מס' פרויקט:

20-1-1-2187

שם הפרויקט:

מימוש וסימולציה של מאיץ למערכות לומדות על רכיב FPGA

מגישים:

שי צבר 208723627

חיים גרודה 312562721

מנחה:

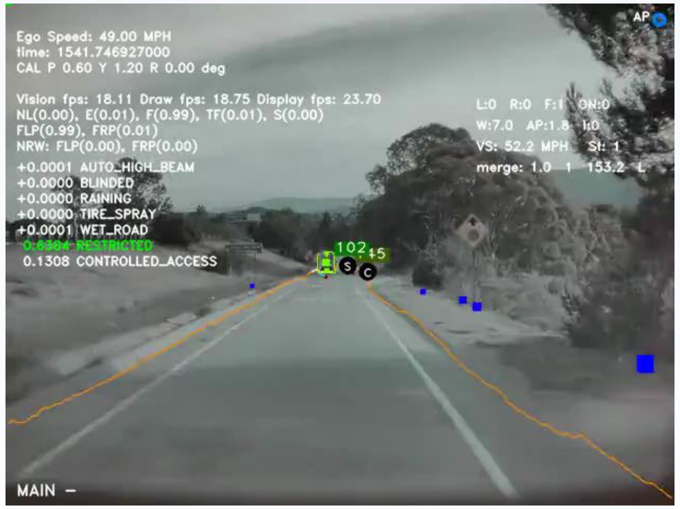
יוני זייפרט

מקום ביצוע:

אוניברסיטה

CNN – Convolutional Neural Network

- Achieved great success in image classification, speech recognition and more.
- Research hotspot in many scientific fields.
- Widely used in the industry, such as for autonomous cars, security systems, health and more.



```
Ego Speed: 49.00 MPH
time: 1541.746927000
CAL P 0.60 Y 1.20 R 0.00 deg

Vision fps: 18.11 Draw fps: 18.75 Display fps: 23.70
NL(0.00), E(0.01), F(0.99), TF(0.01), S(0.00)
FLP(0.99), FRP(0.01)
NRW: FLP(0.00), FRP(0.00)
+0.0001 AUTO_HIGH_BEAM
+0.0000 BLINDED
+0.0000 RAINING
+0.0000 TIRE_SPRAY
+0.0001 WET_ROAD
 0.6384 RESTRICTED
 0.1308 CONTROLLED_ACCESS

L:0  R:0  F:1  ON:0
W:7.0  AP:1.8  I:0
VS: 52.2 MPH  St: 1
merge: 1.0  1  153.2  L
```

Video: Tesla computer vision

CNN Structure

- **Convolution –** Feature extraction using filters
- **Activation –** Introduce nonlinearity
- **Pooling –** Reduce spatial dimensions
- **Fully Connected –** Neural network (matrix multiplications)

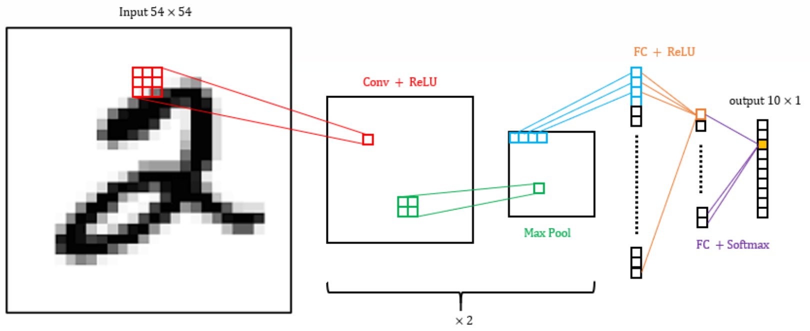


Figure: Implemented CNN model

Motivation and Objectives

Motivation

- High accuracy comes with long computation time. Approx. 90% of CNN computations are in the convolutional layers
- Computation resources cause high-power consumption
- The need to include such image recognition capabilities in embedded systems with tight real-time and power constraints

