A Novel Approach to Verify CNN Based Image Processing Unit

Sumit K. Kulshreshtha,
Intel Technology India Pvt Ltd,
sumit.kumar.kulshreshtha@intel.com,

Raghavendra J N,
Intel Technology India Pvt Ltd,
Raghavendra.Jn@intel.com

Abstract- Convolutional Neural Network (CNN) based image processors make use of highly configurable and compute intensive hardware components, which may not be configured by human but by the underlying machine learning engines. Due to very large configuration space and limited time to market, it is practically impossible to completely verify such designs using simulation-based verification approach. This paper defines a novel approach to verify CNN based image processor using Formal Verification (FV). Formal verification can effectively be used to verify such designs for all the possible configurations and for all the data values in much lesser time. This paper presents various techniques to reduce the overall verification time and their importance in verification of CNN based designs.

I. Introduction

In recent years, Convolutional Neural Networks, have revolutionized image processing and computer vision applications such as face detection and authentication, human gesture and posture estimation etc. Image processors using CNN are very complex due to large configuration space and complex data manipulation algorithms. It puts a challenge for verifying such designs. In simulation-based verification, exercising directed or constrained random tests is not feasible due to very large configuration space ($\sim 2^{Nx1000}$; $N > 0$) of CNN based designs. Using random tests is also not a good approach due to limited time to market and random tests don't guarantee for complete coverage in limited time. Hence due to very high degree of configurability and limited time to market, it is impossible to completely verify CNN based designs using simulation-based verification approach.

This paper describes, how Formal Property Verification (FPV) can be leveraged to verify a complex image processor. Generally, control path and data path are verified separately using formal property verification and data path verification methodologies respectively. In our design, the data path (Algorithms) was so deep embedded into the control path that it didn't make sense to separately verify them. Hence, we used only FPV to verify both control path and data path together. This paper explains development and usage of Assertion IPs (AIPs), which can be used in an image processor and typical interfaces. This paper also presents a proven method to model image processing data path algorithms which can efficiently be used in FPV.

II. Techniques to verify CNN based image processor

The techniques explained in this paper are listed as below:

1. Development of frame assertion IP.
2. Techniques for faster FV testbench bring up.
3. Techniques to make sure last data is sent out and nothing is stuck within the DUT.
4. The usage of assertion IP for dead-lock/live-lock detection in CNN designs.
5. Techniques to write FPV friendly algorithm models.
6. How a combination of overlapping local checkers can guarantee bug free CNN design.

1. Development of frame assertion IP

For processing in image processor, an image is divided into multiple frames and each frame is further divided into multiple pixels as shown in Figure 1. The image processor is a combination of control and data path components which work at the boundary of image frames. This paper suggests an assertion IP(AIP), developed for image frame attributes, associated with each pixel. This frame AIP helps to drive in correct frames attributes and provides checkers at the output side to make sure correct frame attributes are driven out of the Design Under Test (DUT).

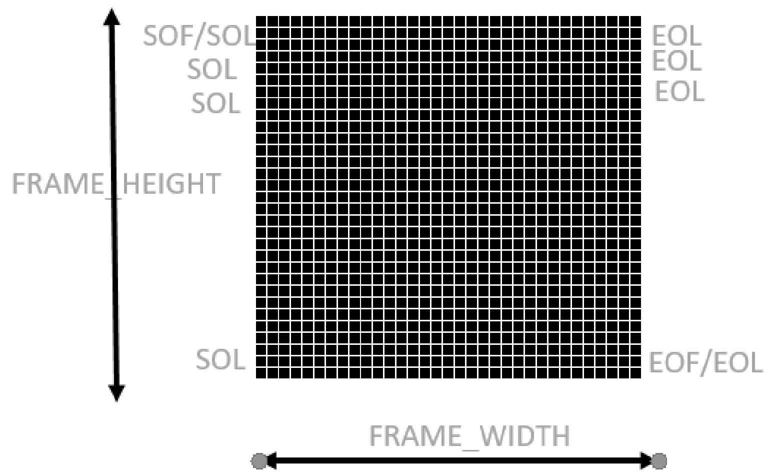

Figure 1. Image frame structure with attributes.

