Study of Hardware-Software Co-design of Vehicle(Car) detection using HOG and SVM

R&D presentation

Neelam Sharma (18307R030)

Indian Institute of Technology, Bombay

June 28, 2020

Overview

Motivation

- Feature extraction is the first processing step in most CV tasks.
- CNN learned features outperform the HOG handcrafted features in visual object classification and detection tasks.
- Although learned features achieve more than $2\times$ accuracy, it comes at a large $311\times$ to $13{,}486\times$ overhead in energy consumption.[1]
- Modern CV algorithms require high computational complexity, which makes their deployment on battery-powered devices challenging due to the tight energy constraints.

[1] Amr Suleiman et al. "Towards closing the energy gap between HOG and CNN features for embedded vision". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017, pp. 1–4.

Histogram of Oriented gradients (HOG)

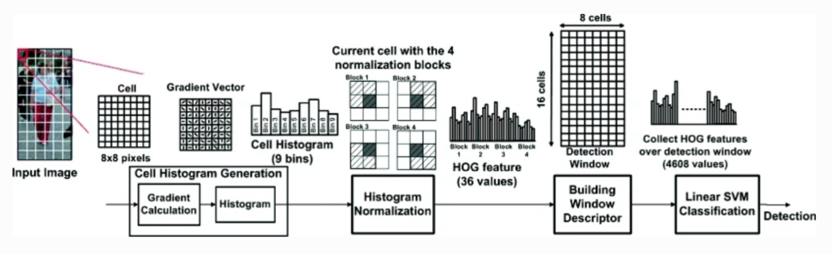

Figure: Overall flow: Object detection algorithm using HOG features[2]

[2] C Bagavathi and O Saraniya. "Hardware Designs for Histogram of Oriented Gradients in Pedestrian Detection: A Survey". In: *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE. 2019, pp. 849–854.

Stages of HOG

1. **Preprocessing**
   - Optional global image normalization equalisation that is designed to reduce the influence of illumination effects.
   - Replace each colour channel value as log or square root of the value

$$new\_pixel\_val_i = \sqrt{old\_pixel\_val_i}\ (\text{OR}) = \log(old\_pixel\_val_i) \tag{1}$$

2. **Gradient calculation**
   - Computes the first-order image gradients.
   - These capture contour, silhouette and some texture information, while providing further resistance to illumination variations.
   - The gradients in x and y directions are computed by convoluting with 1×3 and 3×1 sized kernels respectively to obtain $\mathrm{G_x}$ and $\mathrm{G_y}$.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad K_y = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$$

   - Magnitude and gradients are computed as:

$$\mid G(x,y) \mid = \sqrt{G_x^2 + G_y^2}. \tag{2}$$

$$tan(x,y) = \frac{G_y}{G_x}. \tag{3}$$

Gradient Calculation

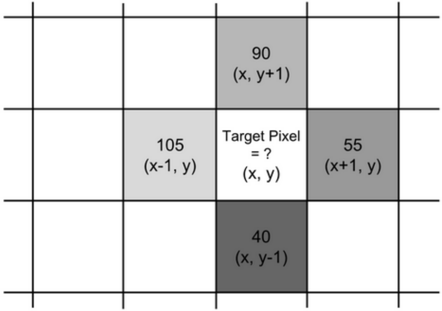

Figure: Four neighbours are required to calculate gradients at (x,y)[3]

[3] Lilian Weng. *Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS*. https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html. Accessed: 2020-06-04.

3 **Histogram extraction**

- A sliding window of size 40×100 pixel is composed of 6×18 blocks. Each block consists of 3×3 cells and a cell has a histogram of 9 discrete bins.

$$Total\ no.\ of\ features = 6 \times 18 \times 3 \times 3 \times 9 = 8748$$

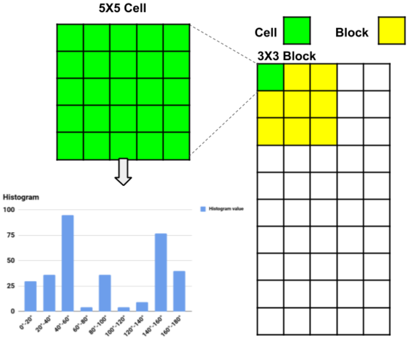

Figure: Histogram extraction from 5 × 5 pixels cell and 3 × 3 block

Stages of HOG (Contd.)