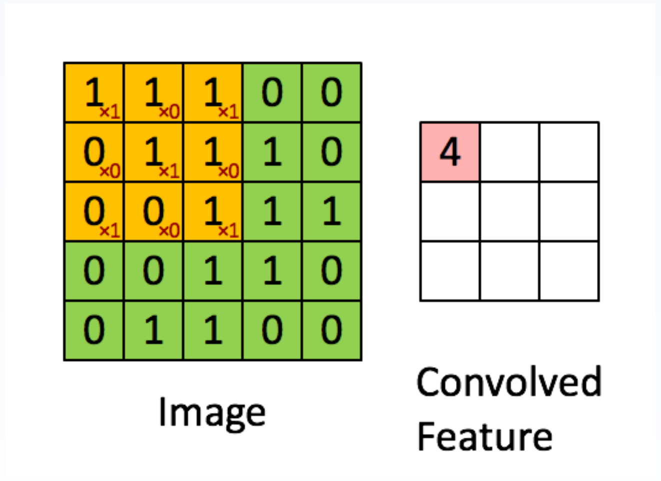
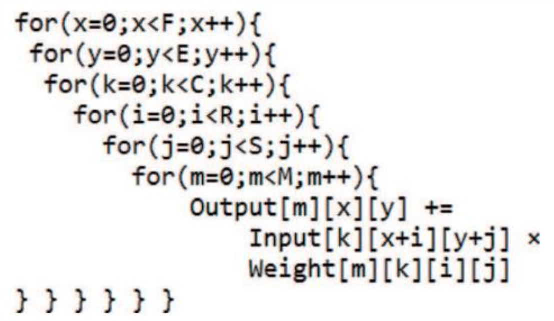


Figure: 2D convolution. [source]

Objectives

- **Acceleration** – by hardware parallelization
- **Accuracy** – reduce loss due to fixed point arithmetic
- **Power** – dedicated hardware processing elements



```
for(x=0;x<F;x++){
 for(y=0;y<E;y++){
  for(k=0;k<C;k++){
    for(i=0;i<R;i++){
      for(j=0;j<S;j++){
        for(m=0;m<M;m++){
            Output[m][x][y] +=
                Input[k][x+i][y+j] ×
                Weight[m][k][i][j]
} } } } } }
```

Figure: Nested ‘for’ loop for convolution calculation

Method

CNN Model

- Classification network
- Output vector of size 10
- Input image 54x54 pixels
- 6 layers deep

ZedBoard FPGA

- Zynq-7000 SoC
- Programmable logic (FPGA)
- Processing Subsystem (CPU)
- Vivado development tools

Fixed Point Calculations:

