, formal makes things much easier as the designer need not spend days in order to setup the formal environment.

A formal testbench is not reusable for different designs as the behavior of each design is implementation specific. Therefore, a formal testbench created for this design cannot be reused for any other design. Though the way of applying formal is the same, the exact properties of one design cannot be reused for other designs.

It's important to understand that in formal, not all failures of properties lead to a bug in the design. There is a great possibility of under constraints in the environment. The challenging task in formal is the iterative task in coding of assertions and achieving full coverage. Formal Property Verification (FPV) approach of formal technology was applied and properties were developed for the modules that control the operation of the entire CNN. There were totally 3 bugs found in the design. Within just a week of setup, 1 bug was found in the design. In 8-12 weeks, 2 bugs were found. Since the scenarios found were in general, we have confirmed it's a bug. There were totally 72 assertions written. As the FIFO was instantiated 8 times in the design, the assertion written for one FIFO was instantiated by the tool for all the FIFO's. Though there are more failures of properties, we believe it's because of the under constraint in the environment. The future work of the project would be to verify the convoluted data available at the output of the accelerator through Data Path Verification approach of formal technology which would complete the end-2-end verification of the design.

TABLE I. RESULTS

Sl no.	Result		
	Properties	***Passed***	***Cex***
1	Assertions	47	25
2	Assumptions	6	0

REFERENCES

[1] K. Devarajegowda, L. Servadei, Z. Han, M. Werner and W. Ecker, "Formal Verification Methodology in an Industrial Setup" 2019 22nd Euromicro Conference on Digital System Design (DSD), 2019.
[2] R. Wang, Y. Guan, X. Li and R. Zhang, "Formal Verification of CAN Bus in Cyber Physical System," 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2020.
[3] L. Duan, Y. Hu, H. Liu, W. Feng and J. Gan, "An Efficient Formal Verification Method in I/O Multiplexing Module Based on VC Formal CC," 2020 IEEE 3rd International Conference on Electronics and Communication Engineering (ICECE), 2020.
[4] Formal Verification, An Essential Toolkit for Modern VLSI Design. 1st Edition - July 24, 2015, Elsevier Publication.
[5] Hardware Design Verification: Simulation and Formal Method-Based Approaches, Published 2005 by Pearson.
[6] Designing Hardware Accelerators at high level of Programming Abstractions, Ravi Prashanth, July 2016.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

[7] Y. Hu and D. Li, "Formal Verification Technology for Asynchronous Communication Protocol," 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2019, pp. 482-486, doi: 10.1109/QRS-C.2019.00092.

[8] M. Girish, G. Gopakumar and D. S. Divya, "Formal and Simulation Verification: Comparing and Contrasting the two Verification Approaches," 2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS), 2021, pp. 41-44, doi: 10.1109/ACCESS51619.2021.9563305.

[9] R. Shah and K. Agrawal, "Formal Verification Aware Redundant Sequential Logic Optimization to Improve Design Utilization," 2021 22nd International Symposium on Quality Electronic Design (ISQED), 2021, pp. 233-237, doi: 10.1109/ISQED51717.2021.9424336.