Each frame has below attributes,

SOF – Start of Frame
EOF – End of Frame
SOL – Start of Line
EOL – End of Line

The very first pixel of a frame must have SOF and the last pixel must have EOF set. First pixel of each line must have SOL and last pixel of each line must have EOL set. The dimension of the frame is expressed as FRAME_HEIGHT x FRAME_WIDTH. Where, FRAME_HEIGHT represents number of lines and FRAME_WIDTH represents number of pixels in a line.

The frame AIP has three set of assertions:

i. Assertions to make sure each pixel has the right attributes (SOL, EOL, SOF, EOF) as per its location within the frame. The location of a pixel can be calculated using two counters (x_cnt and y_cnt), FRAME_HEIGHT and FRAME_WIDTH. One counter (x_cnt) tracks the location of a pixel in a line. Second counter (y_cnt) tracks the line number in the frame. With the help of x_cnt, y_cnt, FRAME_HEIGHT and FRAME_WIDTH, the properties (P1 – P8) for attributes are written as shown below:

P1: sol_for_new_line

valid_pixel && (x_cnt == 0) |-> SOL

P2: no_sol_for_no_new_line

valid_pixel && (x_cnt != 0) |-> ~SOL

P3: eol_for_line_end

valid_pixel && (x_cnt == FRAME_WIDTH-1) |-> EOL

P4: no_eol_for_no_line_end

valid_pixel && (x_cnt != FRAME_WIDTH-1) |-> ~EOL

P5: sof_for_frame_start

valid_pixel && (x_cnt == 0) && (y_cnt == 0) |-> SOF

P6: no_sof_for_no_frame_start

valid_pixel && ((x_cnt != 0) || (y_cnt != 0)) |-> ~SOF

P7: eof_for_frame_end

valid_pixel && (x_cnt == FRAME_WIDTH-1) && (y_cnt == FRAME_HEIGHT-1) |-> EOF

P8: no_eof_for_no_frame_end

valid_pixel && ((x_cnt != FRAME_WIDTH-1) || (y_cnt != FRAME_HEIGHT-1)) |-> ~EOF

First set of assertions in this AIP is used at all the frame interfaces which include both primary and intermediate interfaces.

ii. Assertion to make sure once a frame is started (SOF), it must get completed (EOF).

valid_pixel && SOF |=> s_eventually EOF

iii. Assertion to make sure if there is no frame, eventually a frame appears.

~Frame |=> s_eventually Frame
Or
~valid_pixel |=> s_eventually valid_pixel

The effort to create this AIP is one day but it saves a lot of time as this AIP is instantiated at multiple interfaces.