- Better computation time
- Less hardware resources
- Possible loss of accuracy

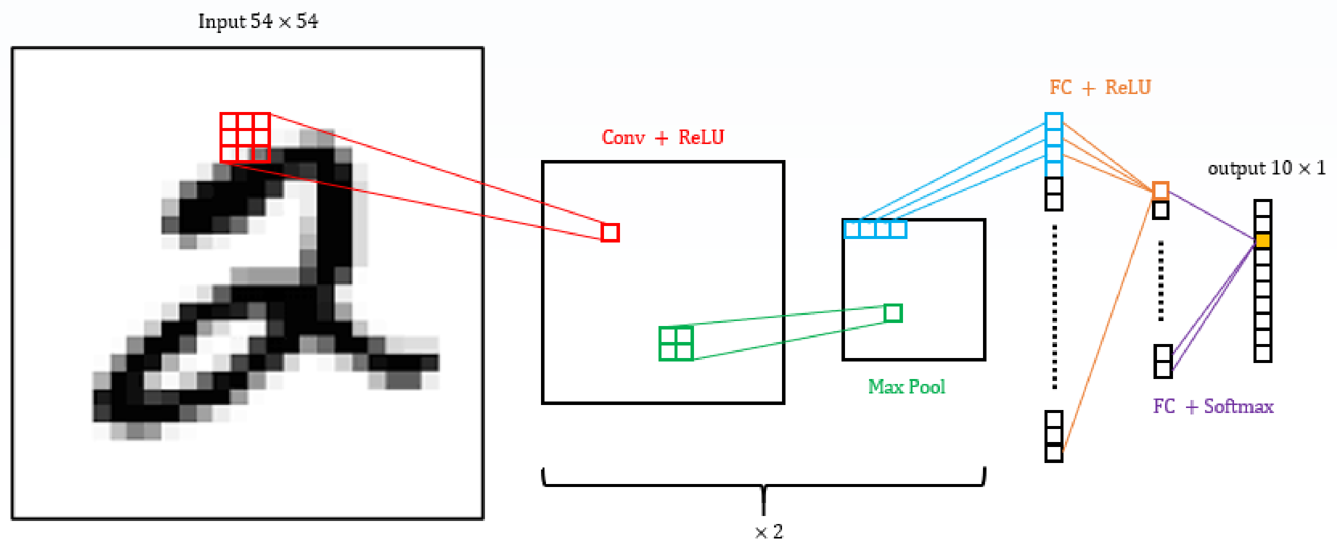


Figure: The implemented CNN classifying a handwritten digit

High Level Synthesis (HLS)

- RTL abstraction
- Focus on functionality
- Easily explore algorithmic changes
- Simple platform retargeting

Hardware Design

Processing system (PS):

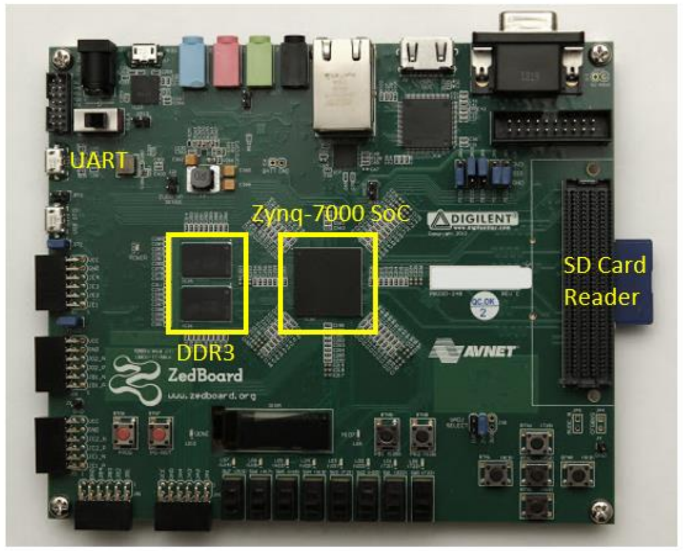
- ARM Cortex A9 667MHz CPU
- Input Output – SD card, UART
- Control and configuration
- DDR memory unit

