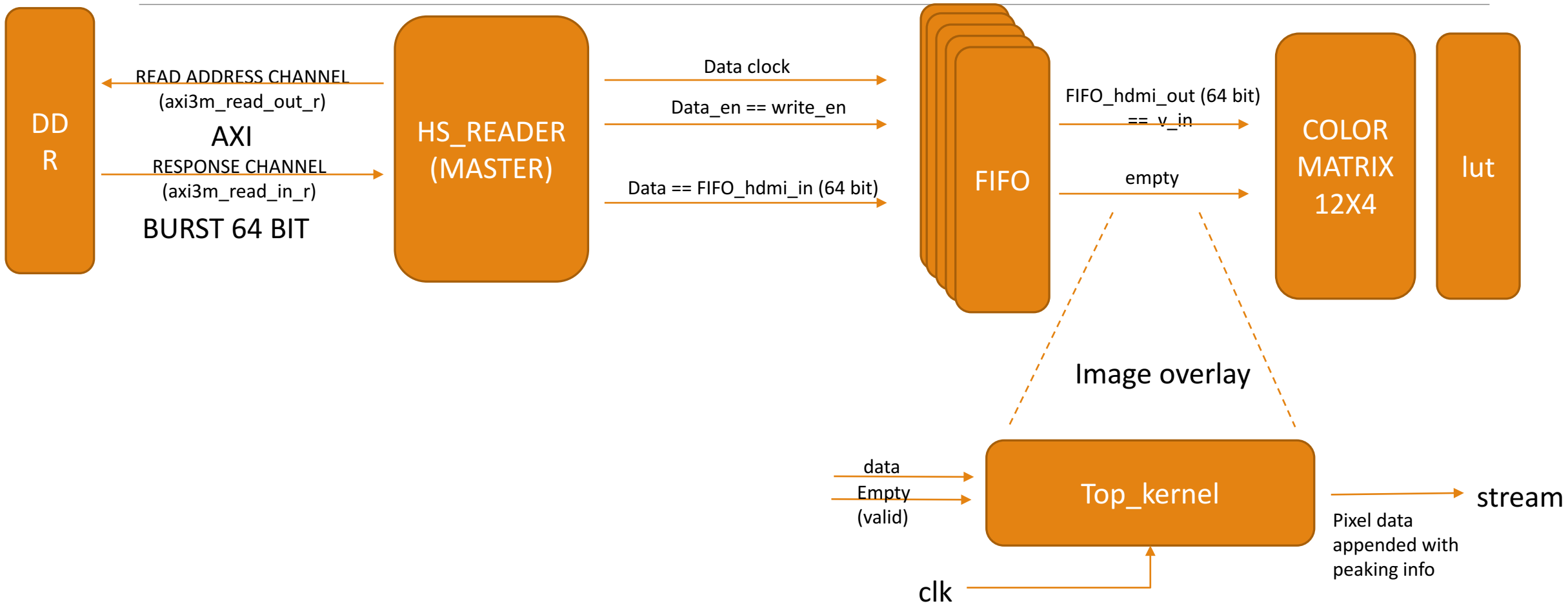
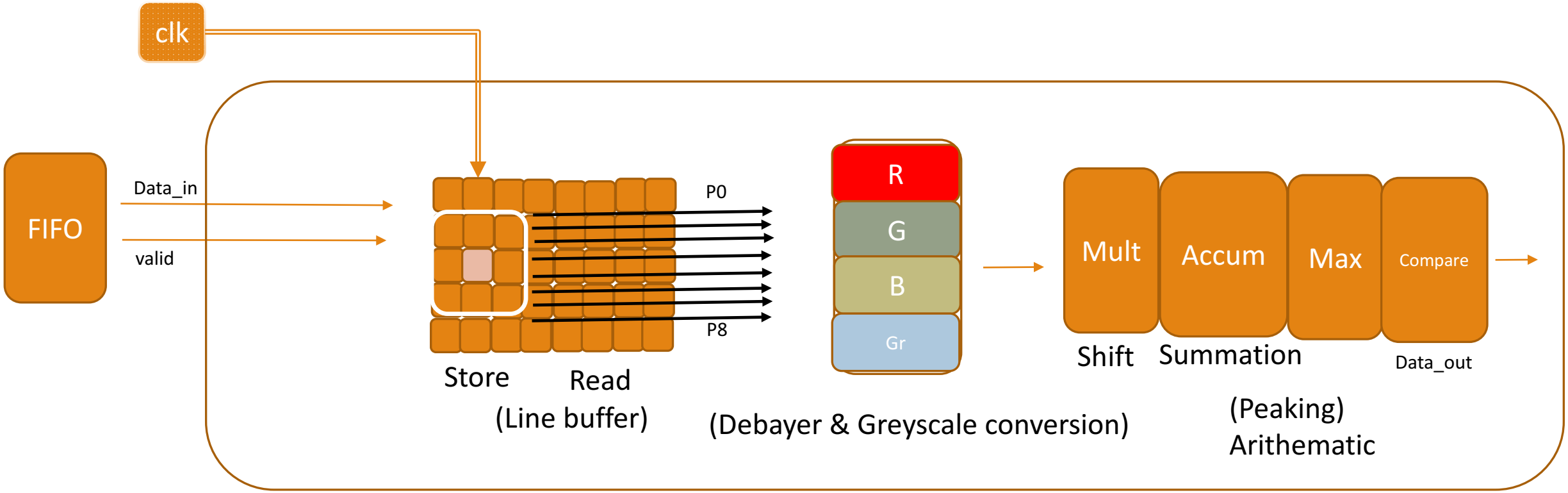
Real-time Focus Peaking