2. Techniques for faster FV testbench bring up

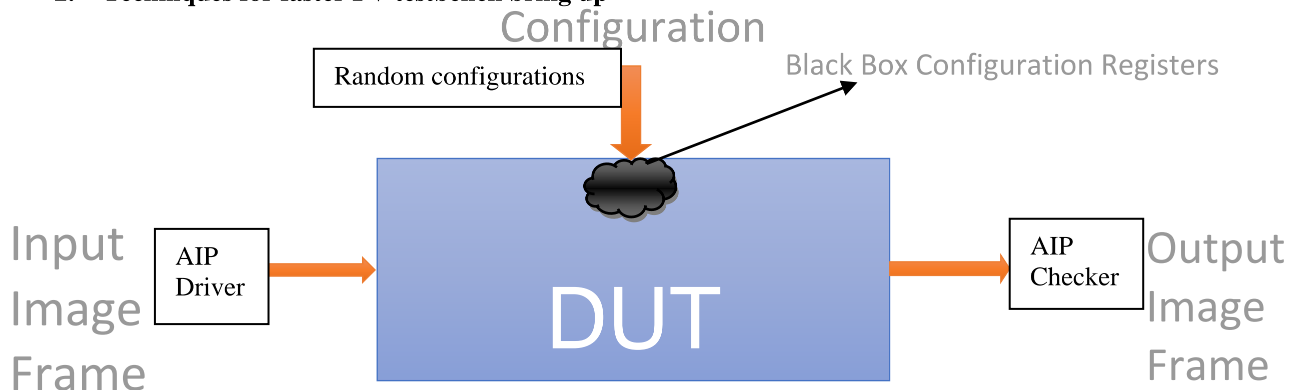

Figure 2. FV Testbench.

A typical FV testbench is shown in Figure 2. It has mainly three interfaces, input image frame, output image frame and configuration bus to configure the DUT registers. Various methods to make the FV testbench bring-up faster are explained below:

A. The configuration register set is black boxed which provide two different benefits.

i. Reduction of design complexity - Reduces the design complexity in terms of number of flops and hence reduces the state space.

ii. Zero cycle configuration - The registers are configured using an AMBA bus protocol and it takes many cycles to configure these registers. As these registers are black boxed, they are configured in zero cycle which saves many cycles of configuration.

B. As both input and output of the DUT are image frames, the frame AIP can be used at primary input and primary output interface. At the input side, the AIP properties act as assumptions to make sure correct frame is driven

into the DUT. At the output side, the AIP properties act as assertions and check if the DUT output frame is having pixels with correct attributes.

C. Maximize the usage of assertion IPs wherever possible. Depending upon the design implementation, use any other generic/custom AIP such as AIP's for valid-ready interface, credit-based interface, fifo, counter etc. Even the frame AIP can also be used for intermediate sub interfaces withing the DUT.

3. Techniques to make sure last data is sent out and nothing is stuck within the DUT

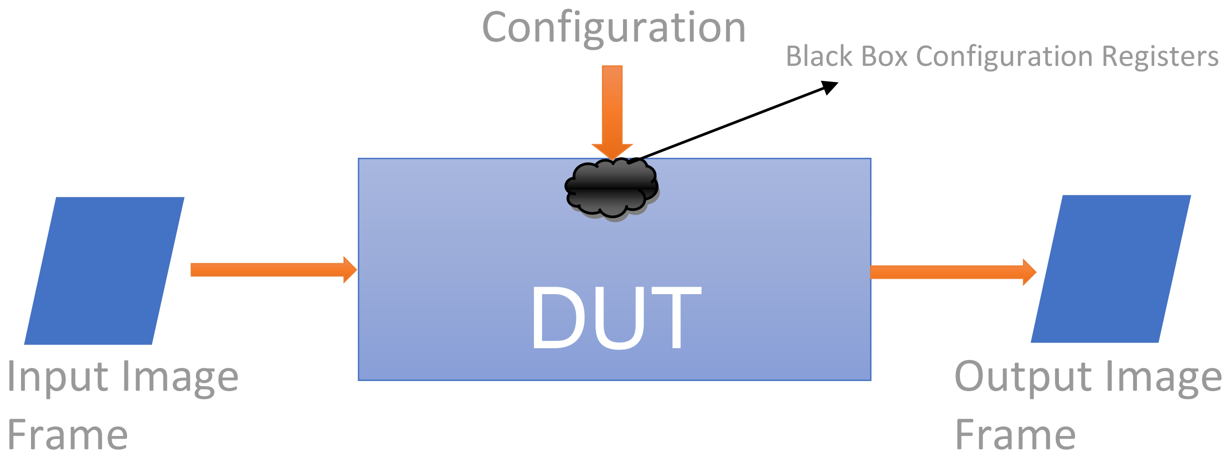

Figure 3. DUT with only one input frame and one output frame

In a typical image processor, pixels are selected or dropped from the input image, based on the configured horizontal and vertical strides. A selected pixel is sent out of the DUT (pixel push mechanism) based on certain predefined events. Example of such events may include,

a) A selected pixel is kept in an internal buffer. A newly selected pixel pushes the previously selected pixel out of the DUT.
b) When end of frame (EOF) is reached for input frame, the buffered pixel is pushed out of the DUT.