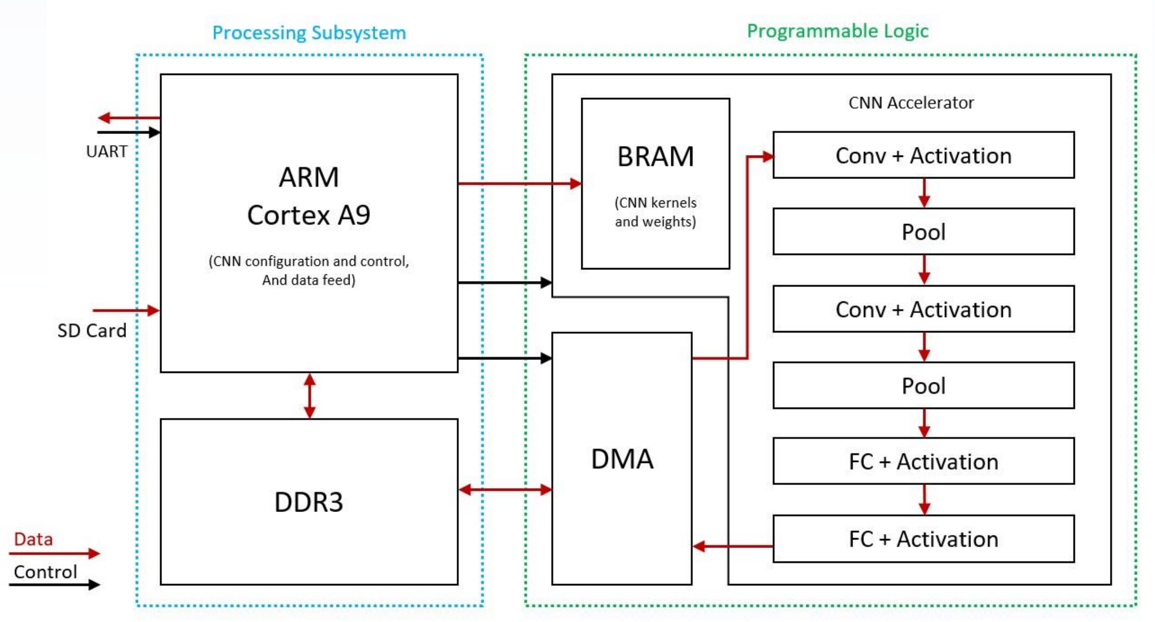
Programmable Logic (PL):

- Artix-7 FPGA with 6.6M logic cells
- DMA controller for data transfer
- Processing Element per CNN layer
- BRAM for kernels and weights

AXI interfaces

- Stream – high throughput (input data)
- Lite – static data (kernel/weights)





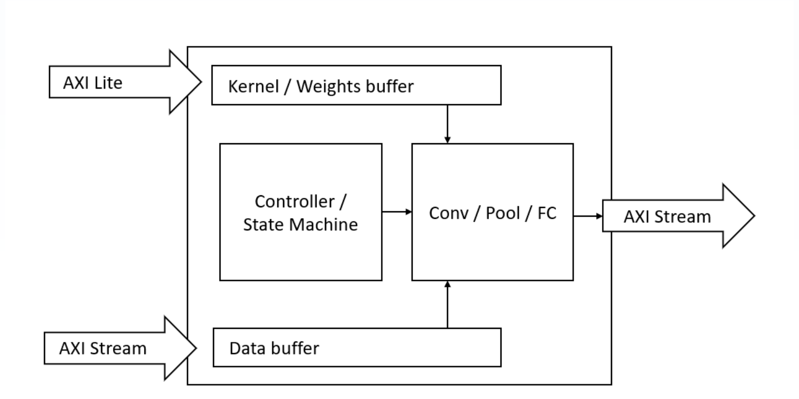
Hardware Design

Processing Element

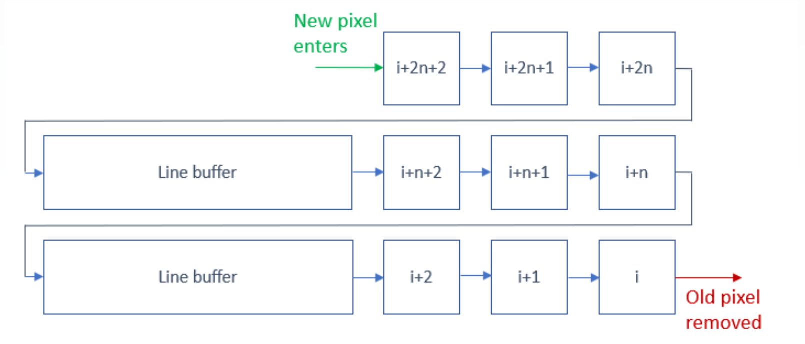
- Implements single CNN layer
- Includes BRAM buffers
- AXI List/Stream interfaces
- FIFO implementation
- Max/Avg pooling support
- ReLU activation support
- Auto generate HLS sources
- Auto generate Vivado designs