VHDL BASED KERNEL

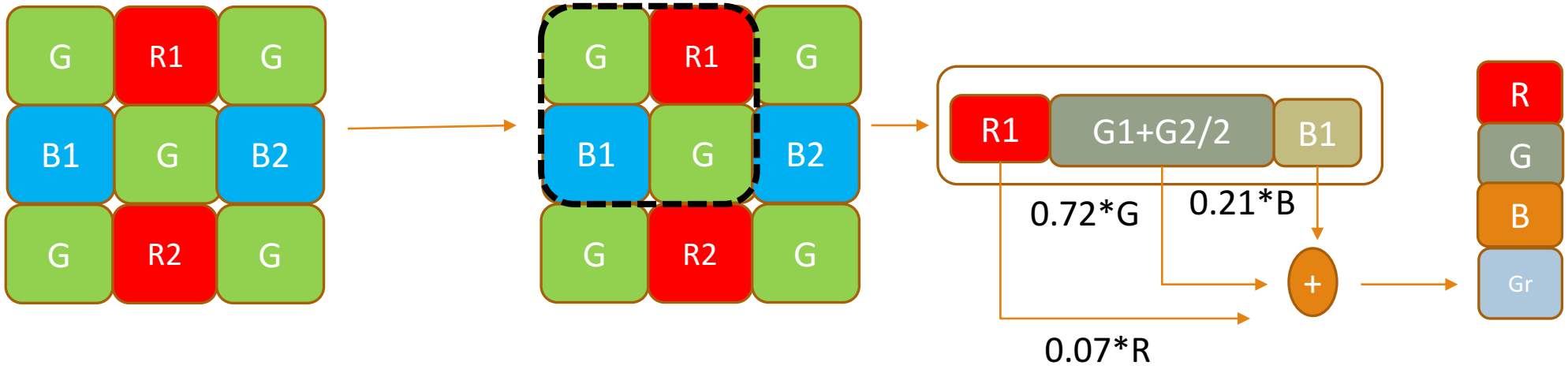
Output Image Processing Pipeline



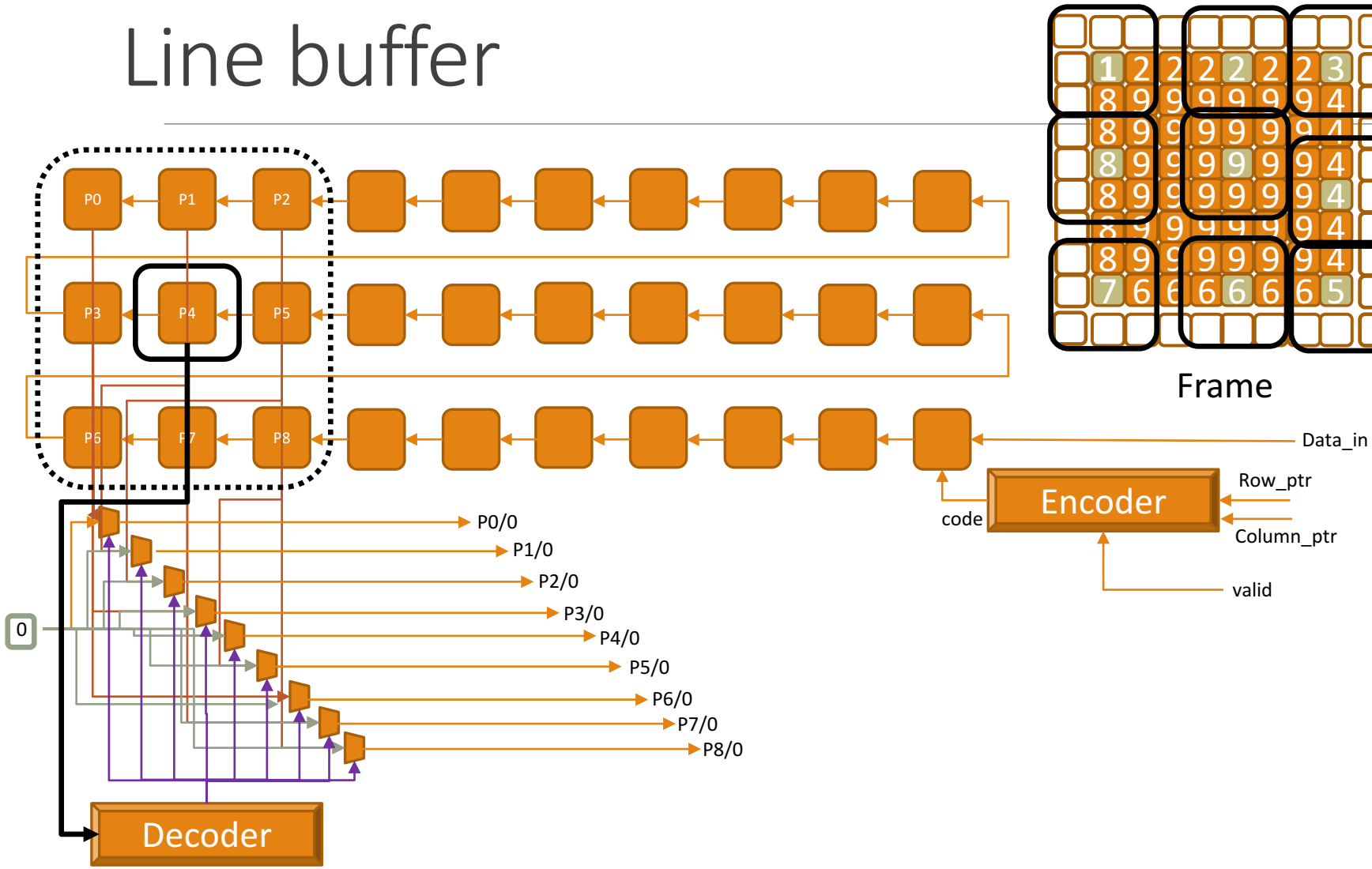
Top Module


clk
FIFO
Data_in
valid
P0
P8
Store
Read
(Line buffer)
R
G
B
Gr
(Debayer & Greyscale conversion)
Mult
Accum
Max
Compare
Shift
Summation
Data_out
(Peaking)
Arithematic