If not properly implemented, the pixel push mechanism may introduce a bug and such bugs are difficult to expose as they are mainly triggered by corner cases. Few of the examples for such bugs are

- DUT does not give out EOF,
- no more pixels come out of the DUT after a certain output pixel count,

an event does not push output pixel

There are very high chances of not having stimuli for such corner cases in simulation-based verification and the bug gets escaped. Although FPV can catch all such bugs due to its exhaustive nature, it can be time consuming to accomplish the task. The method we propose here to build a formal testbench can accelerate the process. When the frame AIP is instantiated at the DUT output interface, the second set of assertions in the frame AIP can catch all such bugs. The assertion looks like as below

SOF |=> s_eventually EOF

According to this assertion, if DUT starts giving out a frame, it must give out the whole frame. If there is a bug and a pixel is stuck in the internal buffer, this assertion will fail. To catch all such bugs, the FV testbench drives in only single frame as shown in the Figure 3 and expects only one frame output. If one full frame is received at the DUT output, it means there is no basic bug in the pixel push mechanism. Sometimes after processing one complete frame, there may be some sticky bugs, which can only be exposed if we drive more than one frame. The strategy to expose these bugs is to drive in a fixed known number of input frames. We recommend driving 3 frames and expect three frames at the output side. This strategy makes sure that there is no bug in the pixel push mechanism and all the expected pixels come out of the DUT.

4. The usage of assertion IP for dead-lock/live-lock detection in CNN designs

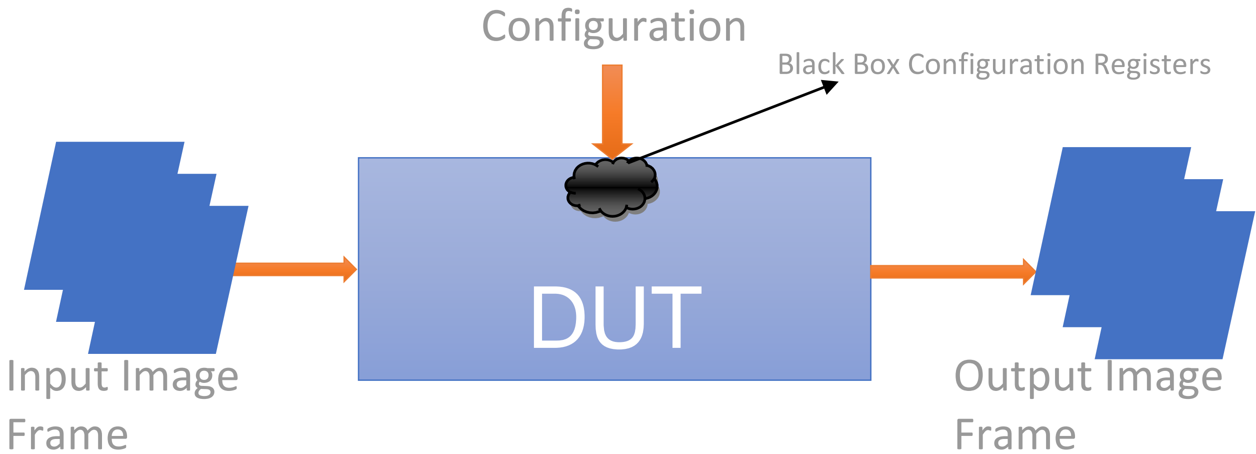

Figure 4. DUT with infinite input frames

The deadliest deadlocks or live locks, causing system hang triggered by various conditions, might be difficult or impossible to exercise in simulation-based verification. Liveness properties are best to detect such dead/live locks. The third set of assertions in the frame AIP is best used in order to detect the hangs due to these locks.

