Ho Chi Minh City University of Technology
Faculty of Electrical & Electronics Engineering
Department of Telecommunications Engineering

FPGA platform applied for facial expression recognition using CNNs

Instructor: Prof. Dr. Thuong Le-Tien
Conducted by: Sy Nguyen-Tan

Ho Chi Minh City University of Technology.
Date: June 21st, 2019

Content

- Abstract
- Problem definition
- Model Implementation
- Hardware Design
- Experimental results
- Conclusion

Abstract

Abstract

- Cameras are used everywhere as flexible sensors for numerous applications
  - A required image processing should be local on embedded computer platforms with **performance requirements and energy constraints**
- A challenging problem is the design of efficient accelerators for that purpose, since some CNNs require a lot of repetitive image processing
  - I show and implement solutions to **quantize the trained CNN as well as perform and optimize computations on an Embedded System**
- The design flow is evaluated by implementing the previously trained CNN to recognize facial emotions from face image implemented in python on a PC
  - The project explains the process of **porting the CNN algorithm from python to C/C++ and then executing it on a ZYNQ FPGA board**
  - The bottleneck of the CNN is the convolutional layers and that is why **different solutions for that accelerator are analysed** and some of them implemented on VHDL, connected to the embedded processor and their performance is tested.

Problem Definition

Problem Definition

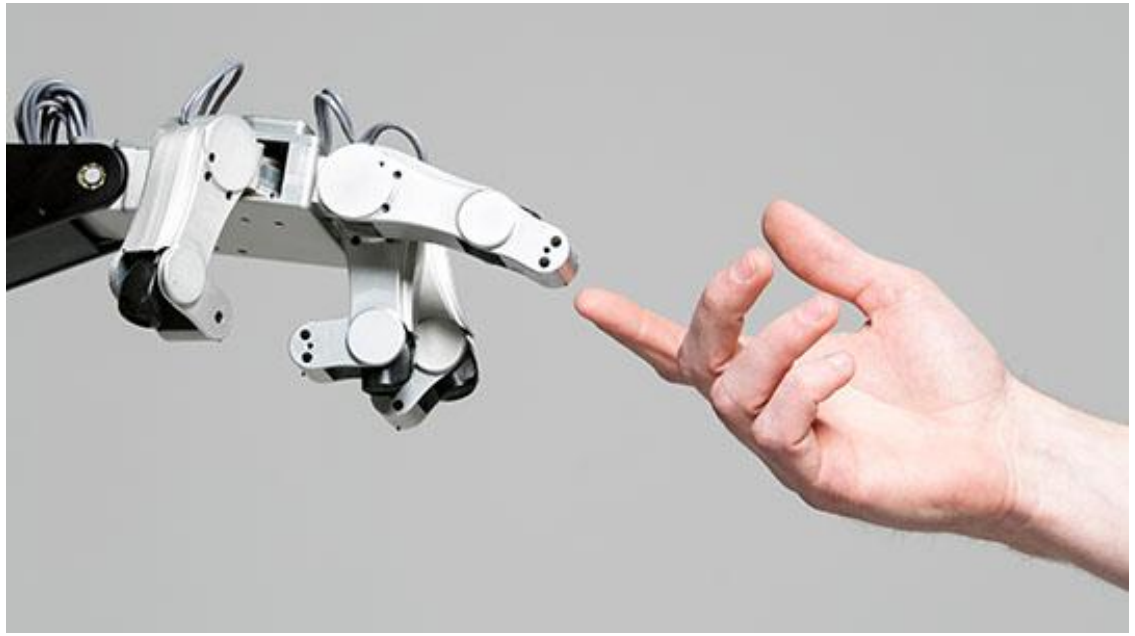


Figure [1]: Human- Robot Interaction

Figure [2]: Emotion Recognition

- Human- Robot interaction is increasing its attention.
=> **Understanding** the **facial gestures and visual cues** of an individual is a **need**

- FER system allows a robot to **understand the expression** of humans in turn **enhancing its effectiveness** in performing **various tasks**

Source: [1][2] Internet

Problem Definition

Figure [3]: Artificial Intelligence Development

- In recent years, **the development of artificial intelligence** opens up the opportunity to create **more complex systems** to **bring people and robots closer together**.

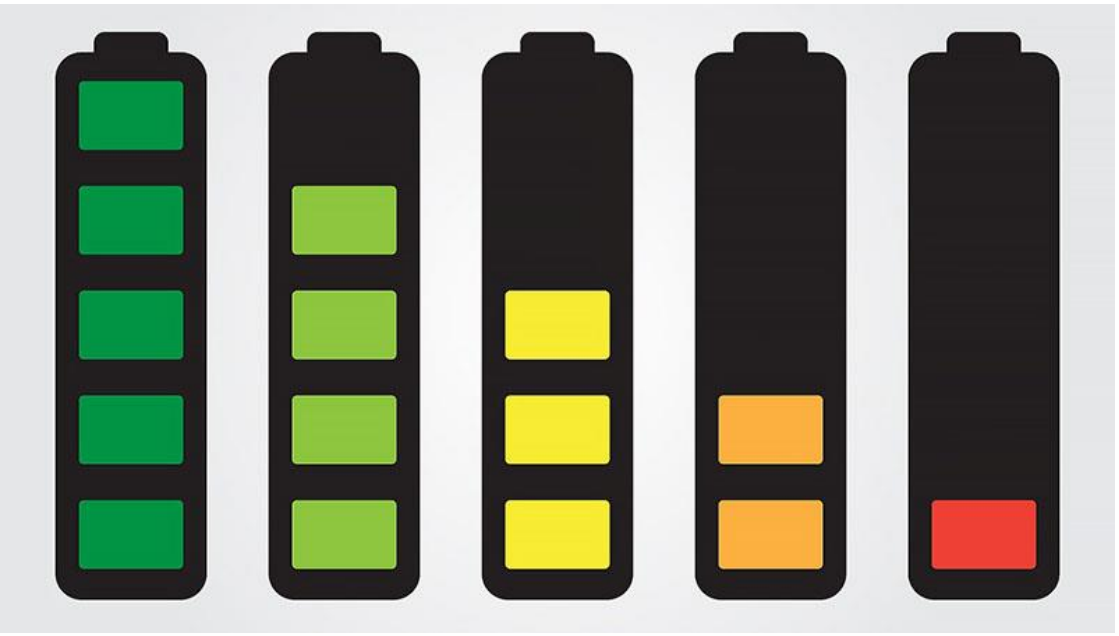


Figure [4]: Energy consumption Problem

- "**The energy industry is facing decades of transformation**"[5]. Yet the implications of the changes underway go far deeper. There are political, economic and social issues at stake, but it may also require each of us to make some **fundamental shifts** in our behavior too.

Source: [3][4] Internet

[5] 2016 World Energy Scenarios, **World Energy Council,** https://www.worldenergy.org/wp-content/uploads/2016/10/World-Energy-Scenarios-2016_Full-Report.pdf

Problem Definition

Figure [6] FPGA acceleration

Table [7]	GPU	FPGA
Execution Time (s)	**18.7**	**21.4**
Power Consumption (W)	**30**	**15**

- Furthermore, reduced data precision allows FPGA to reach performance in range of **Tera-Operations per second**

Source: [6]: Xilinx FPGA Power, https://www.renesas.com/us/en/solutions/key-technology/fpga-power-solutions/fpga-power-xilinx.html

[7]: Wonlai Zhao, Haohuan Fu, Wayne Luk, "F-CNN: An FPGA-based framework for training Convolutional Neural Network" 2016, IEEE 27th International Conference on Application-specific System, Architecture and Processors (ASAP)

Model Implementation

Dataset

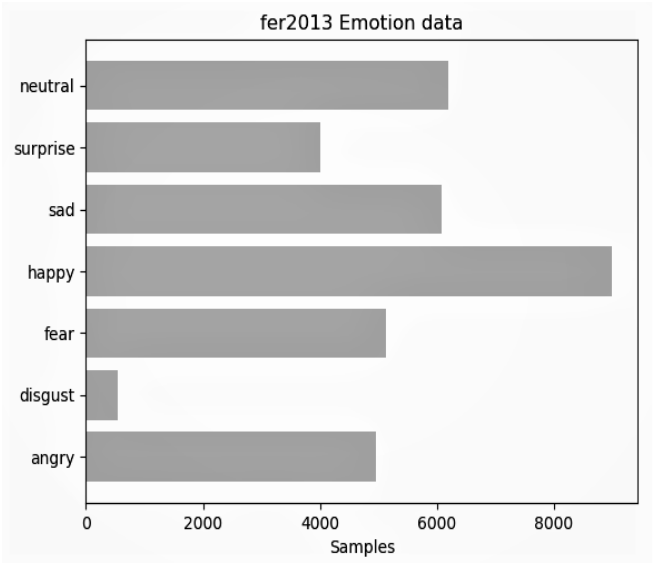


Figure [9] Structure of FER-2013 dataset

[8]	Training set	Public testing set	Private testing set
Image Quantity (sample)	**28709**	**3589**	**3589**

Table [10] FER-2013 details

- FERC-2013 has seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral) [9]

Source: [8] Internet
[9][10] Kaggle, FER2013, *https://www.kaggle.com/deadskull7/fer2013.*