4 **Histogram Normalization**

- Orientation histograms are normalized over the blocks to which they belong.

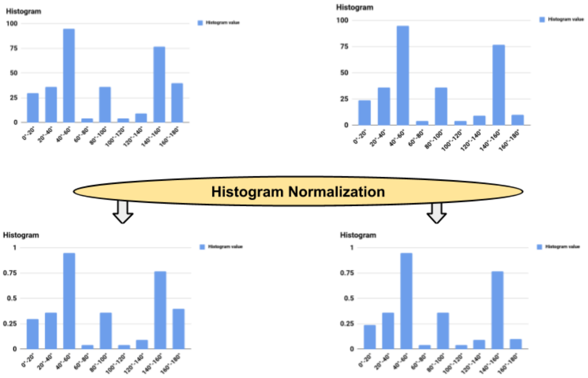

Figure: Histogram normalization over 3×3 block

Support Vector Machine

- Popular supervised Machine Learning Algorithm used for classification along HOG features[4].
- LinearSVC implementation is used as an SVM-classifier in this work. Equation which is used to form the hyperplane is given by:

$$y = w^T x + b \tag{4}$$

where, w = SVM coefficients, x = Feature vector, b = Bias term

- Prediction is done as:

$$\begin{aligned} y > 0;\ \ & Detected\ \ car(s) \\ y <= 0;\ \ & Detected\ \ no\ \ car \end{aligned} \tag{5}$$

[4] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 886–893.

Exploration with non-linear kernels

Kernel	Predicted +ve Actual +ve	Predicted +ve Actual -ve	Predicted -ve Actual +ve	Predicted -ve Actual -ve
LinearSVC	114	1	0	95
Sigmoid	114	1	0	95
RBF/Gaussian	114	1	0	95
Polynomial (degree=1)	114	1	0	95
Polynomial (degree=2)	114	1	0	95
Polynomial (degree=4)	114	1	0	95
Polynomial (degree=8)	114	2	0	94

Table: Confusion matrix comparison of Non-linear kernels for SVM with C=1.0 for all kernels except LinearSVC

Non-maximum Suppression

- Object detection using HOG suffers from a problem of detecting multiple bounding boxes around the same object.

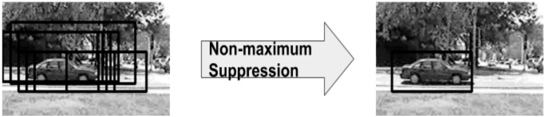

Figure: Non-maximum Suppression filters out bounding boxes

- Overlapping of two sliding windows with areas area1, area2 and an overlap area represented as overlap_area is given by:

$$Overlapping = \frac{overlap\_area}{area1 + area2 - overlap\_area} \tag{6}$$

- Overlapping threshold: 30%

Approximate arctan2 implementation

- Orientation of gradients are computed as:

$$arctan2(y,x) = \begin{cases} atan(\frac{y}{x}), & \text{if } x > 0 \\ atan(\frac{y}{x}) + 180^\circ, & \text{if } x < 0 \text{ } and \text{ } y \geq 0 \\ atan(\frac{y}{x}) - 180^\circ, & \text{if } x < 0 \text{ } and \text{ } y < 0 \\ +90^\circ, & \text{if } x = 0 \text{ } and \text{ } y > 0 \\ -90^\circ, & \text{if } x = 0 \text{ } and \text{ } y < 0 \\ \text{undefined}, & \text{if } x = 0 \text{ } and \text{ } y = 0 \end{cases} \tag{7}$$

- In HLS, $arctan2(y,x)$ is implemented using CORDIC(COordinate Rotation DIgital Computer) algorithm.
- CORDIC algorithms are slow to compute $arctan2(y,x)$ because they are sequential in nature[5].