GRB Gr Module



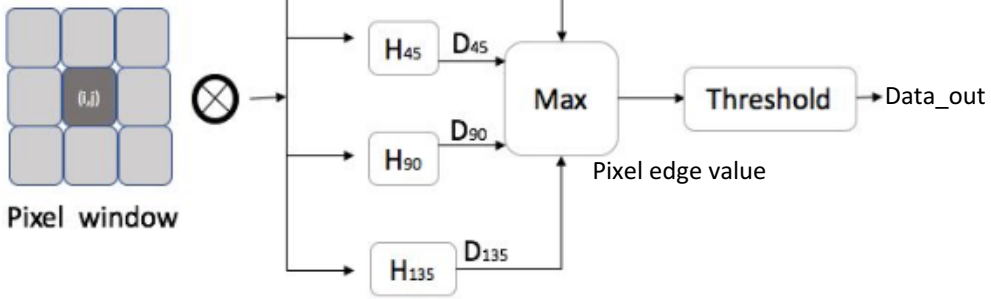
Line buffer

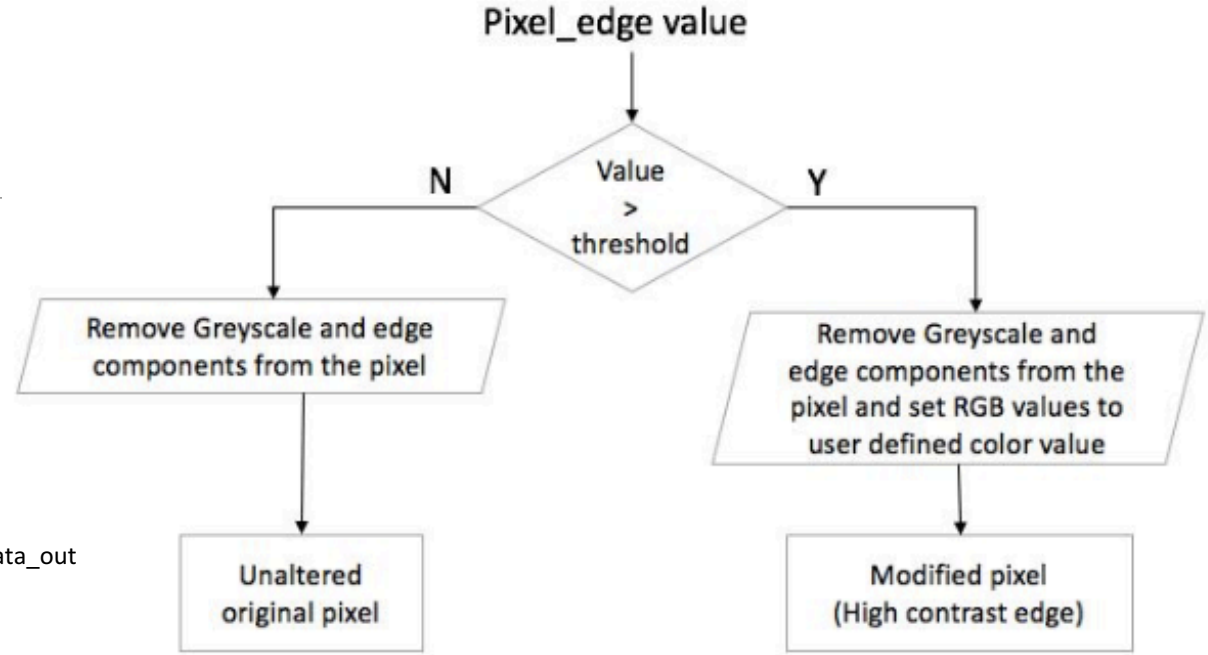


Case	Code	Condition
C1	1000	L_ptr=0 &c_ptr=0
C2	1001	L_ptr=0 &c_ptr/=0 &c_ptr /= (columns-1)
C3	1010	L_ptr=0 & c_ptr /= (columns-1)
C4	1011	L_ptr/=0 & L_ptr/=0(Rows-1) & c_ptr /= (columns-1)
C5	1100	L_ptr/=0(Rows-1) & c_ptr /= (columns-1)
C6	1101	L_ptr/=0(Rows-1) & c_ptr/=0 & c_ptr /= (columns-1)
C7	1110	L_ptr/=0(Rows-1) & c_ptr/=0
C8	1111	L_ptr/=0 & L_ptr/=0(Rows-1) & c_ptr/=0
C9	0001	Normal case
	0000	NO ACTION

L_ptr = Row pointer
C_ptr = Column pointer

Sobel kernel



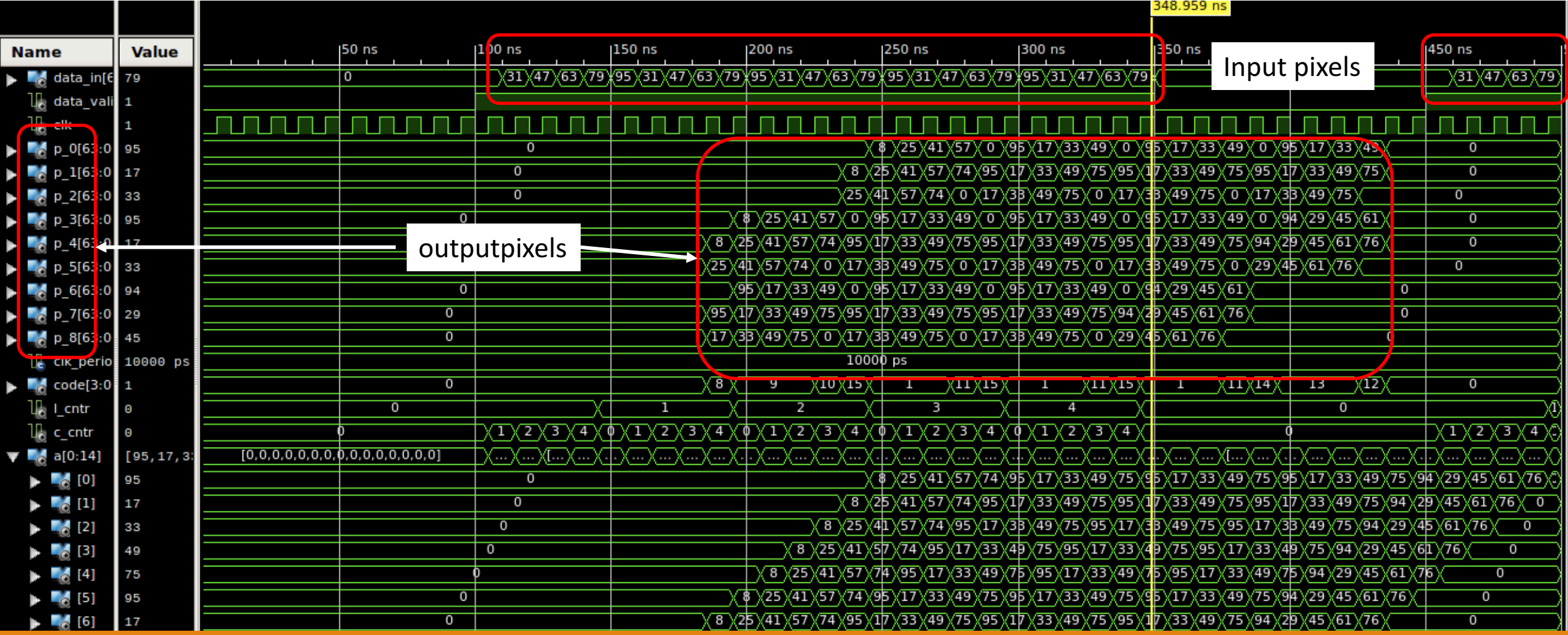


$$H_0 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad H_{45} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad H_{90} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_{135} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

Multiplying by:-
-1 = 2s compliment
-2 = left shift and 2s compliment
2 = left shift