Dataset

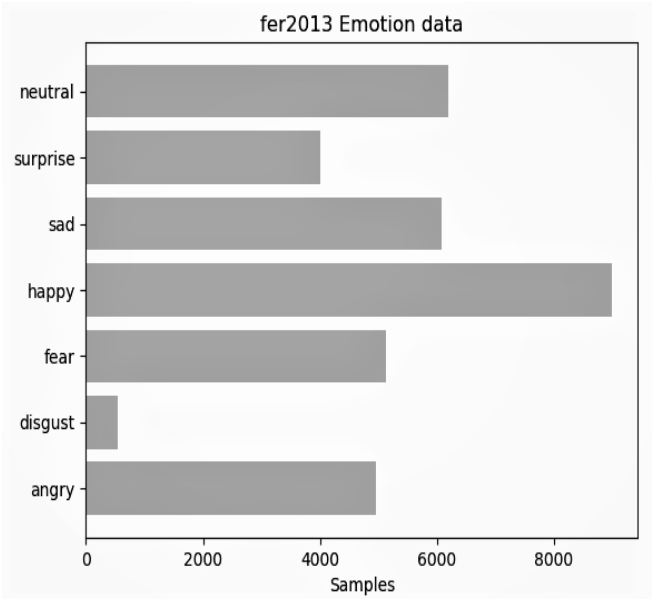


Figure [11] Structure of FER-2013 dataset

		Image Quantity (sample)
Emotion	Angry	4953
	Disgust	547
	Fear	5121
	Happy	8989
	Sad	6077
	Surprise	4002
	Neutral	6198

Table [12] FER-2013 details

Source: [11][12] Kaggle, FER2013, *https://www.kaggle.com/deadskull7/fer2013.*

Dataset Cleaning

1. Remove strange data

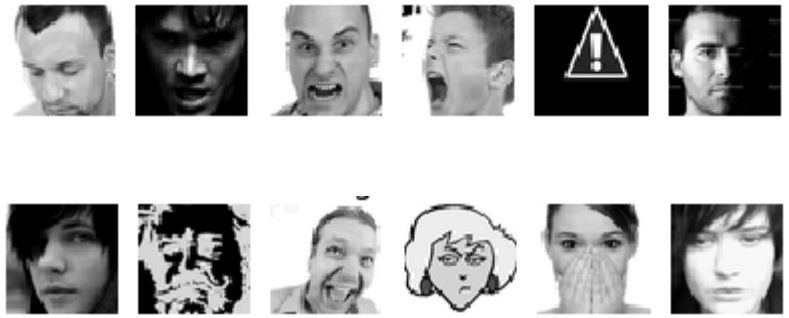


Figure [13] Some wrong image in FER-2013

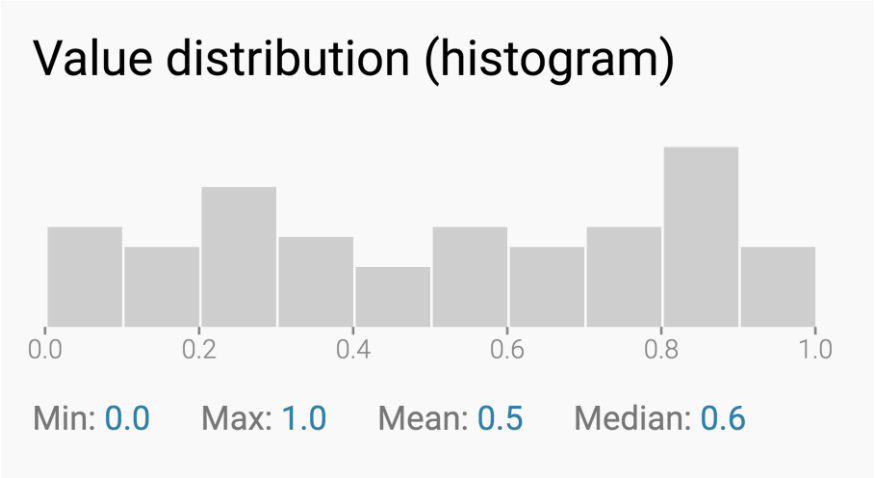


Figure [14] Example of histogram distribution

- There are some images that are not good (e.g. some images are pixelated, irrelevant, from animations)

  ⇨ The maximum of the histogram

Source: [13] Kaggle, FER2013, https://www.kaggle.com/deadskull7/fer2013.
[14] Internet

Dataset Cleaning

2. Merge class 0 and 1

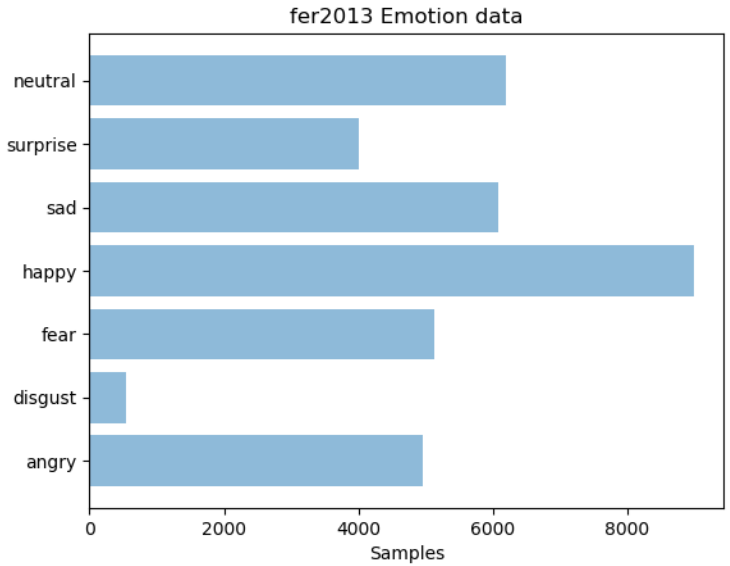


Figure [15] FER-2013 structure

Figure [16] Angry and Disgust samples

- ⇨ Merger class 0 and 1 together
- ⇨ The recognized emotions and labels are reduced to 6:
  - 0-(Angry + Disgust)
  - 1-Fear
  - 2-Happy
  - 3-Sad
  - 4-Surprise
  - 5-Neutral

- ◉ Class 1 has a very small amount
- ◉ This class, (disgust) is very similar to anger

Source: [15] Internet
[16] Kaggle, FER2013, *https://www.kaggle.com/deadskull7/fer2013.*

The correct data

Figure [17] FER-2013 correct data

- 31097 are left after 'strange images' removal.
- Deleted 1695 strange images.

⇨ The data is ready for preparation

- The data are a little bit modified to be correctly fed into the CNN
  - **Convert, normalize and subtract** the const mean value from the data images
  - Label values of the classes are converted to a binary one_hot vector.

Figure [18] Data after finishing preparation

Source: [17] Kaggle, FER2013, *https://www.kaggle.com/deadskull7/fer2013.*

Proposed System

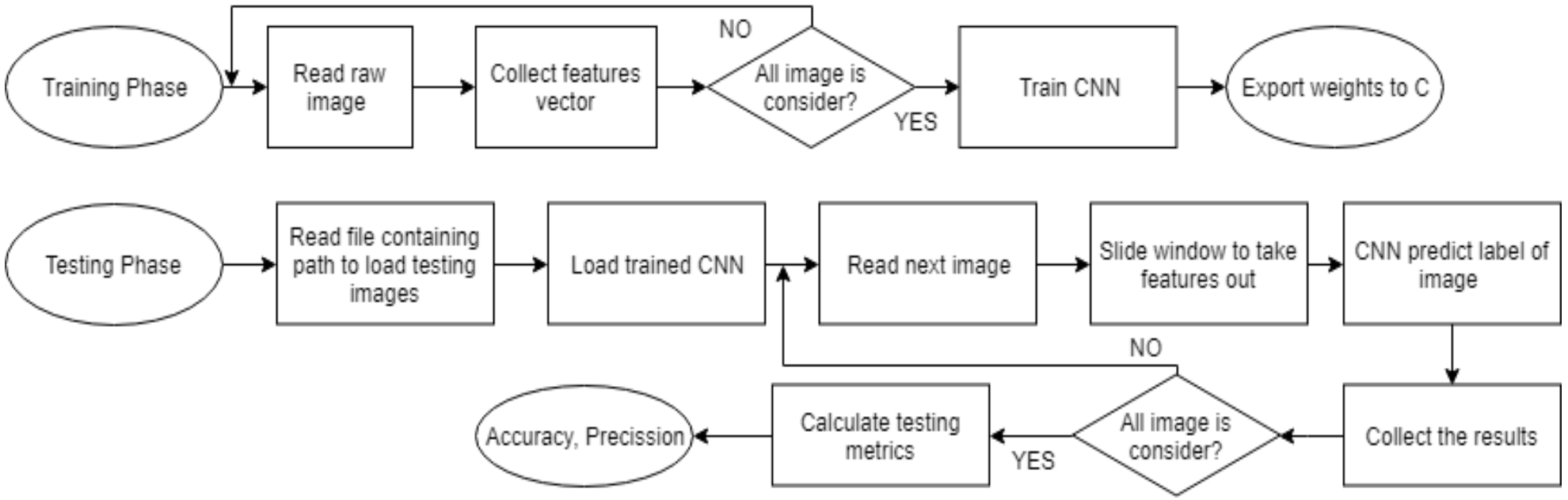


Figure [19] PC-stage diagram

- The GPU based processing is divided into 2 stages including training and testing process. After finishing the deployment of the system on the computer, the weights of the model is exported and downloaded to the FPGA system.