[5] Abhisek Ukil, Vishal H Shah, and Bernhard Deck. "Fast computation of arctangent functions for embedded applications: A comparative analysis". In: *2011 IEEE International Symposium on Industrial Electronics*. IEEE. 2011, pp. 1206–1211.

Approximate arctan2 implementation (Contd.)

Linear interpolation

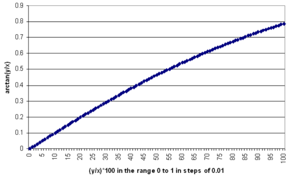

Figure: arctan(x) vs 100*(x) [5]

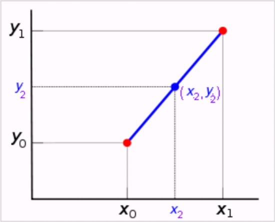

Figure: Linear interpolation between two points [5]

- Using linear interpolation the value of y_2 for x_2 in Figure 7 can be determined by fitting the curve between (x_1, y_1) and (x_2, y_2) linearly as:

$$y_2 = y_0 + (x_2 - x_0)\frac{(y_1 - y_0)}{(x_1 - x_0)} \tag{8}$$

Hardware-Software Co-design Approach

Possible design approaches:

1. Implement complete functionality in software side.
2. Implement complete functionality in the hardware side.
3. Implement a Hardware-Software Co-design.

Techniques for Hardware-Software Co-design

- Algorithmic partitioning
- Communication Constraints
- Accelerator-level parallelism

Algorithm partitioning

- To obtain a first cut analysis for algorithm-wise partitioning profiling using line_profiler[6] of single-threaded program was performed.

Function	# Hits	Total time	Time per hit	% Time
HOG feature computation	217	9379319.0	43222.7	82.9
Prediction using LinearSVC	217	292502.0	1347.9	2.6
Non-maximum suppression	1	403	403	0

Table: line_profile results of top 3 most time consuming parts of the application. **Timer units: 1e-06s**

- HOG feature extraction is the most time consuming part of the entire application.

[6] Robert Kern. *line_profiler and kernprof.* https://github.com/pyutils/line_profiler. Accessed: 2020-06-04.

Communication constraints

- Algorithmic partitioning should be done after considering communication constraints.
- Communication between PS(Processing System) and PL(Processing Logic) can become a bottleneck if communication overheads are not considered in the overall design process.
- Shared memory is also used for addressing communication constraints between PS and PL for improving performance.
- After 8748 HOG-features computation it is required them to be passed to Linear-SVM classification stage, which if implemented on PS will cause a communication overhead.

Accelerator-level parallelism

- Computation of HOG-features is independent for all sliding windows.
- More than one accelerators can be used for speeding up the execution.
- This method can improve performance for sure but at the cost of increased resource utilization.

System-level overview

- Processing logic (Microblaze core) initializes BRAM.
- HOG-SVM then transfers the initialized sliding window from BRAM to its internal memory.
- HOG-SVM computes the confidence score of the presence of a car in an image and returns its value through AXI slave interface connected to the PS.

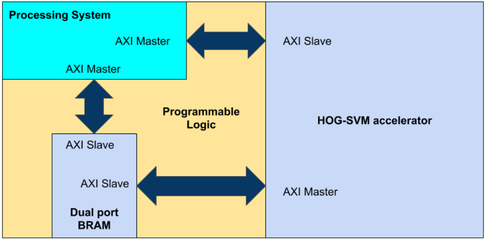

Figure: System-level view of the accelerator

Accelerator Design: Implementation of approximate arctan2

- LUT holds 101 values of $atan(arg)$ where $arg \in [0, 1]$.
- Using signs of x,y and linear interpolation $arctan2(x, y)$ can be computed.