Simulation

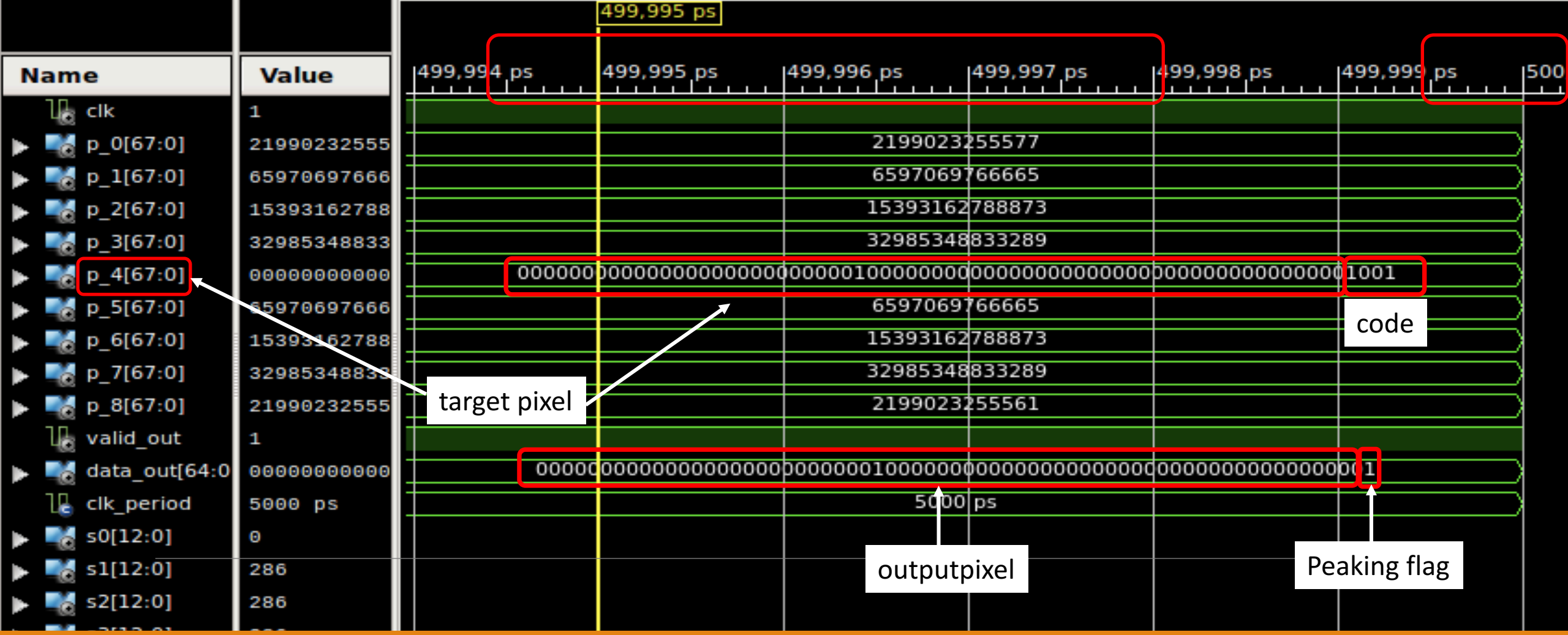
- To Simulate the IP the main structure was the line buffer.
- To test the line buffer, external signals in form of random pixel values were forced on the line buffer (5x5 for testing purpose) over a period of time so that all the pixels streams through the pipeline.
- The result of the pipeline was observed and the generated window pixels were compared so as to verify the correct functioning of the pipeline.
- The top module was tested to see if the outstream is generated correctly when predicted, throughout the output stream valid_out remains high and low otherwise and the peaking_flag is raised on the correct positions where high contrast change is observed.



Simulation result for line buffer

Simulation results for Line buffer

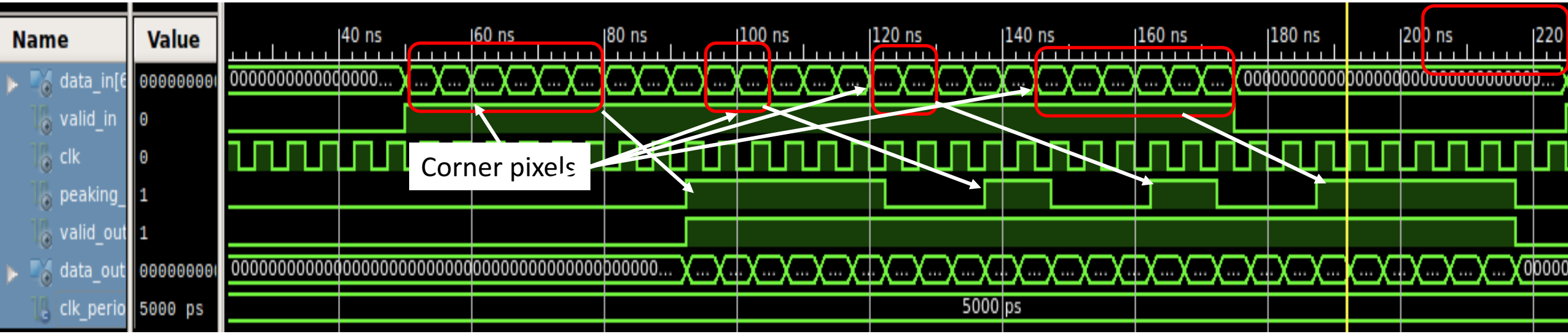
From the simulation results of the line buffer module, we can conclude that based on the line and column counters (l_cntr and c_cntr resp.) the ‘code’ vector is perfectly generated. The data Is stored in the buffer represented as vector ‘a’ and the output is produced as expected based on the extracted code.



Simulation result for peaking kernel

Simulation results for Sobel kernel

From the simulation results of the sobel kernel, we can see that the code extracts the target pixel information, generates summation values (as ‘s0/1/2’ - 286) which is greater than 200 (which is set as threshold). The peaking flag is set high in the output pixel. This proves that this kernel is functioning correctly.