CNN Architecture

1. Model 1 - Overfitting the data TODO not overfitting with 35k data

- A baseline softmax classifier using **a single convolutional layer and a one fully connected layer**
- The equation of the classifier is simply:

$$Y = \textbf{softmax}(\textbf{\textit{ReLU}}(x^{*}W_1 + b_1)W_2 + b_2)$$

64 filters with size 8x8.
AdamOptimizer with a learning rate of 0.004

```
Iteration i= 700 , train accuracy= 0.328125 , loss= 1.61648
test accuracy= 0.265625

Iteration i= 800 , train accuracy= 0.40625 , loss= 1.49183
test accuracy= 0.25

Iteration i= 900 , train accuracy= 0.421875 , loss= 1.48842
test accuracy= 0.28125

Iteration i= 1000 , train accuracy= 0.4375 , loss= 1.36199
test accuracy= 0.203125
```

Figure [20] Training Process of model 1

⇨ The model was overfitting
  - test accuracy of only 28%.
  - It could not find any features with this architecture

CNN Architecture

Solution

- More advanced architectures.
  - Since the first convolutional layer will just extract some simplest characteristics of the image such as edges, lines and curves
  - Apply different techniques such as
    - **dropout and pool**
    - **implement a neural network of more layers**
- Balance dataset
  - The amount of "happy" images was even bigger(8k), that is why we decided to skip 3k random "happy" class images

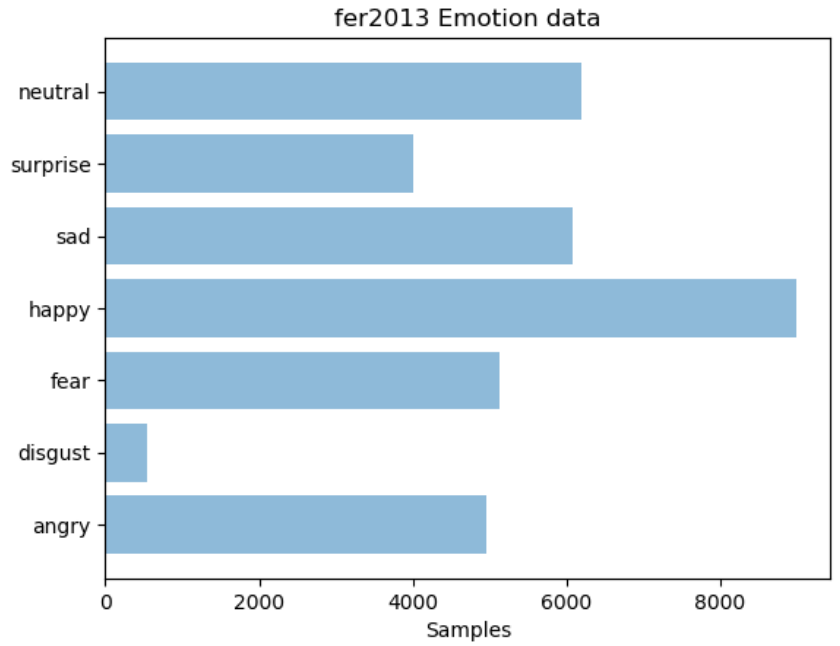


Figure [15] FER-2013 structure

Source: [15] Internet

CNN Architecture

Solution **more advanced architectures**

- **Techniques :**
  - Choosing Hyperparameters
  - ⇨ the number and dimension of the filter
  - ⇨ the number of layers,
  - ⇨ ReLU non linear function f(x) = max(0, x)
  - Pooling Layers
  - Dropout Layers

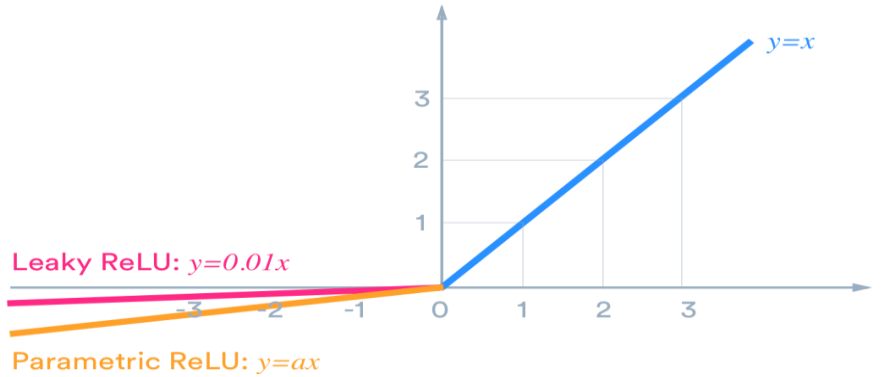


Figure [21] ReLU-non linear function

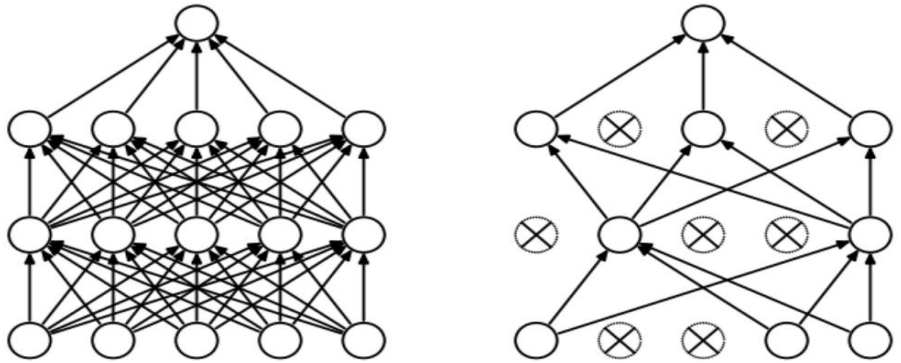


Figure [22] Dropout structure

Source: [21][22] Internet

CNN Architecture

2. Model 2 - 4 x Convolutional Layers, 1x Fully Connected

- Comment
  - This second model has **better performances** compared to the first one
  - Reaches a test accuracy up to 35% while still showing **overfitting**.
  - After few iterations it finished learning stucking on **0.15625 of test accuracy**

⇨ **This model has been discarded.**

```
Iteration i= 8050 , train accuracy= 0.953125 , loss= 0.294972
test accuracy= 0.34375

Iteration i= 8100 , train accuracy= 0.96875 , loss= 0.0559145
test accuracy= 0.3125

Iteration i= 8150 , train accuracy= 0.984375 , loss= 0.0277839
test accuracy= 0.328125

Iteration i= 8200 , train accuracy= 0.25 , loss= nan
test accuracy= 0.15625

Iteration i= 8250 , train accuracy= 0.125 , loss= nan
test accuracy= 0.15625

Iteration i= 8300 , train accuracy= 0.171875 , loss= nan
test accuracy= 0.15625

Iteration i= 8350 , train accuracy= 0.125 , loss= nan
test accuracy= 0.15625

Iteration i= 8400 , train accuracy= 0.15625 , loss= nan
test accuracy= 0.15625
```

Figure [23] Training Process of model 2

CNN Architecture

2. Model 2 - 4 x Convolutional Layers, 1x Fully Connected

- The second model consists of a **4 layer convolutional layer with a final fully connected layer.**
- The equation of the classifier is simply:

$$x = \textbf{maxpool2x2}(\textbf{\textit{ReLu}}(\textbf{\textit{ReLU}}(x * W_1 + b_1)W_2 + b_2))$$

2 times (also for W_3, b_3, W_4, b_4

$$y = \textbf{softmax}(xW_5 + b_5)$$

1,2,3 layers, 16 filters with a dimension of 7x7
4,5,6 a dimension of 5x5
AdamOptimizer with a learning rate of 0.001 s.

CNN Architecture

3. Model 3: Computational graph - 6 Layers, Conv-Relu-Maxpool, 1 Fully Connected

- The third model consists in a **6 layer convolutional layers with a final fully connected layer.**
- The equation of the classifier is simply:

$$x1 = \textbf{maxpool2x2}(\textbf{\textit{ReLu}}(\textbf{\textit{ReLU}}(x * W_1 + b_1)W_2 + b_2))$$

2 times (also for W_3, b_3, W_4, b_4)

$$x2 = \textbf{\textit{ReLU}}(x_1 * W_5 + b5)$$

There is 1 additional conv layer and finally a fully connected layer at the end.

$$y = \textbf{softmax}(x_2 * W_6 + b_6)$$

1,2,3 layers, 22 with a dimension of 8x8
4,5,6 a dimension of 4x4
AdamOptimizer with a learning rate of 0.001 s.