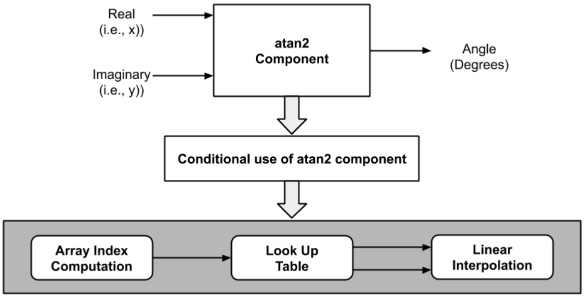

Figure: Computation flow of arctan2 function

Accelerator Design: Block Design of Accelerator

Figure: Block diagram of the accelerator

Two external interfaces are provided from accelerator:

- **Master-AXI interface**, that requests the data from memory providing the address of the location to be accessed.
- **Slave-AXI interface**, connected to Microblaze core to return the confidence_score obtained.

Accelerator algorithm

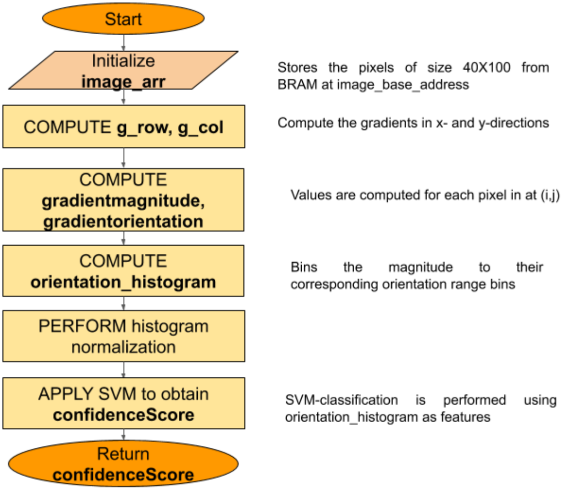

Figure: HOG-SVM accelerator algorithm flow

Overall system integration on Vivado

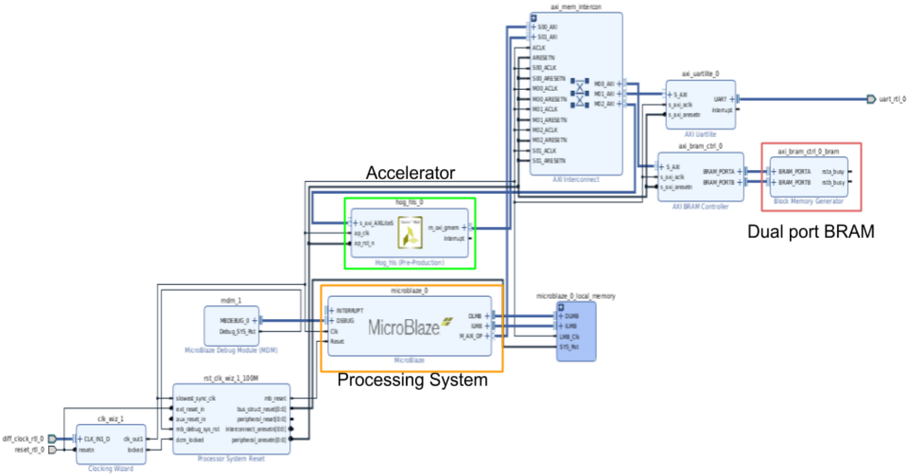

Figure: Block diagram of the entire design

Results: Setup

Setup

- **Dataset:** UIUC Image Database for Car Detection[7]
- **Software:** Vivado HLS 2019.1 and Vivado 2019.1
- **FPGA Family:** Artix-7
- **Xilinx Part no:** XC7A200T-1SBG484C

[7] Shivani Agarwal, Aatif Awan, and Dan Roth. *UIUC Image Database for Car Detection*. 2004. URL: https://cogcomp.seas.upenn.edu/Data/Car/.

Synthesis: Comparison of approximate arctangent function with CORDIC implementation

Synthesis comparison results obtained with two implementations of $arctan2(y, x)$ in Vivado HLS:

- Timing Summary

Parameter	**Approximate arctan()**	**HLS arctan()**
Estimated clock frequency	9.378 ns	8.552 ns

Table: Synthesized timing summary results of two arctan() implementations

- Latency

Parameter	**Approximate arctan()**		**HLS arctan()**	
Latency (in cycles)	**min**	**max**	**min**	**max**
	3	274	41	196

Table: Latency (in cycles) comparison

Synthesis: Comparison of approximate arctangent function with CORDIC implementation (Contd.)