Simulation result for line buffer

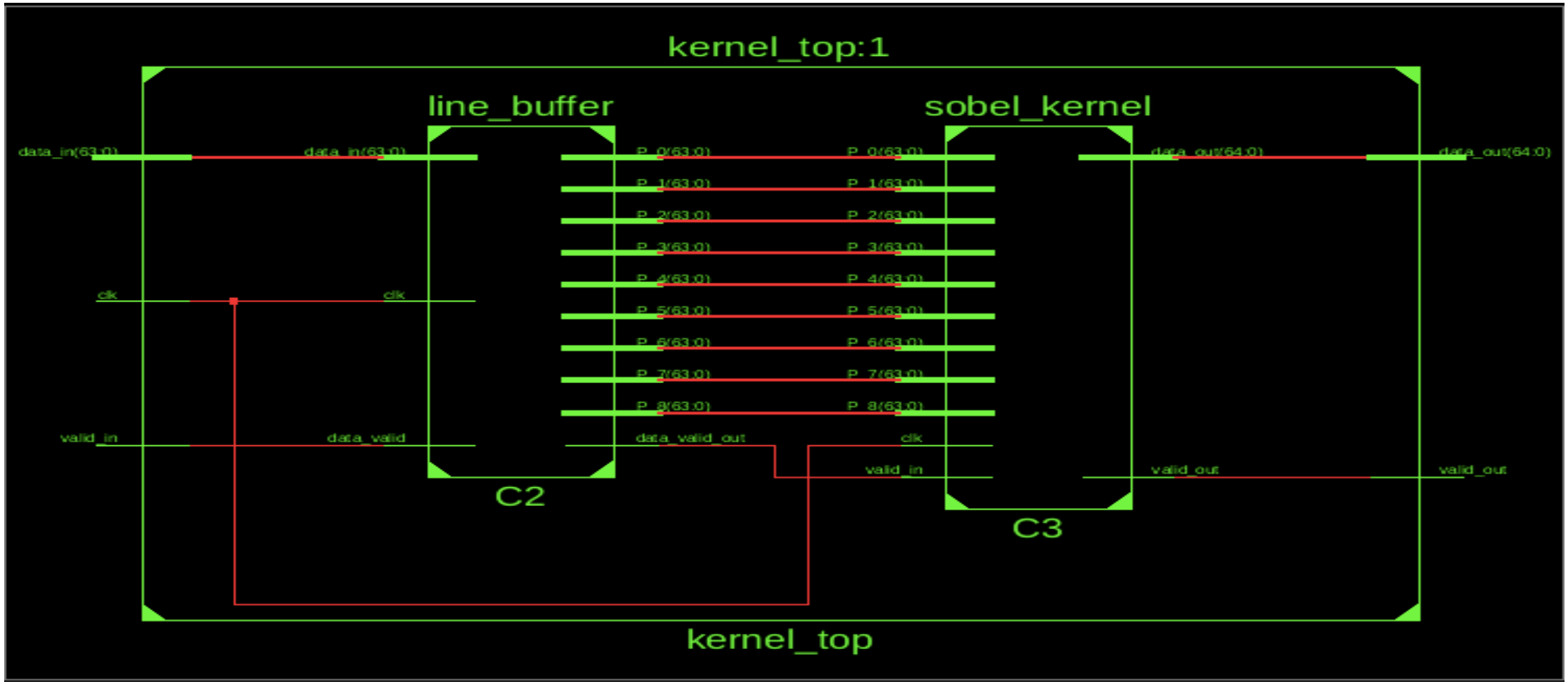
Simulation results for top kernel

For simulation the input values were given such a way that the corner pixels of the frame that has the highest contrast change triggers the peaking flag. Hence, from the simulation we can see that the peaking flag is high after a certain delay at predictable positions which proves that the IP works functionally fine.

Resource utilization

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	120	35200	0%
Number of Slice LUTs	686	17600	3%
Number of fully used LUT-FF pairs	15	791	1%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	2	80	2%

- The resource utilization shows that the IP utilizes very less resources compared to the available resources for xc7z020-3clg400 device used in AXIOM beta