CNN Architecture

3. Model 3: Computational graph - 6 Layers, Conv-Relu-Maxpool, 1 Fully Connected

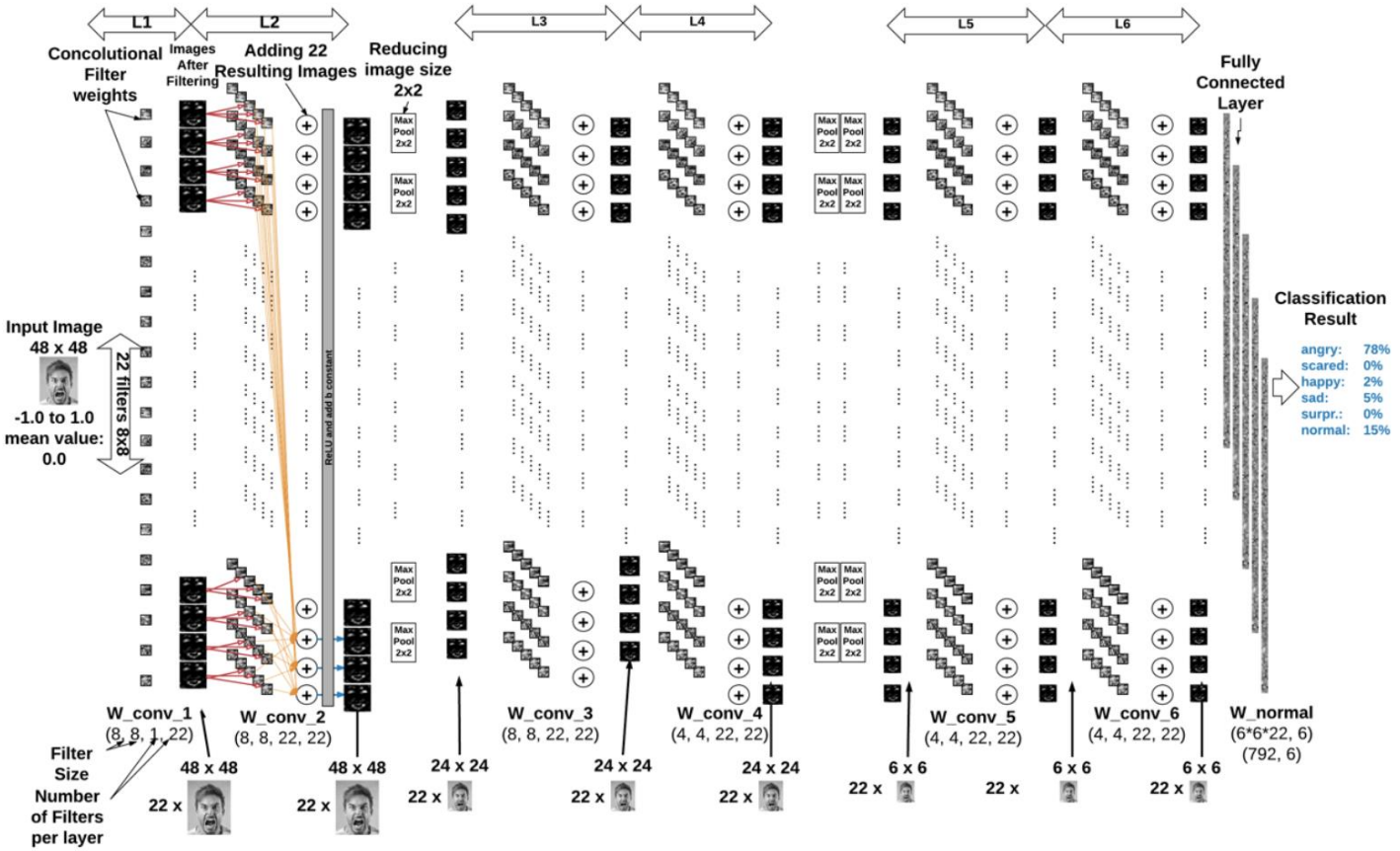


Figure [24] The chosen model

CNN Architecture

3. Model 3: Computational graph - 6 Layers, Conv-Relu-Maxpool, 1 Fully Connected

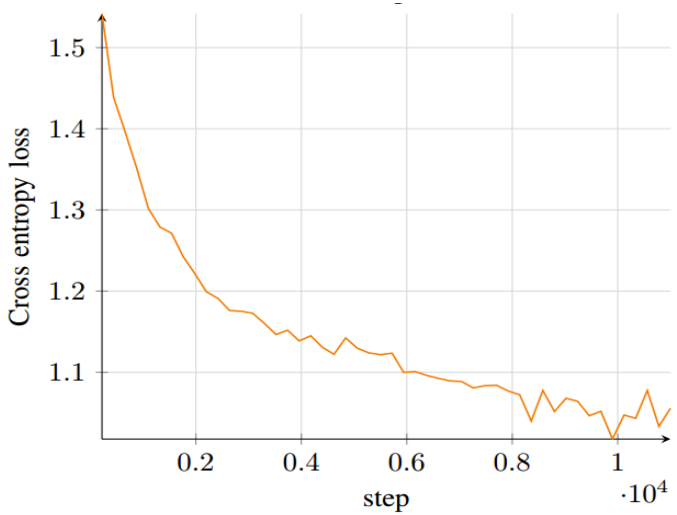


Figure [25] Cross entropy loss graph

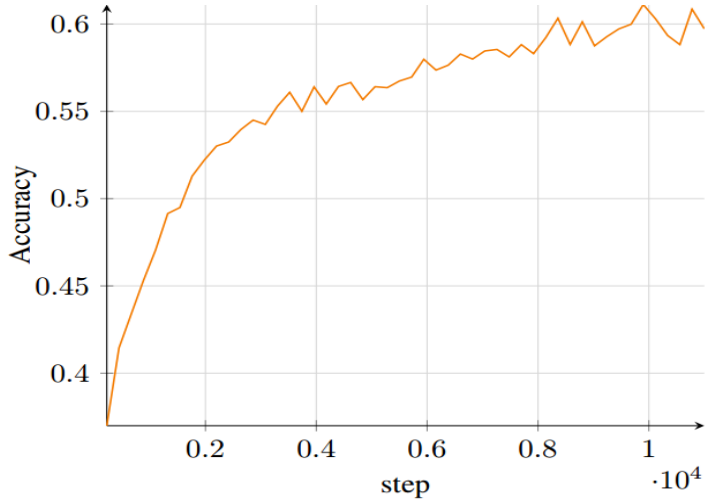


Figure [26] Accuracy of chosen model

- Comment
  - This second model reaches **a test accuracy up to 62%** which is the best result could be got. Model have **prevented the overfitting problem** observed with the previous models.

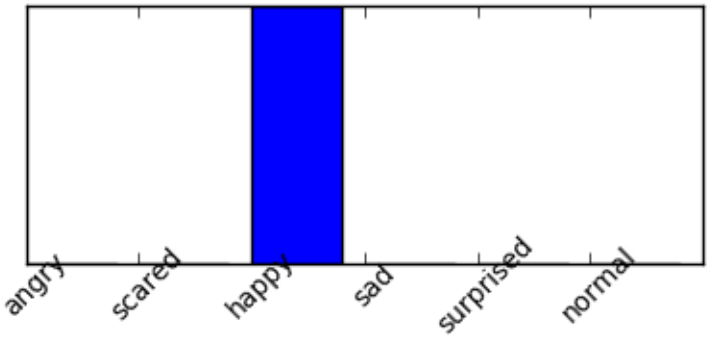
CNN Architecture

3. Model 3: Computational graph - 6 Layers, Conv-Relu-Maxpool, 1 Fully Connected

- Result:

Emotion	Accuracy
angry	**0.6956**
scared	**0.5714**
happy	**0.3157**
sad	**0.3157**
surprised	**0.7272**
normal	**0.4545**

Table [27] Model Result



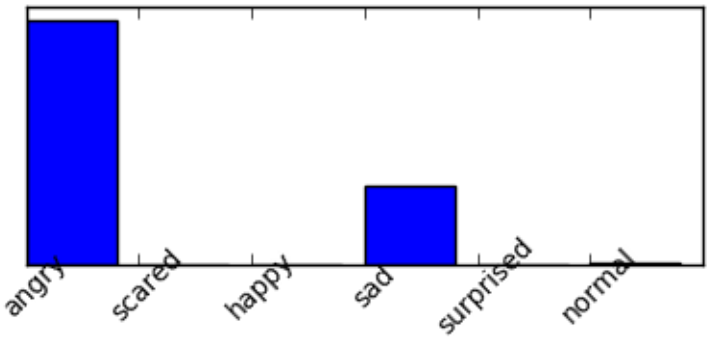


Table [28] Testing sample

CNN Architecture

3. Model 3: Computational graph - 6 Layers, Conv-Relu-Maxpool, 1 Fully Connected

- Compare to other methods:

Method	Accuracy	Test sample quantity
Shima [29]	**62.4%**	**3589**
Anonymos SVM [30]	**59.8%**	**3589**
Proposed method	**61.8%**	**3000**

Figure [31] Compare to other methods

Source: [29] Shima A., Azar *F. Convolutional Neural Networks for Facial Expression Recognition.* Proceedings of the Stanford University report http://cs231n.stanford.edu/reports/2016/pdfs/005_Report.pdf.
[30] Amine Horseman SVM for Facial Expression Recognition. A demonstrate project using SVM

CNN Architecture

1, Conclusions and comments

- The data contains a lot of noisy data, i.e. faces are rotated and of different size.
- A lot of emotions in the dataset were labeled wrong. (e.g. happy images in sad images).
- (I think) That is why we couldn't achieve very good accuracy.
- The accuracy is very good for "Happy" and "Surprised" class. These images seems to be the most "clean" as data.
- The computational power to train CNN is very high, therefore it was very time consuming to try different computational graphs.
- The facial emotion recognition is has very complicated features that were hard to explore for our computational graphs.

CNN Architecture

2, Possible improvements in the future

- For sure the CNN would perform better if the faces were always the same size and aligned straight.
- We could try another, deeper CNN architectures to extract more features.