- Utilization

Parameter	Approximate arctan()	HLS arctan()
BRAM_18K	1%	1%
DSP48E	3%	2%
FF	2%	2%
LUT	9%	9%

Table: Utilization Summary comparison

Synthesis: Accelerator

Synthesis comparison results obtained with two designs of accelerator in Vivado HLS as:

1. **Design A:** Design not optimized with any of the pragmas.
2. **Design B:** Design optimized with pragmas provided by Vivado HLS.

- Timing summary comparison

Clock	Target (in ns)	Estimated (in ns)		Uncertainity (in ns)
		Design A	Design B	
ap_clk	10.00	9.342	8.75	1.25

- Comparison of latency in clock cycles

Latency (Design A)		Latency (Design B)		Interval (Design A)		Interval (Design B)	
min	max	min	max	min	max	min	max
1517922	2932532	792487	1253287	1517972	2932532	792487	1253287

Synthesis: Accelerator(Contd.)

- Utilization Summary

Name	BRAM_18K		DSP48E		FF		LUT	
	Design A	Design B	Design A	Design B	Design A	Design B	Design A	Design B
DSP	-	-	-	3	-	-	-	-
Expression	-	-	2	1	0	0	3224	4047
FIFO	-	-	-	-	-	-	-	-
Instance	5	2	46	24	18980	16713	21683	18633
Memory	170	103	-	-	0	0	0	0
Multiplexer	-	-	-	-	-	-	1478	2765
Register	-	0	-	-	2249	3580	-	256
Total	175	105	48	28	21229	20293	26385	25701
Available	730		740		269200		134600	
Utilization (%)	23	14	6	3	7	7	19	19

Table: Utilization summary comparison

Synthesis: Overall block design

Resource	Utilization	Available	Utilization %
LUT	11541	133800	8.63
LUTRAM	405	46200	0.88
FF	12461	267600	4.66
BRAM	116	365	31.78
DSP	28	740	3.78
IO	5	285	1.75
BUFG	3	32	9.38
MMCM	1	10	10.00