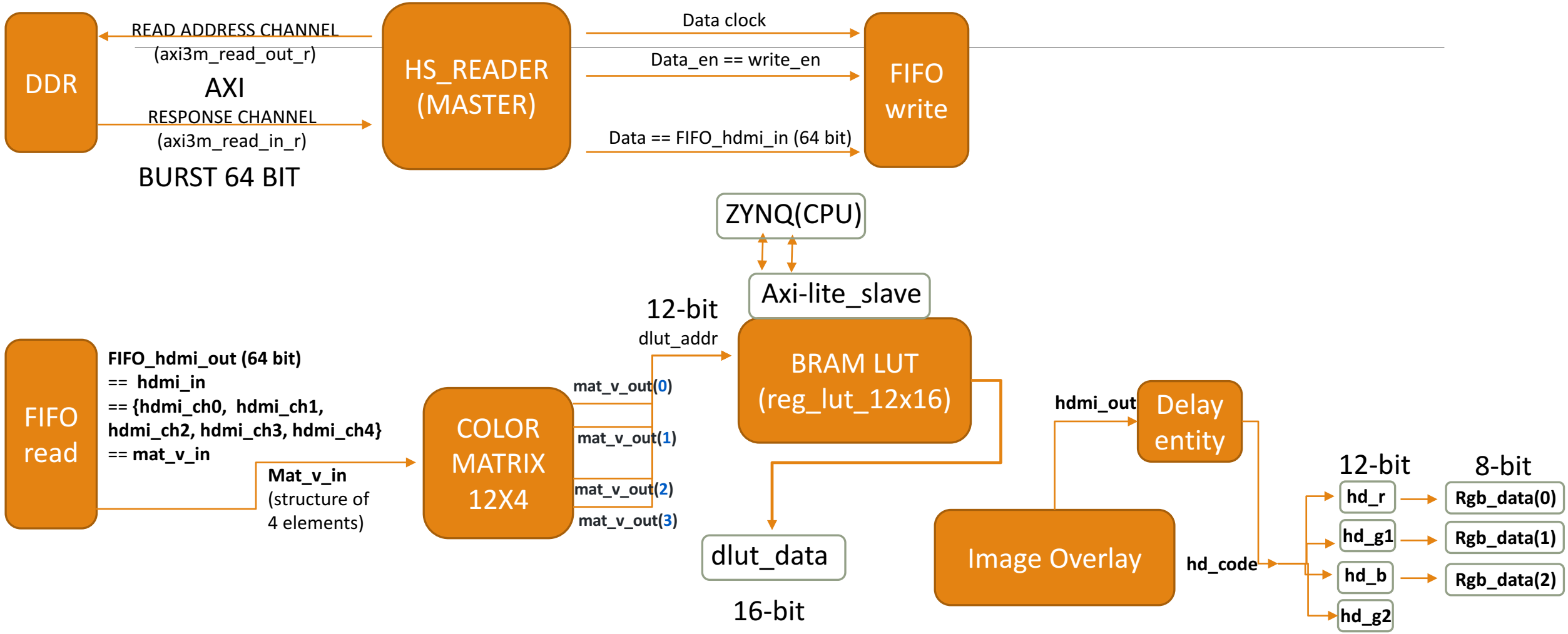


RTL schematic of the IP

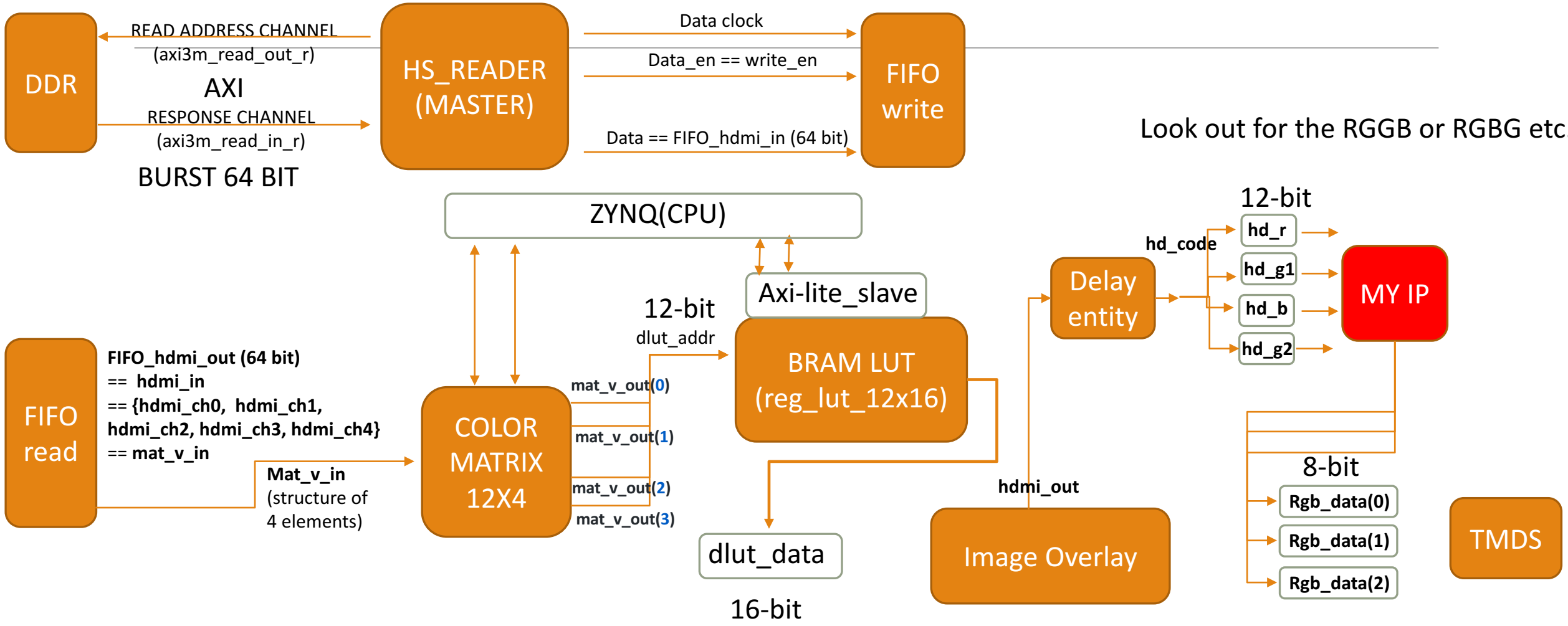
Instantiation

INSTANTIATE THE IP INTO THE AXIOM PIPELINE

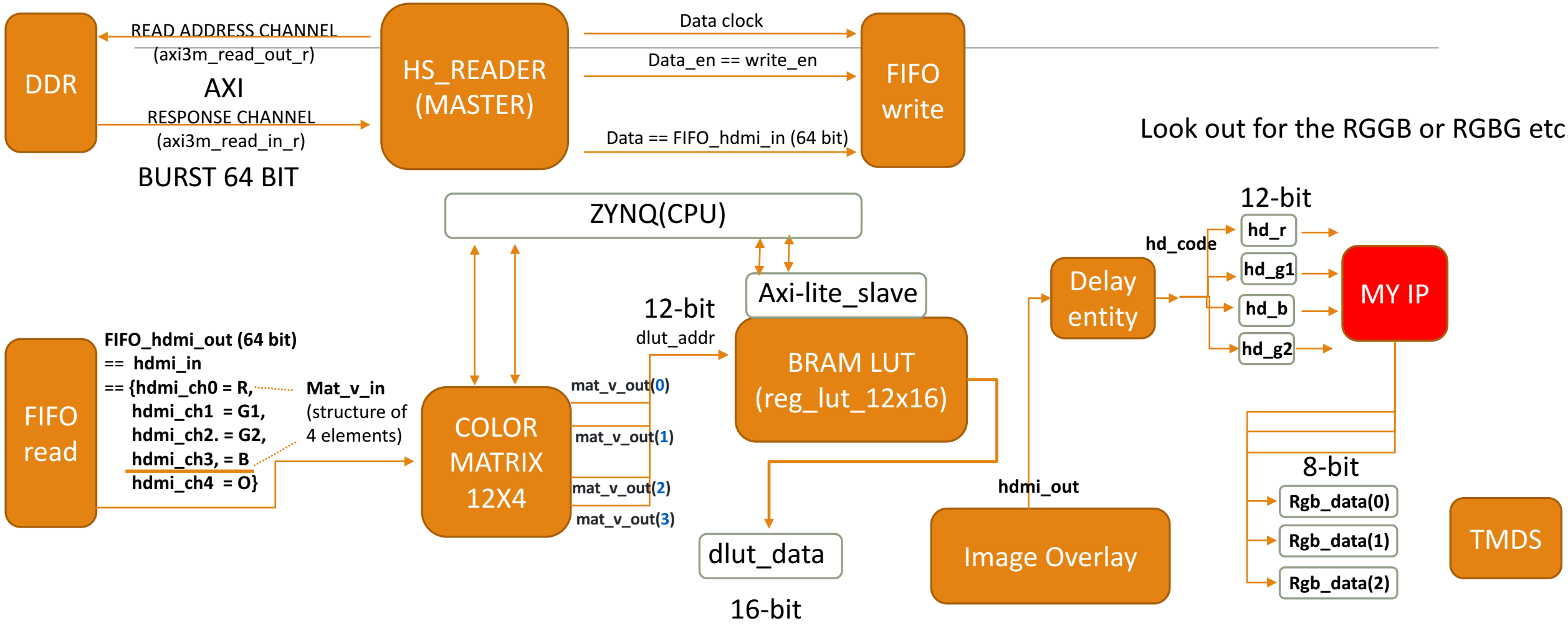
Output Image Processing Pipeline