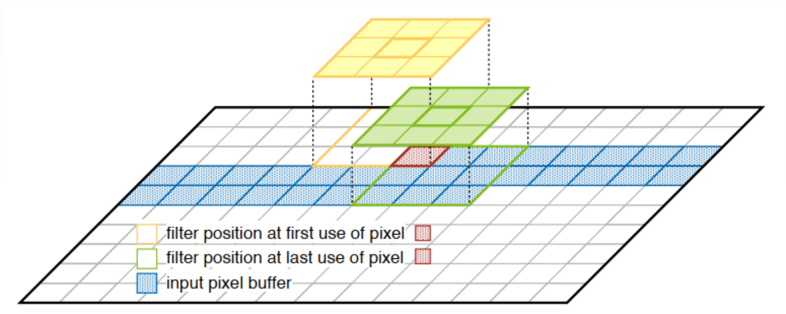
Sliding Window

- Neighborhood extraction
- Included per CONV/POOL PE
- Data reuse – Pixel transferred once
- Memory utilization



Figures: Processing Element design

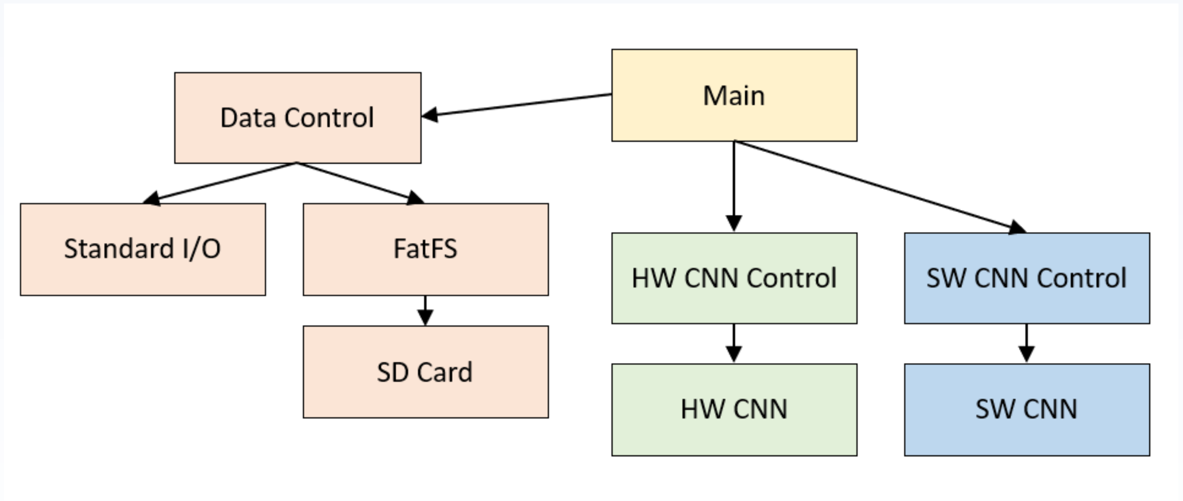



filter position at first use of pixel
filter position at last use of pixel
input pixel buffer

Figures: Data reuse / sliding window illustration

Software Design

- Cross platform: PC and FPGA
- Written in C in Vivado SDK
- Software implementation of CNN
- Hardware control modules
- Data control and statistics



FPGA platform (ARM Cortex A9 CPU)

- Bare Metal (no OS)
- FatFS – Free file system software module
- SD Card / UART I/O interfaces
- HAL headers generated by Vivado

PC platform (Intel i7 CPU)

- Linux based
- Single threaded
- Standard I/O interfaces
- Include simulation of HW calculations