Table: Post-Implementation Utilization Summary of Design B in Vivado

C-simulation

Console ✕ | Errors | Warnings | DRCs

Vivado HLS Console

```
-0.282757, -0.282444, -0.283434, -0.283584, -0.283868, -0.285263, -0.285263, -0.286140, -0.286925, -0.286925, -0.287107, -0.287710, -0.288634, -0.289780, -0.
-0.323810, -0.323980, -0.323980, -0.323969, -0.322849, -0.323729, -0.324578, -0.325452, -0.325452, -0.325452, -0.325700, -0.325700, -0.325700, -0.325700, -0.
-0.370050, -0.370050, -0.369969, -0.369969, -0.375193, -0.375552, -0.375552, -0.375552, -0.375552, -0.375552, -0.375552, -0.375552, -0.376776, -0.376575, -0.
-0.388581, -0.392765, -0.390852, -0.392983, -0.394421, -0.394421, -0.394421, -0.393289, -0.393185, -0.393211, -0.393580, -0.393119, -0.393615, -0.393615, -0.
-0.427130, -0.427204, -0.427204, -0.427784, -0.428438, -0.430772, -0.430058, -0.430058, -0.430058, -0.430058, -0.430943, -0.430943, -0.431007, -0.431007, -0.
-0.526001, -0.525539, -0.525539, -0.525539, -0.525945, -0.526998, -0.527535, -0.527535, -0.528098, -0.528098, -0.526666, -0.527274, -0.527274, -0.533001, -0.
-0.580904, -0.581333, -0.580401, -0.579224, -0.577666, -0.579837, -0.580969, -0.580969, -0.584646, -0.585290, -0.585290, -0.584930, -0.578549, -0.579098, -0.
-0.616002, -0.615820, -0.615515, -0.617022, -0.618091, -0.618091, -0.618068, -0.619146, -0.619238, -0.619238, -0.619646, -0.619646, -0.621187, -0.620495, -0.
-0.639745, -0.639745, -0.639745, -0.639828, -0.641844, -0.640545, -0.640545, -0.638080, -0.638696, -0.639625, -0.639625, -0.641405, -0.641405, -0.644654, -0.
-0.725065, -0.725065, -0.725065, -0.725065, -0.725065, -0.725065, -0.725065, -0.724150, -0.724150, -0.724150, -0.724150, -0.724150, -0.724150, -0.724150, -0.
-0.773452, -0.772101, -0.771480, -0.770657, -0.770657, -0.770657, -0.770657, -0.770657, -0.770804, -0.770804, -0.770690, -0.770154, -0.770154, -0.770154, -0.
-0.829776, -0.823790, -0.822673, -0.822673, -0.829859, -0.831151, -0.831931, -0.832939, -0.832727, -0.832727, -0.833072, -0.833072, -0.833209, -0.832961, -0.
-0.910965, -0.908113, -0.908113, -0.907403, -0.907982, -0.909676, -0.910381, -0.912703, -0.913328, -0.913709, -0.913835, -0.913428, -0.913428, -0.913428, -0.
-0.989721, -0.985484, -0.983826, -0.980690, -0.978809, -0.978809, -0.978050, -0.977190, -0.975644, -0.975644, -0.976686, -0.976826, -0.976826, -0.978740, -0.
-1.007020, -1.007237, -1.005906, -1.005906, -1.007400, -1.008438, -1.010030, -1.010030, -1.010715, -1.012519, -1.013141, -1.013141, -1.013141, -1.018155, -1.
-1.064322, -1.064322, -1.064268, -1.061888, -1.061284, -1.057619, -1.057619, -1.052780, -1.052359, -1.051283, -1.051443, -1.051206, -1.051206, -1.049370, -1.
-1.117553, -1.117357, -1.116626, -1.116626, -1.117239, -1.118464, -1.116601, -1.114562, -1.113995, -1.113303, -1.108739, -1.104768, -1.103570, -1.102633, -1.
-1.121773, -1.121124, -1.121865, -1.128385, -1.136040, -1.136272, -1.136272, -1.136272, -1.136272, -1.136272, -1.136272, -1.136272, -1.136969, -1.138283, -1.
-1.143988, -1.143988, -1.143240, -1.143240, -1.143373, -1.143243, -1.143243, -1.146102, -1.146316, -1.146316, -1.146316, -1.146316, -1.146316, -1.146316, -1.
-1.279661, -1.279661, -1.279734, -1.280331, -1.280747, -1.280747, -1.280747, -1.281362, -1.287675, -1.291014, -1.293083, -1.296141, -1.296141, -1.296141, -1.
-1.366411, -1.365394, -1.364569, -1.364569, -1.368084, -1.369252, -1.368058, -1.369516, -1.369516, -1.369516, -1.370597, -1.370933, -1.370933, -1.376312, -1.