~Frame |=> s_eventually Frame

In order to use detect the hangs, use this property at the input interface as assumption to make sure back to back (with zero cycle or more delay) frames pixels are driven into the DUT as shown in Figure 4. While at the output side, this property is used as assertion which expects back to back (with zero cycle or more delay) output pixels from the DUT. The idea is to keep on driving input pixels and with this constraint, DUT is expected to keep giving out the processed pixels. If there is a hang, this assertion fails. This way using only this assertion, it can be made sure that there is no deadlock or live lock in the design.

5. Technique to write FPV friendly algorithm models

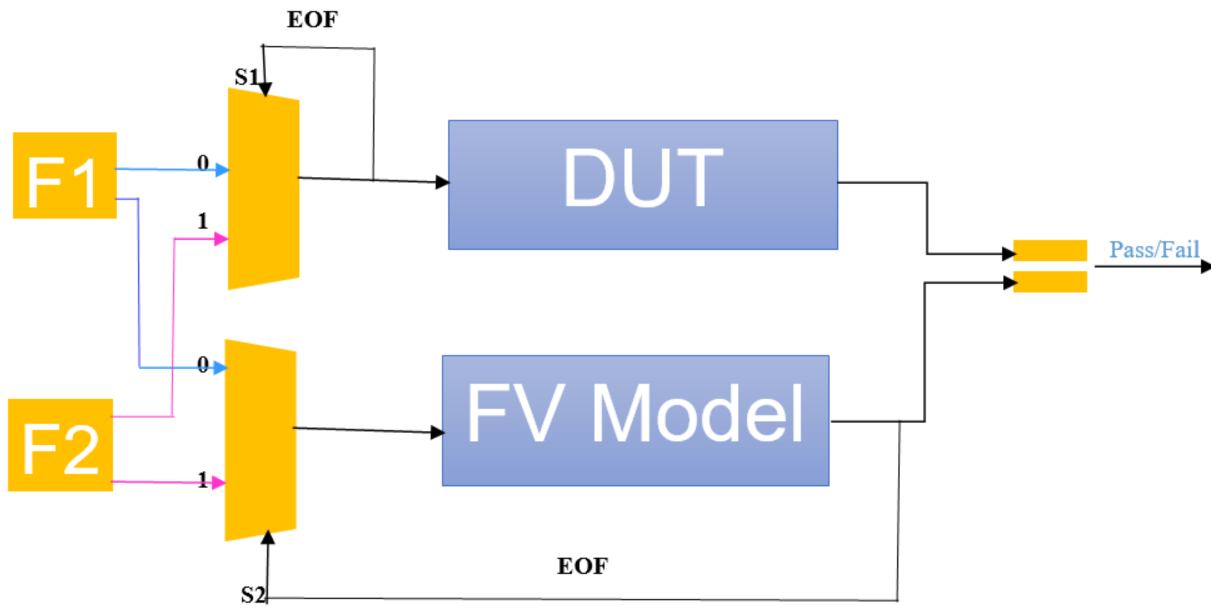

Figure 5. FPV friendly method of data path algorithm

In CNN based image processor, it is very difficult to separate the control path and data path. The data processing algorithms receive one pixel at a time, and it may require few past and future pixels as well in order to process current pixel and produce an output. RTL uses buffers to store pixels and thus processing each pixel takes more than one cycle. If we drive in any random pixel every time, the FV algorithm model would also require similar buffers as RTL and it causes unnecessary complexity.
Another better and easy to implement solution is shown in Figure 5. Which uses two different frames F1 and F2. F1 and F2 can change their value if and only if they are not being used by DUT and FV model. If F1 is being used by either DUT or by FV model, F1 must remain constant. Similarly, If F2 is being used by either DUT or by FV model, F2 must remain constant. The mux selection line S1 and S2 decide which one is the working model. And a working model must remain stable. Whenever the DUT receives EOF, S1 is toggled and after the DUT gives out EOF, S2 is toggled.
Above two conditions can be modeled as below two assumptions

assume (S1 == 0) || (S2 == 0) |-> $stable(F1)

assume (S1 == 1) || (S2 == 1) |-> $stable(F2)

After the DUT has received a full frame (EOF pixel) and it is ready to receive a new frame, the select line S1 is toggled. The DUT may take many more cycles until giving out the previous full frame, FV model must be using the previous frame and it means S2 must remain constant. When the DUT gives out EOF pixel (end of frame), the input to the FV model is toggled (S2 toggling).