HW Implementation

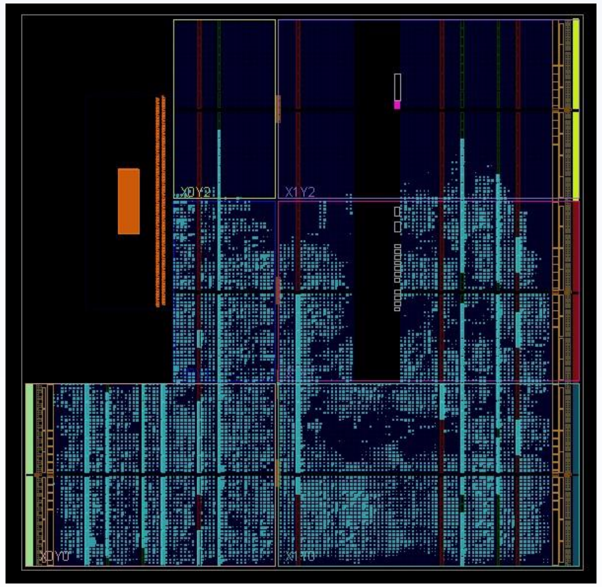


Figure: Implemented Design

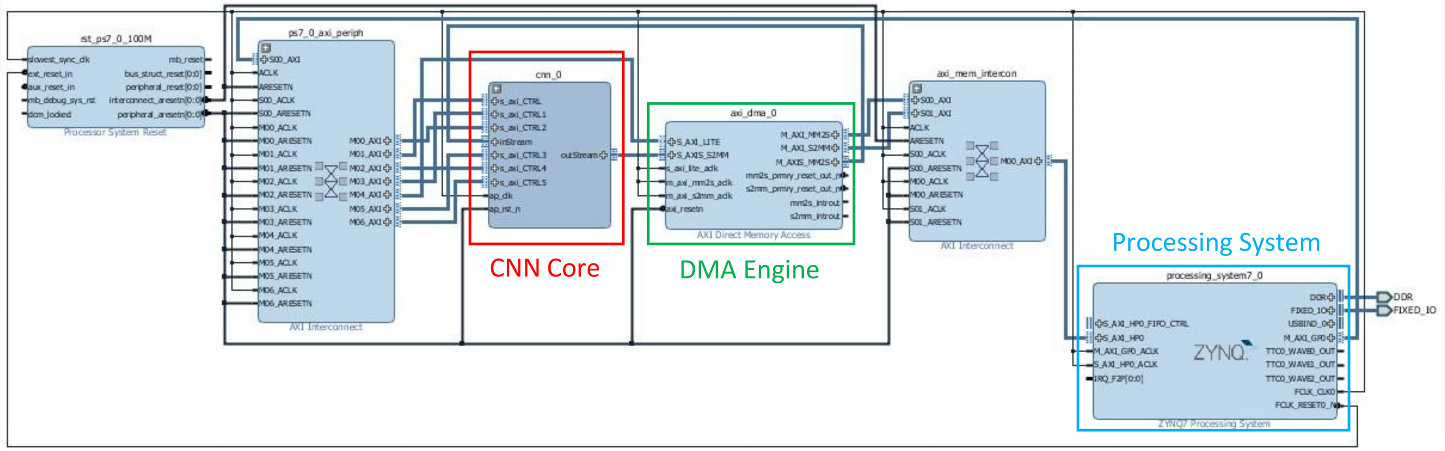


Figure: TOP block design

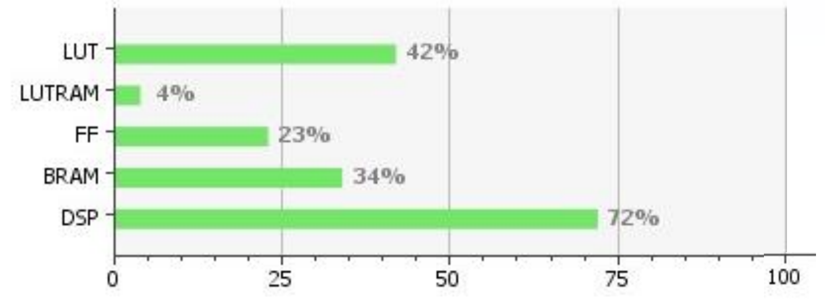


Table: FPGA utilization

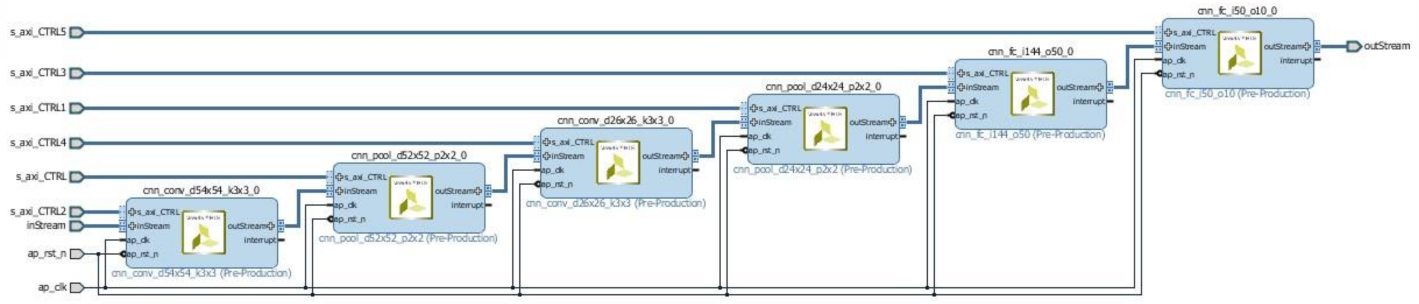


Figure: CNN core block design

Simulation Results

CNN Simulation

- Python Keras deep learning library
- MNIST dataset, 10K test images
- Kernels and weights fed to CNN

Measurements

- Statistics gathered by software
- Image processing time
- Result accuracy
- CNN classification certainty
- Vivado power estimation tool

	Floating point (FP64)	Fixed point (Q32.10)
Accuracy	96.03%	96.02%
Certainty	98.83%	98.82%

Table: CNN accuracy using different arithmetic