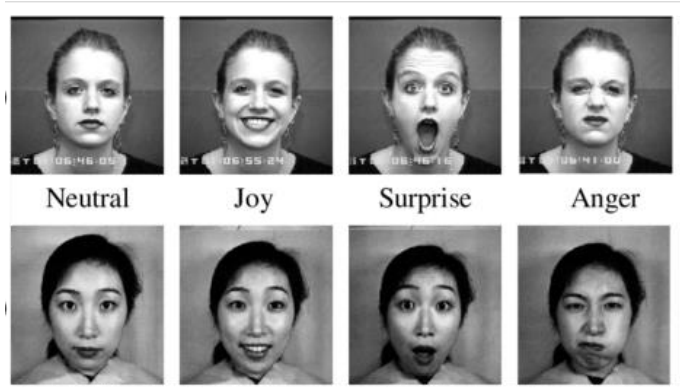


Figure [32] CK+ Dataset

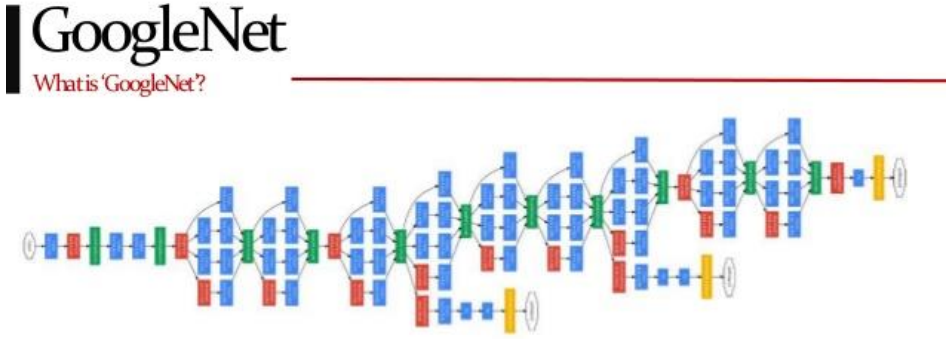


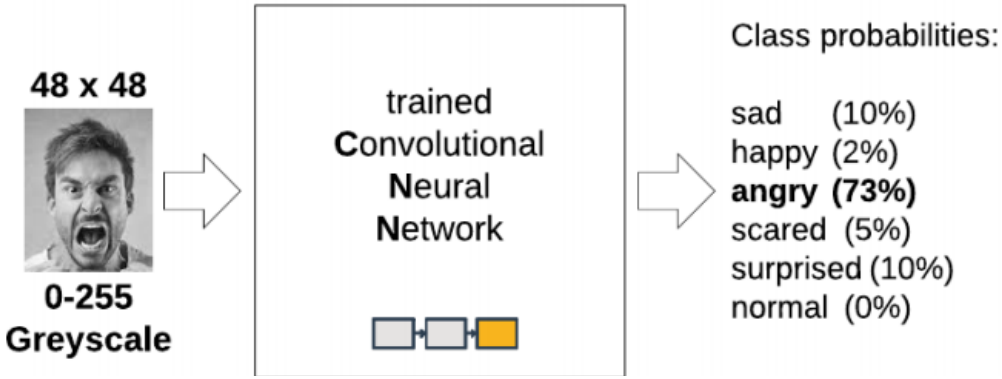
Figure [33] GoogleNet architechture

Source: [32][33] Internet

Hardware Design

CNN Architecture

3. The Trained CNN Model



	L1	L2	L3	L4	L5	L6	L7
n 2D conv to compute	22	484	484	484	484	484	6x6x22 →6 matrix multiplication
kernel length	8	8	8	4	4	4	
n img before layer (in)	1	22	22	22	22	22	
n img after layer(out)	22	22	22	22	22	22	
img length	48	48	24	24	6	6	

Figure [34] Overview the CNN

Xilinx Zynq-7000 SoC ZC706

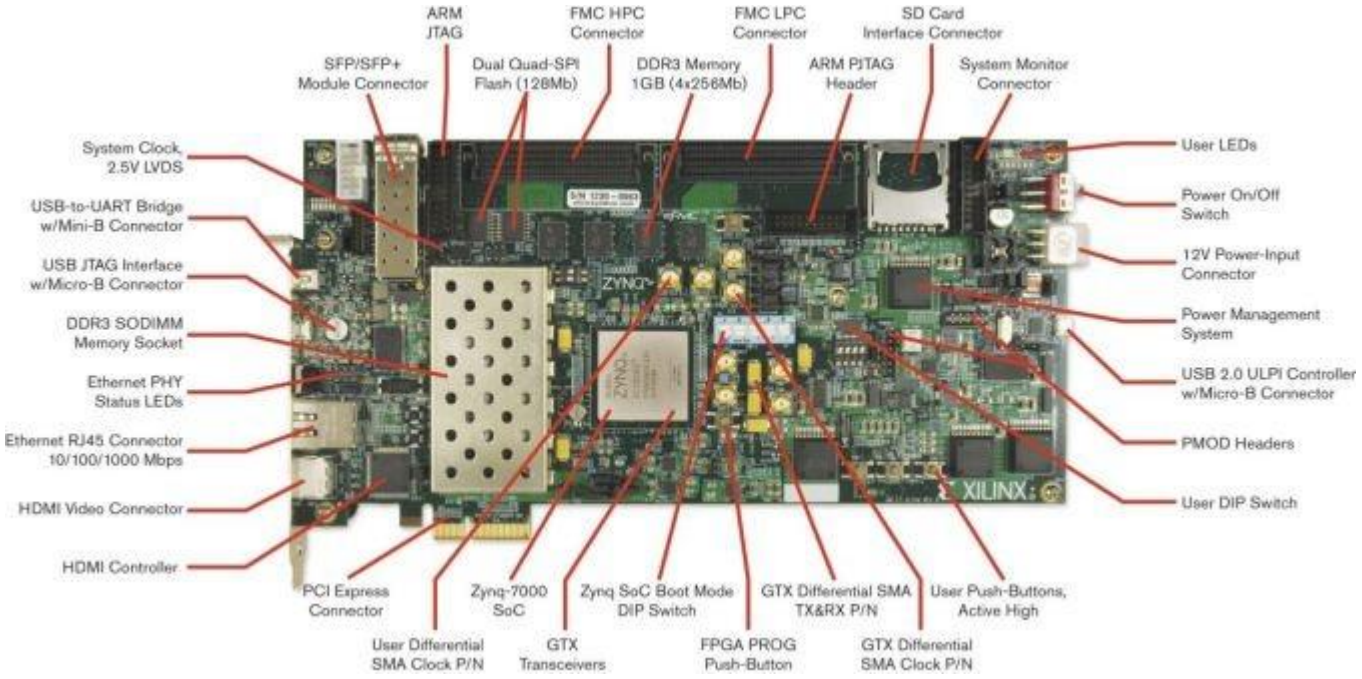


Figure [35] Xilinx Zynq-7000 SoC ZC706

	XC7Z020-CLG484-1
Logic Cells (K)	**350**
Block RAM (Mb)	**19.1**
DSP Slices	**900**
Maximum I/O Pins	**362**

Figure [36] Xilinx Zynq-7000 SoC ZC706 resource

Source: [35][36] Xilinx, https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html

CNN Architecture

- Profiling on a PC and Embedded Platform

The python and C/C++ implementations of the algorithms were used to identify the classification time for each of them and understand the acceleration

1	Macbook PC (3.4GHz Intel) in Python	–python / tensorflow	3.55 s
2	Macbook PC (3.4GHz Intel) in Python	–for loops (single thr)–	126 s
3	Macbook PC (3.4GHz Intel) in C/C++	–for loops (single thr)–	0.6 s
4	Zynq PetaLinux(900MHz) in C/C++	–for loops (single thr)–	11.5 s

Table [30] Time consuming after Profiling on a PC and Embedded Platform

- The execution in **python/tensorflow on a PC is very fast** as for the high-level language, since the **library is well optimized**. The python code (1.) is high-level, it's very close to the C++ (3.).
- Execution 3. Is the fastest, because it's executed on the **fastest machine** and the code is **most low-level (C++).**

PC and Embedded Platform

- Profiling on a PC and Embedded Platform

To identify the bottlenecks of the algorithm, the profiling was executed on the C-code. The results for each layer are shown below