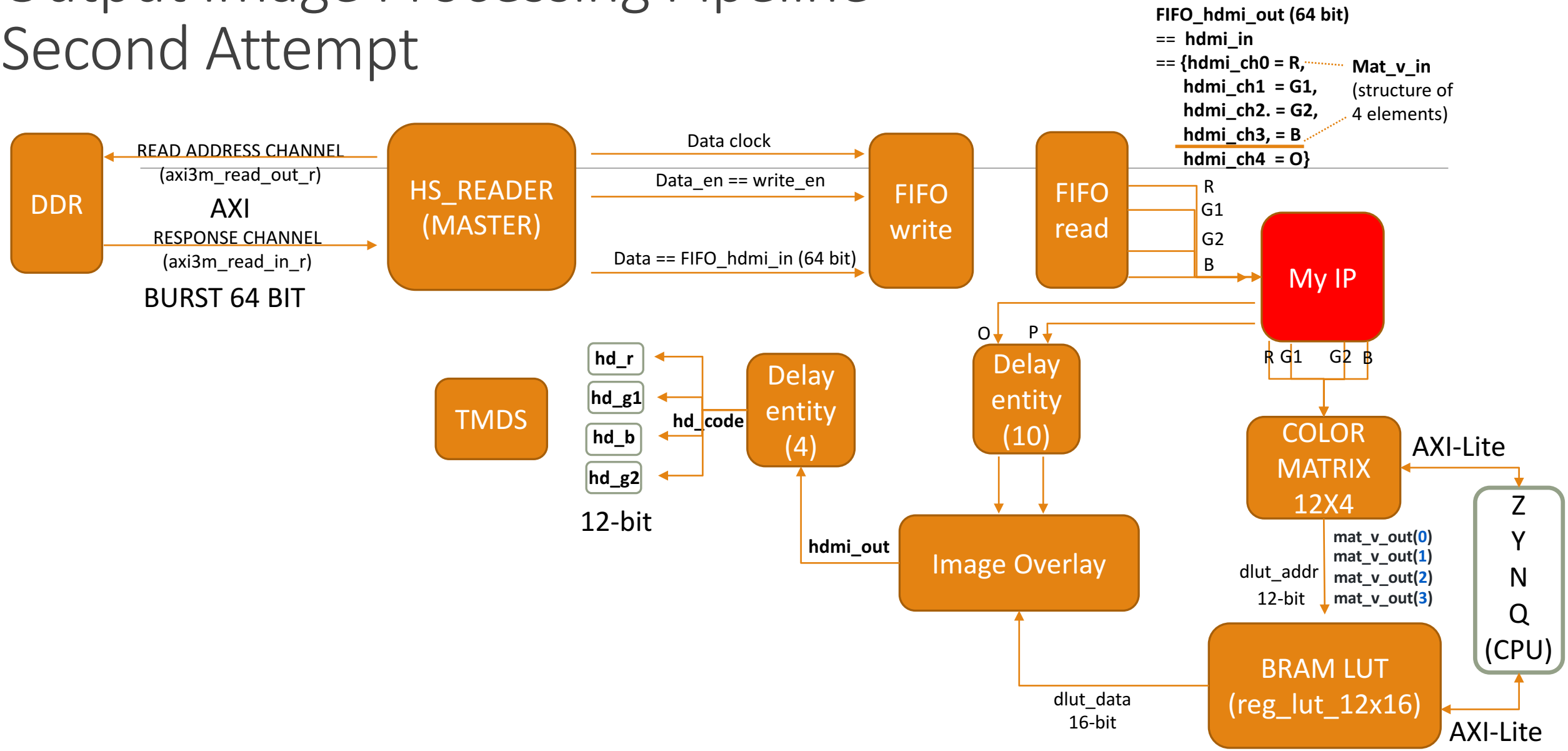
Output Image Processing Pipeline (IP placement) First attempt



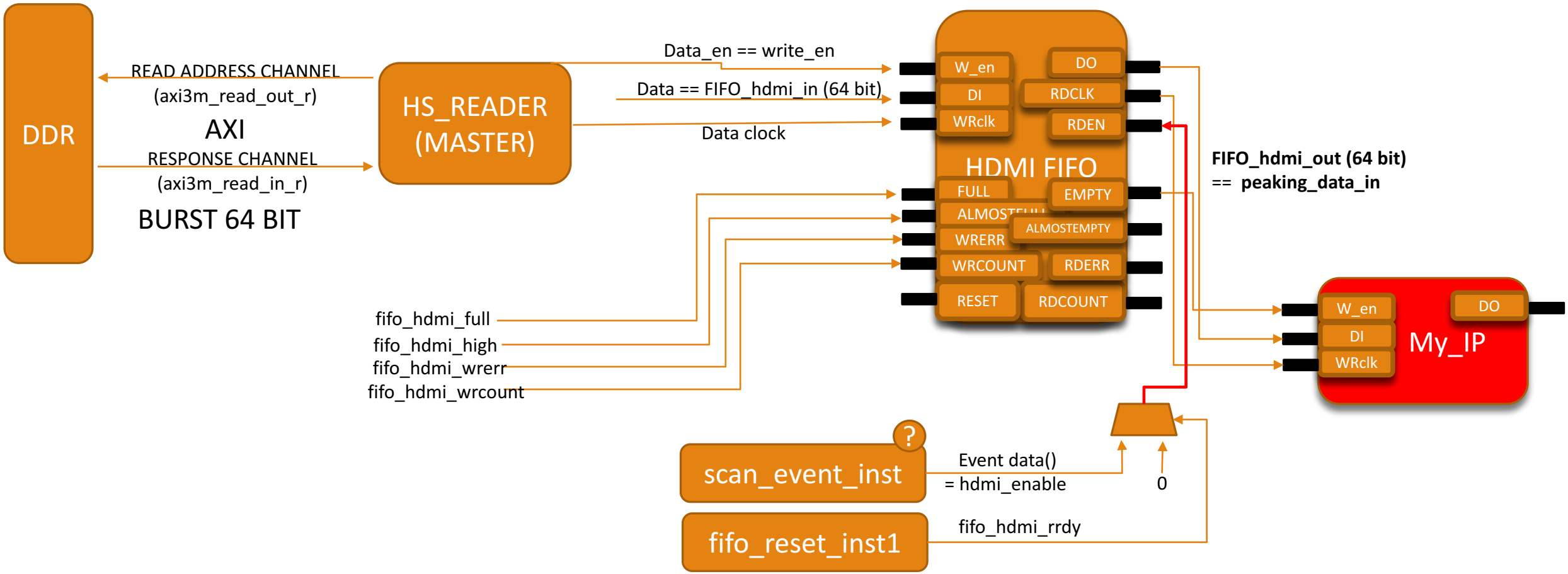
Output Image Processing Pipeline (IP placement) First attempt