	PS	PL
Dynamic Power	77%	23%
Total Power	71%	29%

Table: Vivado power estimation results

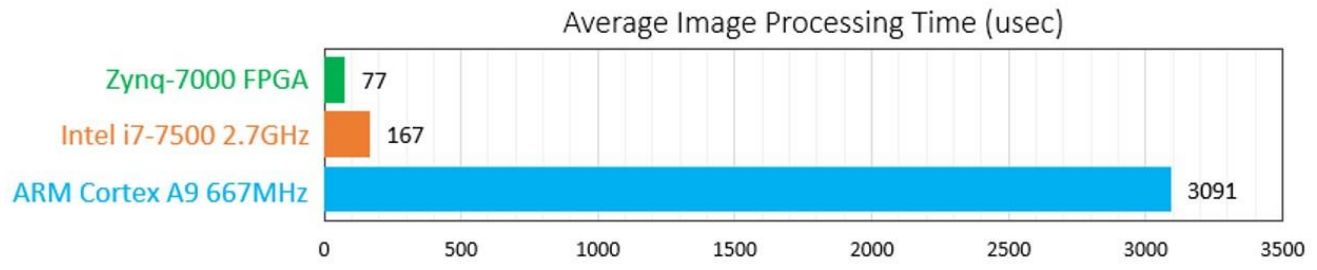


Table: Time comparison

Conclusions and Future Work

Conclusions

- Significant acceleration compared to SW
- Possible to keep accuracy also with fixed point
- Significant power conservation

Future Work

- Add convolution channels (RGB)
- Utilize second ARM core of the Zynq-7000 SoC
- Compare HW with multi threaded SW implementation
- Test accuracy loss for small fixed-point words
- Fully automate Vivado design generation

Appendix and References

- Project repository on GitHub
- ZynqNet - An FPGA-Accelerated Embedded CNN
- Science Direct - Convolutional Neural Networks