With the help of above explained frame switch mechanism, a frame remains constant and all the pixels are known in zero cycle just before it starts acting as a working set for the FV model. As all the pixels are known, the FV model is implemented as a zero cycle or combinational model for the actual specification.
If only a single frame is used, it will impose two possible limitations of non-stress testing. In single frame case,

i. If the frame is always constant, stuck at faults may be undetected.
ii. If frame is changed only if DUT has completed the processing and given out the full frame. It will create a sequential frame processing scenario and stress testing will never happen. While in real scenario, a new frame may start coming into the DUT before previous one is processed.

With the help of above explained method, it becomes much easier to verify a DUT where control and data paths cannot be separated.

6. How a combination of overlapping local checkers can guarantee bug free CNN design

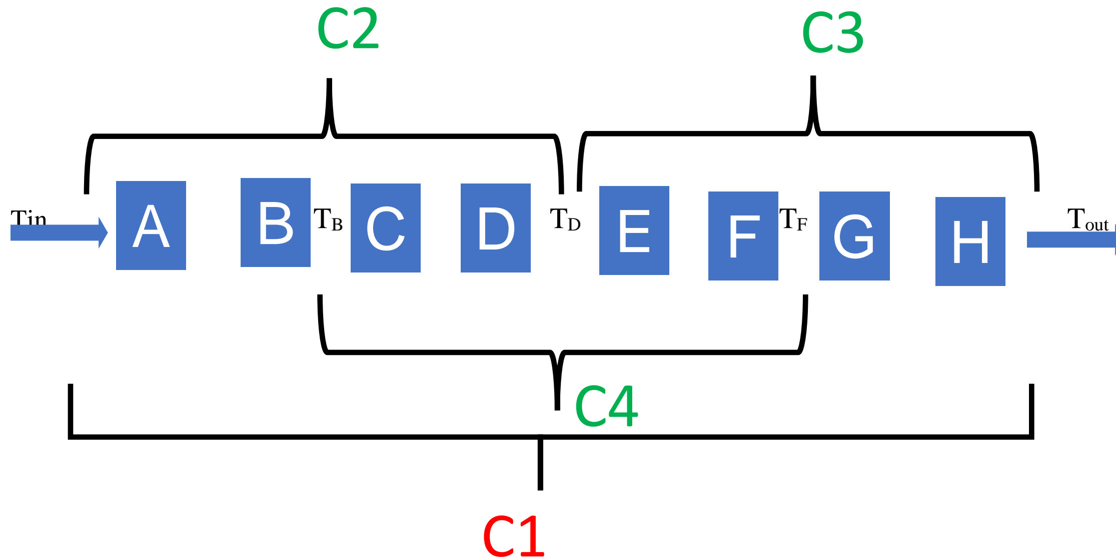

Figure 6. Overlapping checkers and end to end checkers

When it comes to the verification quality, end to end checkers provide more confidence than local checkers. But with CNN based designs, the end to end checkers are not good for two different reasons:

1) Due to multiple data path algorithms and control path from input to output, it is very difficult to model end to end checkers.
2) The end to end checkers for CNN based designs don't get converged due to the complexity of design and checkers.