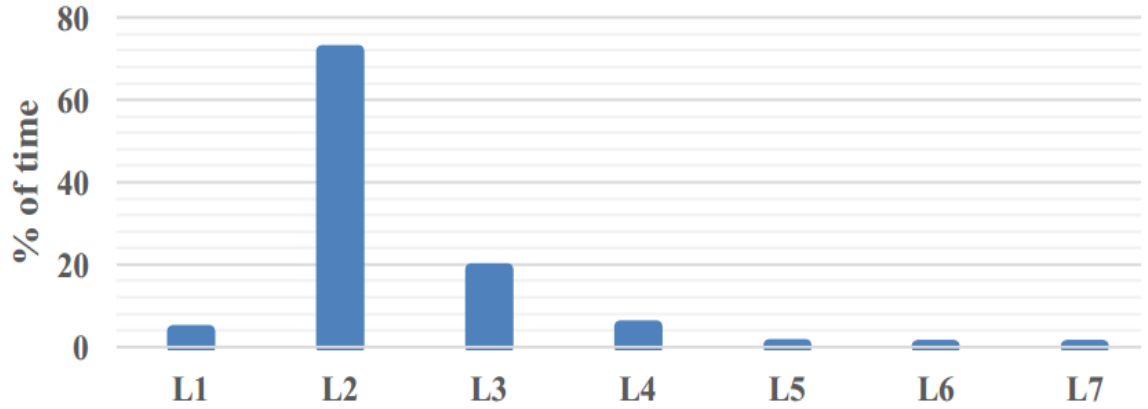


Table [a] Timing consumption of each layer

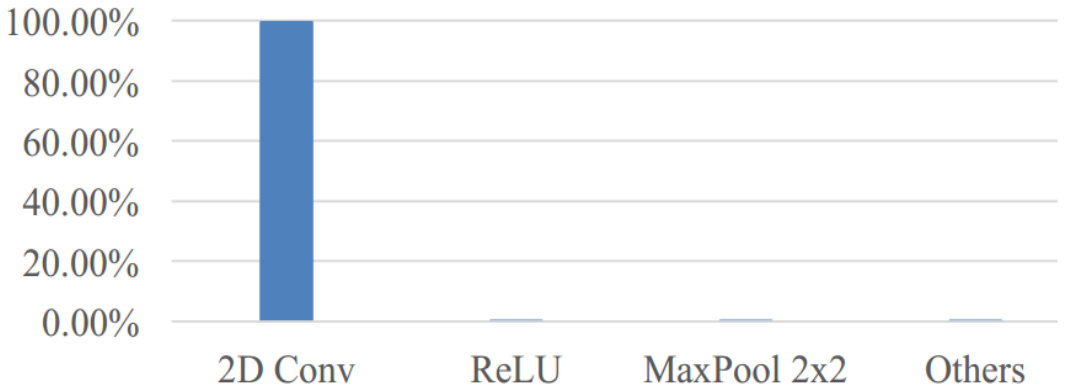


Table [b] Time consumption of each type of layer

Table [37] Time consuming after Profiling on Embedded Platform of each layer

PC and Embedded Platform

- Conclusions:
  - Acceleration of the following functions should visibly shorten the classification time:
    - 2D image convolution, because it's the most computationally heavy task executed with the most of time.
    - ReLU, because it's very easy to implement it in the FPGA as a multiplexer (if x<0 assign 0, else assign x).
  - Since we need to share a lot of data, it'd be valuable to design a solution that would also minimise this communication and therefore minimise the accelerator ←→ processor time.

Quantization

The filter coefficients (constant)					
Max Wc1:	0.581492	Avg Wc1:	0.00039835	Min Wc1:	-0.942267
Max Wc2:	0.919815	Avg Wc1:	-0.0357445	Min Wc2:	-1.39372
Max Wc3:	0.817923	Avg Wc2:	-0.014101	Min Wc3:	-1.02589
Max Wc4:	0.604326	Avg Wc3:	-0.0391957	Min Wc4:	-0.990101
Max Wc1:	0.626381	Avg Wc4:	-0.0127036	Min Wc5:	-1.15402
Max Wc1:	0.624091	Avg Wc5:	-0.0218916	Min Wc6:	-0.839514
Max Wc1:	0.68207	Avg Wn6:	-0.0214262	Min Wn6:	-1.14417
The adder coefficients b (constant):					
Max bc1:	0.0133187	Avg bc1:	-0.0378035	Min bc1:	-0.13341
Max bc2:	0.223921	Avg bc2:	0.0253297	Min bc2:	-0.243458
Max bc3:	0.307745	Avg bc3:	0.0571535	Min bc3:	-0.153931
Max bc4:	0.345763	Avg bc4:	-0.000638	Min bc4:	-0.35225
Max bc5:	0.297876	Avg bc5:	0.0265728	Min bc5:	-0.291342
Max bc6:	0.0384562	Avg bc6:	0.00243238	Min bc6:	-0.0225041
The middle results in between the layers h (calc for 64 test images) :					
Max h1:	0.424823	Avg h1:	0.0094548	Min h1:	0.0
Max h2:	2.96478	Avg h2:	0.0251095	Min h2:	0.0
Max h3:	7.43573	Avg h3:	0.267952	Min h3:	0.0
Max h4:	18.9776	Avg h4:	0.128376	Min h4:	0.0
Max h5:	23.1608	Avg h5:	0.653597	Min h5:	0.0
Max h6:	15.8007	Avg h6:	0.350184	Min h6:	0.0

Table [38] Min, avg and max values of the coefficients and example inter-layer results.

- **Reason**

Because they cannot easily handle the floating-point operations, the usual requirement for signal processing using FPGA is **quantization.**

- The quantisation sweep was performed for the following fixed-point schemes:
  - 2.5 to 2.10 - max values: (-2,2)
  - 3.5 to 3.10 - max values: (-4,4)
  - 4.5 to 4.10 - max values: (-8,8)
  - 5.5 to 5.10 - max values: (-16,16)
- The fixed point fractional values from .5 to .10 represent steps from 0.03125 to 0.00097.

Quantization

- The following experiment in C++ on float coefficients was performed:
  - 1. Prepare the set of 64 test images.
  - 2. Classify each of the image and obtain the classification probabilities for each class.
  - 3. Quantise the coefficients with the certain, signed fixed-point scheme.
  - 4. Perform the CNN algorithm on the same 64 test images, with the quantized coefficients from 3. and “flatten” each of the resulting imaged after each layer, according to maximum value that the signed point can store.
  - 5. Compare the 64*6 class probabilities with results from 2. and calculate the mean and max error.

- that all the probability values was taken into consideration when calculating the error. As an example, if the original classification probabilities were [0.1 0.1 0.5 0.2 0.0 0.1] and the result after quantisation was [0.0 0.0 0.55 0.45 0.0 0.0], the mean error is 0.108 (0.65/6) and the maximum error os 0.25 (0.45-0.2).

Quantization

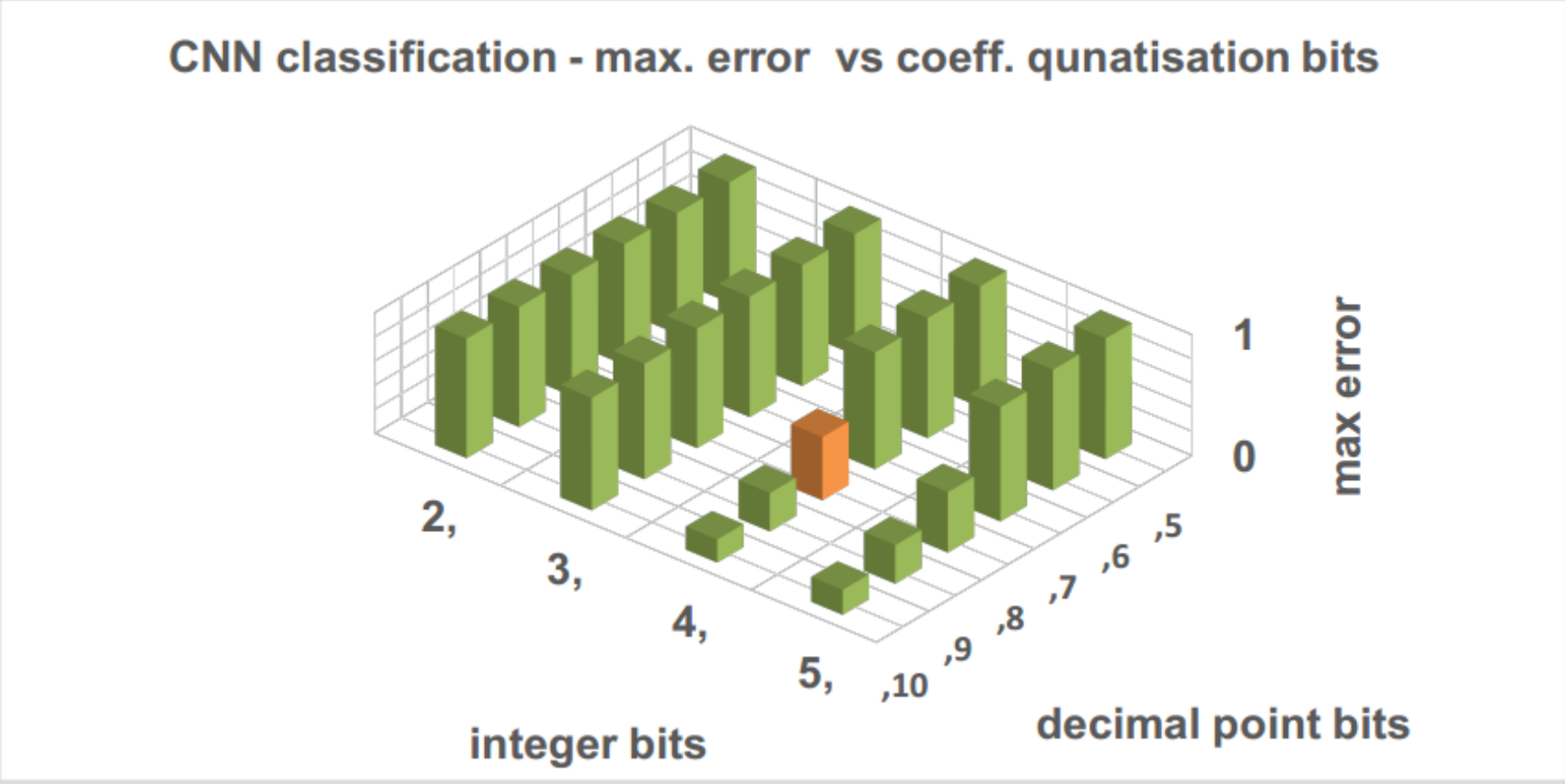


Table [39] Plots showing relative error vs. different fixed-point quantisation.