-1.434413, -1.435459, -1.436045, -1.437981, -1.437438, -1.439194, -1.440199, -1.440055, -1.438866, -1.437046, -1.437046, -1.438698, -1.439560, -1.441216, -1.
-1.462969, -1.462733, -1.462733, -1.461346, -1.461346, -1.465444, -1.465444, -1.463677, -1.462926, -1.462926, -1.461377, -1.460960, -1.460813, -1.460813, -1.
-1.506279, -1.513175, -1.517254, -1.517139, -1.518663, -1.518442, -1.520493, -1.520825, -1.520520, -1.520294, -1.520294, -1.521241, -1.521241, -1.521667, -1.
-1.556228, -1.556228, -1.556228, -1.556228, -1.556228, -1.556228, -1.556228, -1.557307, -1.557307, -1.557307, -1.557307, -1.557307, -1.557307, -1.557307, -1.
-1.549616, -1.550506, -1.550506, -1.550270, -1.550270, -1.550270, -1.550270, -1.550269, -1.550269, -1.545555, -1.545993, -1.545993, -1.545993, -1.546168, -1.
-1.626357, -1.627039, -1.627039, -1.627039, -1.629176, -1.629478, -1.629804, -1.629804, -1.629412, -1.629412, -1.629806, -1.630055, -1.631055, -1.631987, -1.
-1.781791, -1.781791, -1.781791, -1.781791, -1.781791, -1.784585, -1.785836, -1.788403, -1.788954, -1.789516, -1.790637, -1.790645, -1.790645, -1.790645, -1.
-1.896447, -1.898004, -1.899772, -1.900763, -1.900763, -1.898232, -1.899545, -1.899545, -1.902217, -1.902217, -1.902657, -1.901259, -1.901686, -1.898735, -1.
-1.968115, -1.968115, -1.967104, -1.965560, -1.969046, -1.969663, -1.969856, -1.972544, -1.970046, -1.968280, -1.968280, -1.968481, -1.968481, -1.969227, -1.
-2.058059, -2.057902, -2.058134, -2.058134, -2.058134, -2.059465, -2.059465, -2.061120, -2.060023, -2.059723, -2.058716, -2.058599, -2.056831, -2.053071, -2.
Confidence score = -2.102940
Error = 0.000000
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Figure: HLS C-simulation output of the accelerator

RTL Co-simulation

Figure: HLS RTL Co-simulation output of the accelerator

Microblaze RTL simulation

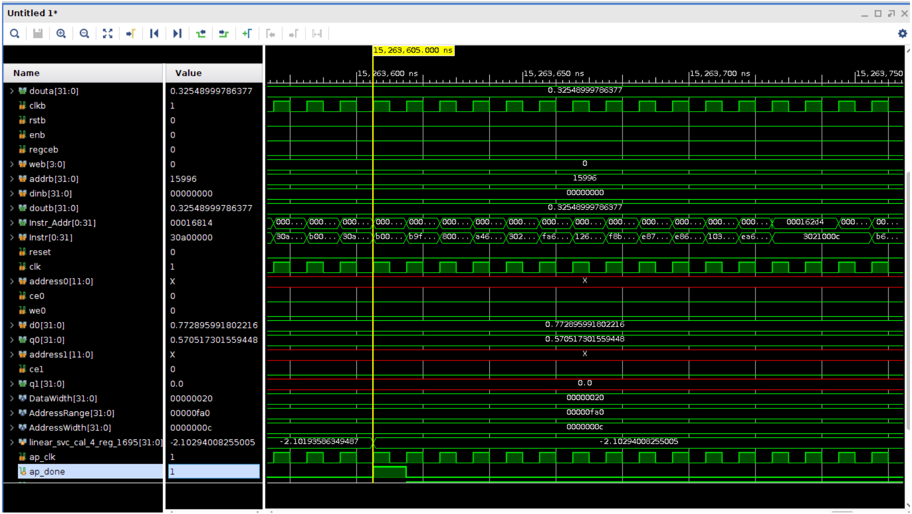

Figure: Microblaze RTL Behavioural simulation

Conclusion

- This work presented the implementation of HOG-SVM accelerator using High-level-Synthesis(HLS) considering hardware and software co-design approach.
- Overall accelerator integration into the system has been tested using behavioural simulation using Vivado 2019.1.

References I

Amr Suleiman et al. "Towards closing the energy gap between HOG and CNN features for embedded vision". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017, pp. 1–4.

C Bagavathi and O Saraniya. "Hardware Designs for Histogram of Oriented Gradients in Pedestrian Detection: A Survey". In: *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE. 2019, pp. 849–854.

Lilian Weng. *Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS.* https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html. Accessed: 2020-06-04.

References II

Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 886–893.

Abhisek Ukil, Vishal H Shah, and Bernhard Deck. "Fast computation of arctangent functions for embedded applications: A comparative analysis". In: *2011 IEEE International Symposium on Industrial Electronics*. IEEE. 2011, pp. 1206–1211.

Robert Kern. *line_profiler and kernprof*. https://github.com/pyutils/line_profiler. Accessed: 2020-06-04.

Shivani Agarwal, Aatif Awan, and Dan Roth. *UIUC Image Database for Car Detection*. 2004. URL: https://cogcomp.seas.upenn.edu/Data/Car/.

Thank You