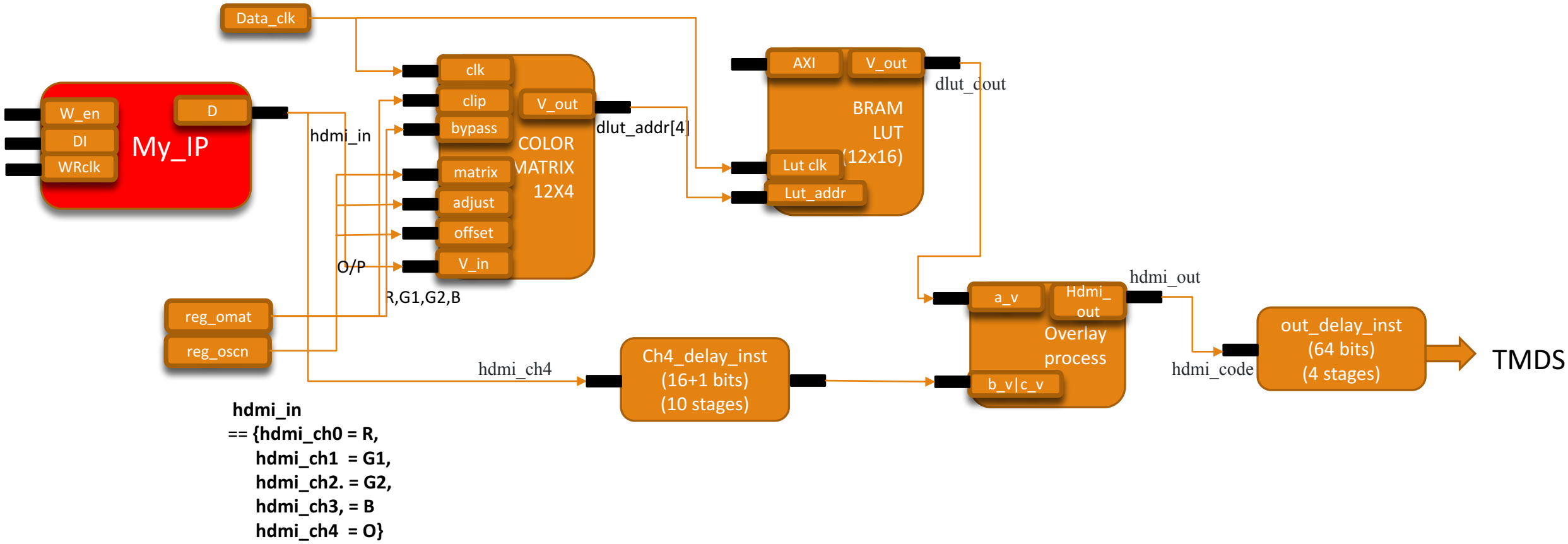
Output Image Processing Pipeline Second Attempt



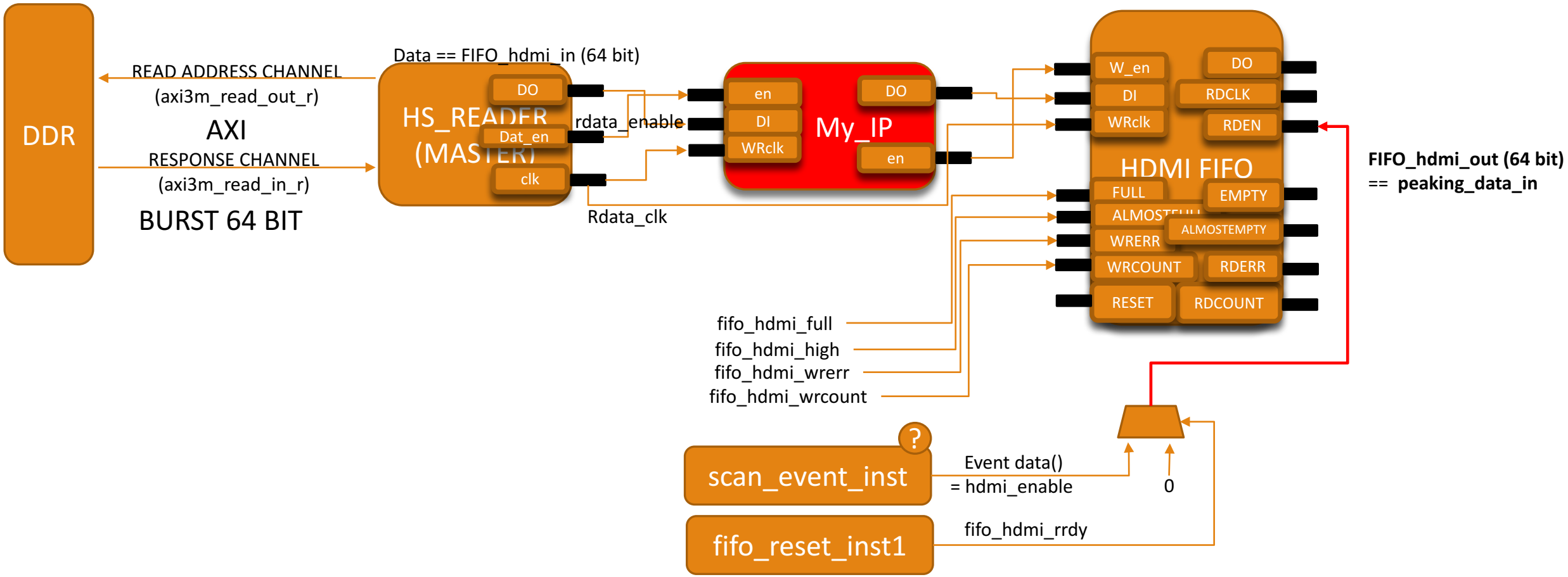
Output Image Processing Pipeline
Second Attempt-Detailed representation of the IP input



Output Image Processing Pipeline
Second Attempt-Detailed representation of the IP output



Output Image Processing Pipeline third Attempt



Conclusions

- The simulation results show that the designed IP generates peaking information for every incoming pixel data of the video frame. Such hardware based processing of the image processing algorithm will firstly, increase the overall processing speed of the system and secondly, decrease the load on the main processor.

Future Works

Firstly, my task will be to remove the synchronization bug in the current AXIOM Beta pipeline so that my IP can be smoothly instantiated in the pipeline and secondly, Video processing units like AXIOM beta faces serious hardware resource constraints, which could hamper further incorporation o efficient dedicated image processing hardware units. This could be dealt quite effectively by using run time partial reconfiguration techniques. I would like to implement different components of the current AXIOM Beta pipeline on the FPGA fabric using run time partial reconfiguration so that only the components required at the specific time will be active while others wont be. This would not only save fabric resources but also save power.

END