Quantization

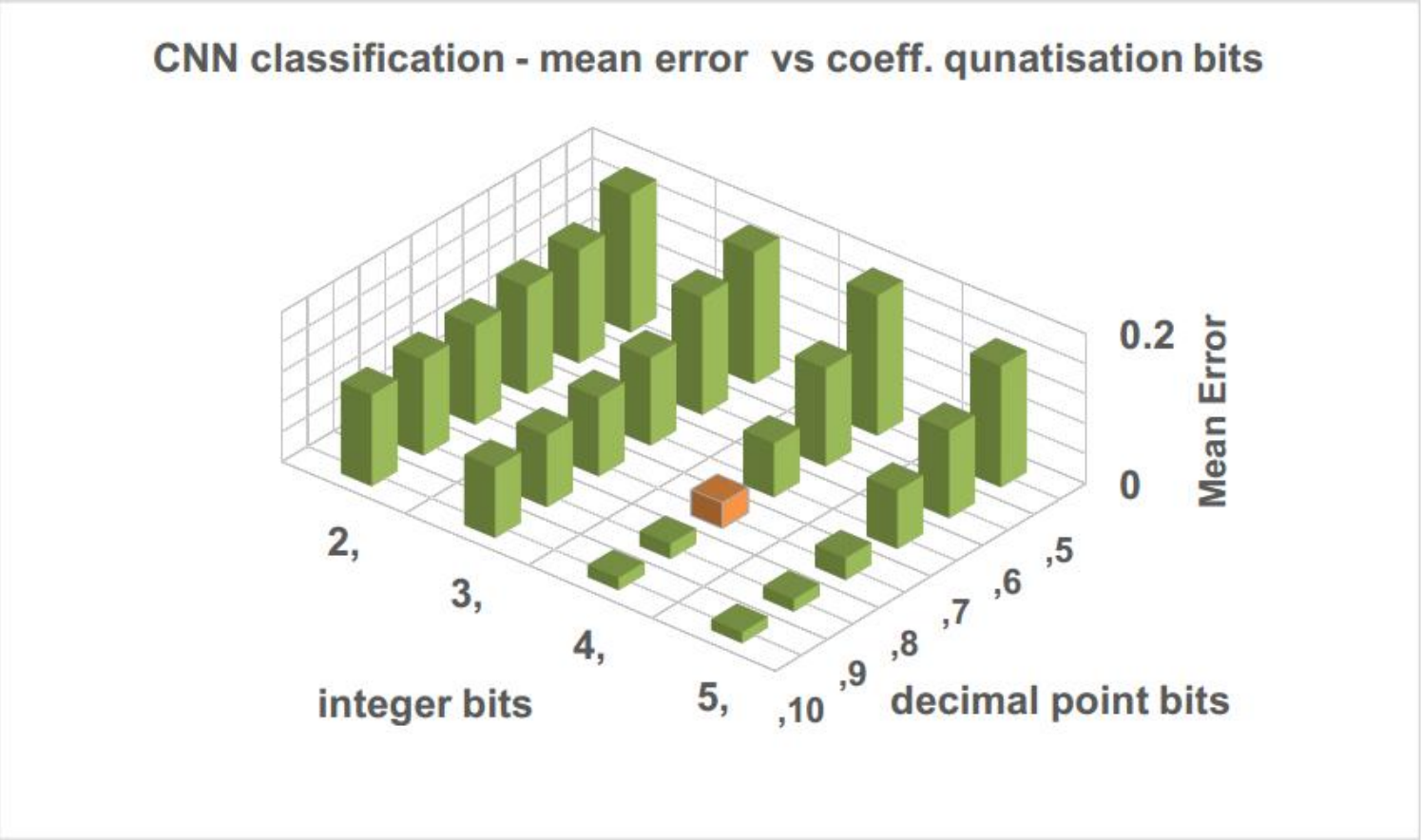


Table [40] Plots showing relative error vs. different fixed-point quantisation.

Quantization

	INTEGER BITS				
FRACTIONAL BITS		2.	3.	4.	5.
	.5	0.18 / 0.99	0.17 / 0.99	0.18 / 0.99	0.16 / 0.99
	.6	0.15 / 0.99	0.15 / 0.99	0.13 / 0.99	0.11 / 0.99
	.7	0.14 / 0.99	0.11 / 0.99	0.07 / 0.96	0.07 / 0.95
	.8	0.13 / 0.99	0.1 / 0.99	0.03 / 0.52	0.02 / 0.49
	.9	0.12 / 0.98	0.09 / 0.98	0.02 / 0.31	0.01 / 0.31
	.10	0.12 / 0.98	0.09 / 0.93	0.01 / 0.19	0.01 / 0.20

Table [41] The result of quantization.

- The quantisation error drops dramatically when increasing the number of bits. The best trade-off between the number of bits and reasonable error is fixed point 4.8.
- That is why all the future accelerators will be done for 12 bit fixed-point (4.8) per pixel.

Convolution Acceleration

- The main purpose in the FPGA acceleration is operation parallelization
  - The operation used here is multiplication – accumulation
- Solution 1

The simplified idea is shown in figure below.

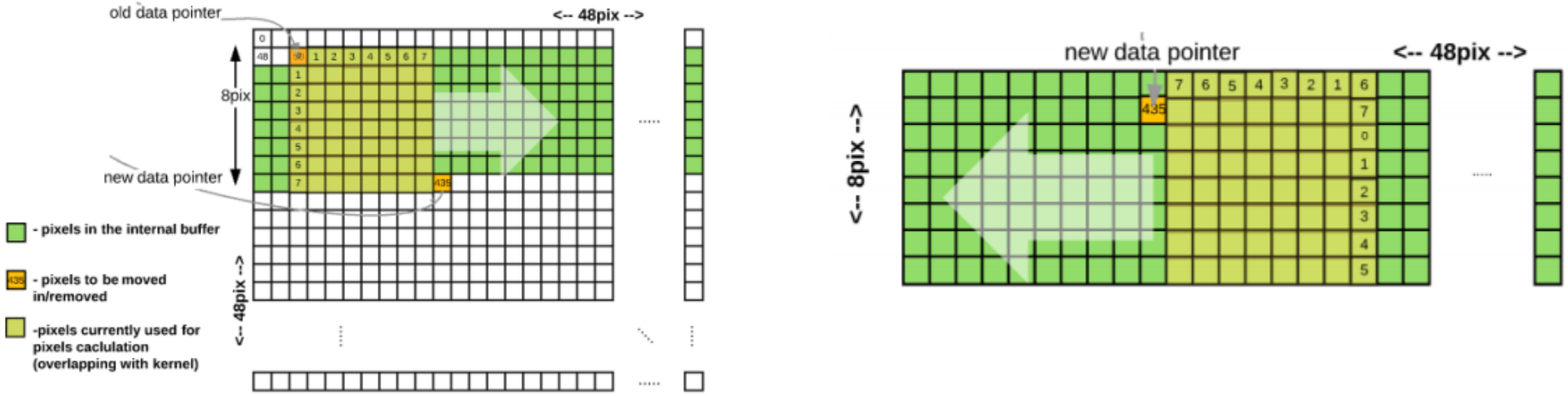


Figure [42] Block diagram showing the main idea of solution 1.

- Characteristics:
  - 1 output pixel per clock cycle (after filling the buffer at the beggining).
  - 64 Multiplications and additions done at a single clock cycle.

Convolution Acceleration

- Solution 1
  - Drawbacks:
    - The pixels shifting logic is very complicated and takes a lot of resources
    - The design takes 70% of the FPGA LUT/FF resources and most of it is for pixel shifting logic.
    - The maximum clock frequency has been set as ~70MHz
    - With the aforementioned characteristics, the estimated classification rate is every 0.5s.

Convolution Acceleration

- Solution 2

The idea of this solution is inspired from the paper “Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks [43]

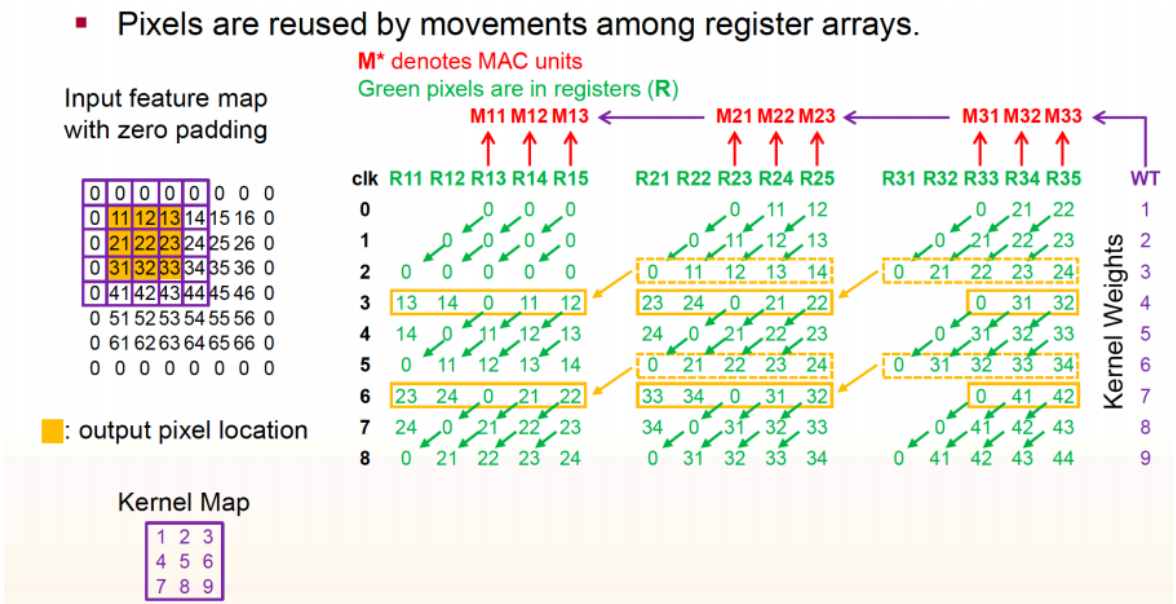


Figure [44] Slide taken from [43] showing the way of pixel re-utilization and parallelization. This example is based on 9-pixel kernel (3x3) and 9-pixel block (3x3).