In this section, a new approach is explained which does not require end to end checkers. We propose to write overlapping local checkers spread throughout the design as shown in Figure 6. In Figure 6, the end to end checker C1, covers the whole design from input to output. But in CNN based image processor, C1 does not even reach to a good bound and does not add any value. Instead in our approach, we suggest breaking the design in to multiple logical partitions and then write local checkers in such a way that each checker covers the whole logical partition. This way end to end checkers are written for each partition and provide confidence in the respective partition. Now the design is again partitioned in such a way that previously partitioned interfaces lie within the new logical partition. The DUT converts T_{in} into T_B followed by T_B into T_D followed by T_D into T_F and finally T_F into $T_{out.}$

This whole process is explained in Figure 6. The checker C1 covers the whole design but does not reach to enough bounds. C1 checks if T_{in} is correctly transformed into T_{out} in t_1 cycles.

C1: Valid_output at H |-> T_{out} == \$past(Transform(T_{in}),t_1)

The DUT is portioned as A, B, C, D, E, F, G, H. The checker C2 covers A, B, C, D and checks if T_{in} is correctly transformed into T_D in t_2 cycles.

C2: Valid_output at D |-> T_D == \$past(Transform1(T_{in}) ,t_2)

The checker C3 covers E, F, G, H and checks if T_D is correctly transformed into T_{out} in t_3 cycles.

C3: Valid_output at H |-> T_{out} == \$past(Transform2(T_D) ,t_3)

Due to smaller cone of influence (COI) both checkers C2 and C3 converge. Then we write another checker C4, which covers C, D, E, F and checks if T_B is correctly transformed into T_F in t_4 cycles. C4 cover the DUT in such a way that it covers the interface(D-F) of previous partition. Convergence of C2, C3 and C4 adds more value than lower bounds of C1.

C4: Valid_output at F |-> T_F == \$past(Transform3(T_B) ,t_4)

III. CASE STUDY

The techniques explained in section II were used with a real design as explained below.

DUT specification:

- **DUT:** CNN based Image Processing Unit
- **Configuration:** 2^{2000} possible configurations.
- **Input:** Up to N input frames; N = [1:12]
- **Output:** Up to M output frames; M = [1:12]
- **Frame Size:** 4k X 2K pixels
- **Number of Data Algorithms:** 12 (Algo 1 - Algo 12 in the Figure 7)

Abstractions used for FPV:

- Reduced Frame size: 16x16

- Black box: Configurations Registers

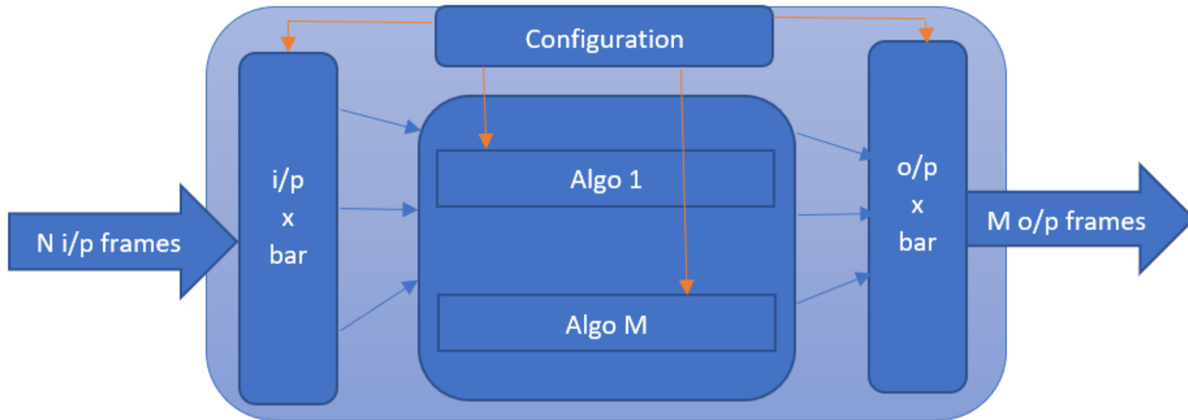

Figure 7. CNN based image processor

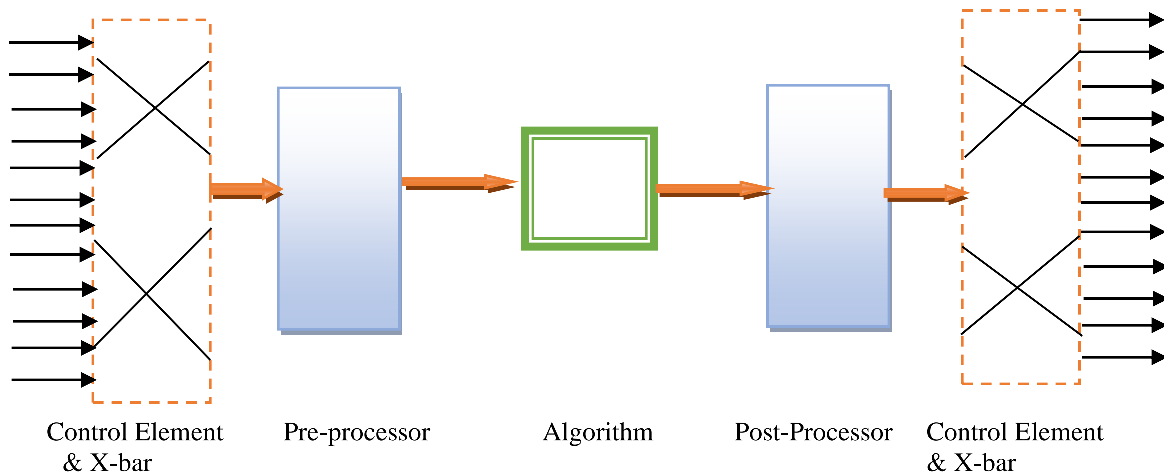

Figure 8. DUT with a configured preprocessor, algorithm and post processor

The case study DUT looks like as shown in Figure 7. The control path contains various configuration setting and based on these configurations, DUT decides the pre-processing algorithm and the main data algorithm for each frame, frame mixing, frame routing etc. Up to 12 input and 12 output frames can be processed in parallel. It supports up to 12 algorithms. When DUT in Figure 7 is configured for a particular use case, it looks like as shown in Figure 8. Which shows the DUT, configured for a given algorithm and a fixed control path.
The challenges in verifying this DUT were

i. Time – The time to verify this design was 20 weeks.
ii. Coverage – The design was explored for all the possible configurations with 100% coverage due to exhaustive nature of FPV. The FPV testbench constraints were written in such a way that they allowed all the possible configurations.

Details of bugs found
More than 45 design bugs were found and finally a bug free design is delivered. The bounds for failing assertions were from 2 to 812. Where the failure with bound 2 was due to the routing in the initial pre-processing. The fail with bound 812 was in pixel push mechanism, where pixel was dropped after one full frame was processed. Few bugs

found due to deadlock. Live locks were detected when multiple frames were interacting to produce final output frame. Data path algorithm related bugs were detected by the technique as explained in section II.

CONCLUSION

Verification of CNN based image processor is very complex due to large configuration space. It is not feasible to cover all the configurations using simulation-based verification. In such scenario, FPV has been proven very much useful. Formal Property Verification (FPV) techniques as explained in this paper are leveraged effectively to verify the complex CNN based image processor design where data path is deeply embedded into the control path which comes with infinite configurations. The techniques described in the paper helped us in two different aspects of verification as

1. Reduce the overall verification time and resource by more than 50%
2. 100% Coverage with guaranteed design quality

The further plan is to leverage this technique to verify more advanced CNN based design across client and server based SOC's.

REFERENCES

[1] Panagiotis Kouvaros, Alessio Lomuscio, "Formal Verification of CNN-based Perception Systems," ArXiv vol. abs/1811.11373, 2018.
[2] Hoang Dung Tran, Stanley Bak, Weiming Xiang, Taylor T. Johnson, "Verification of Deep Convolutional Neural Networks Using ImageStars," Springer, vol. 12224 LNCS, pp. 18-42, CAV2020.