Source: [43] Yufei Ma et al., *Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks*, Arizona University.

Convolution Acceleration

- Solution 2
  - Note:
    - After 9 clock cycles, the convolution results of 3x3 block of pixels is available in the DSPs. Moreover, the same pixel shifter can be used to use it with another sets of DSPs and another sets of Kernels to compute another block results in parallel.
    - The decision about the exact level of parallelization was determined on basis of the number of DSPs. In particular, 900 DSPs are available.

    The CNN algorithm has image sizes of **48x48, 24x24, 6x6** ⇨ it makes sense to make the single block size (that is computed as single convolution result after N clock cycles) as 6x6
    The CNN has 22 images as input/output, ⇨ use **6 * 6 * 22 = 792 DSPs**. This means, **88% of the available DSPs are utilized.**

Pixel Shifting Logic

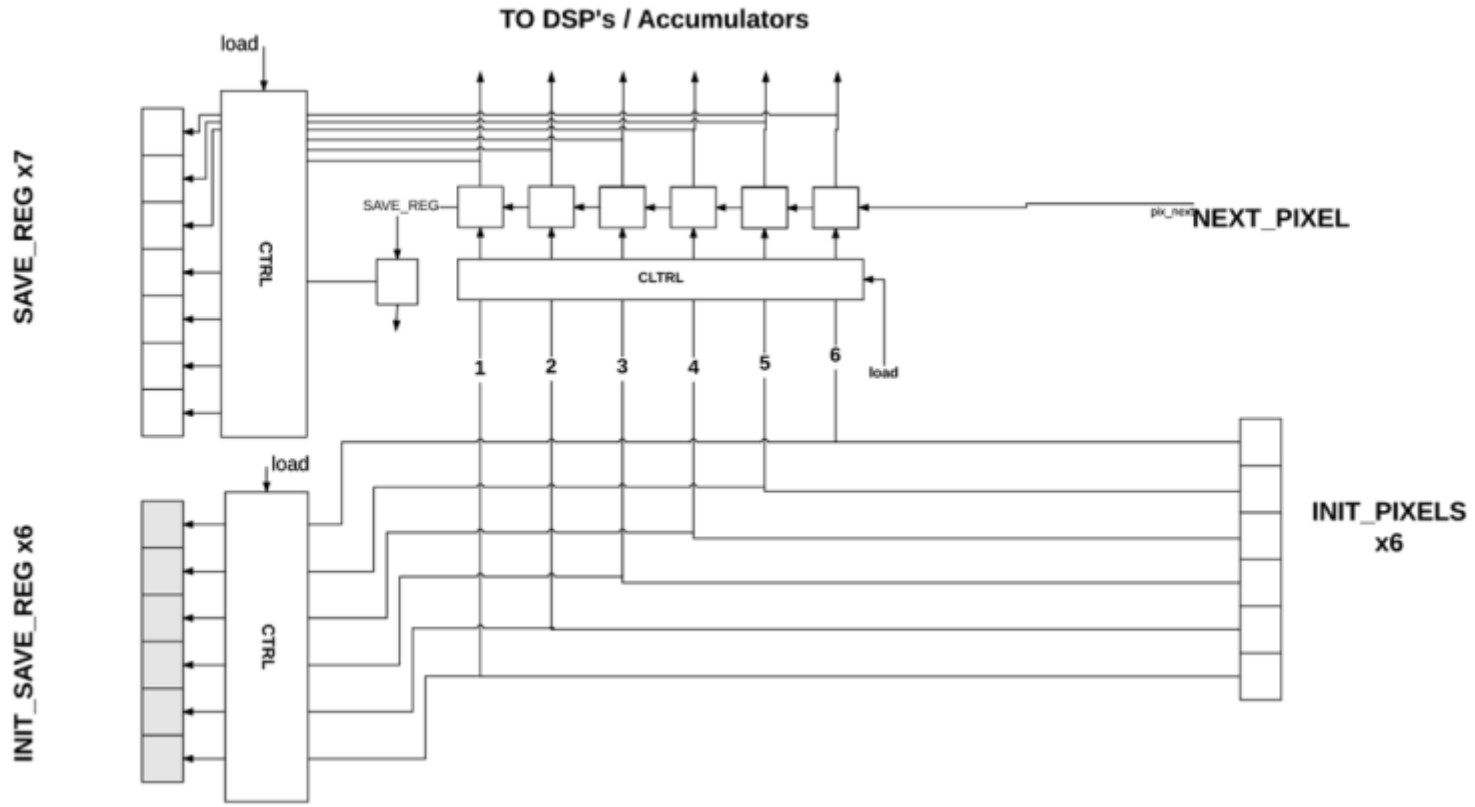


Figure [45] Block Diagram of the single block pixel shifter.

Pixel Shifting Logic

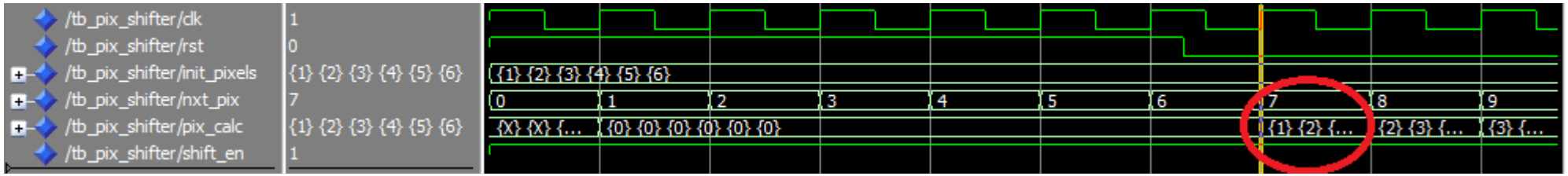


Figure [46] Test Bench showing basic operation of the pixel shifter.

- Each pixel shifter is responsible for 6 pixels ⇨ 6 of them to shift 6x6 block image. The pixels are loaded from BRAM at the first cycle, and then only last pixel shifter takes the image from different image BRAMs and all others are taking the values from the previous block.

Pixel Shifting Logic

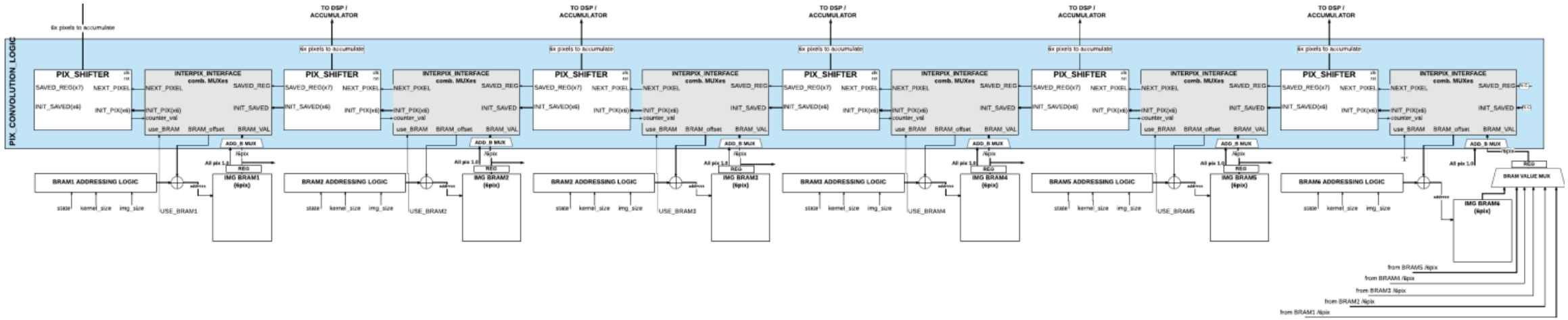


Figure [47] Diagram Showing details of pixels shifting logic.

- Each pixel shifter block needs to store 7 + 6 = 13 pixels. In the pixel logic there are 6 pixel shifter, so all the logic need only 13 * 6 = 78 registers storing 12-bit pixels.
  - Between single pixel shifter, there are combinatorial interfaces (grey) that are set of multiplexers that are interfacing the data between pixel shifters.

Pixel Shifting Logic

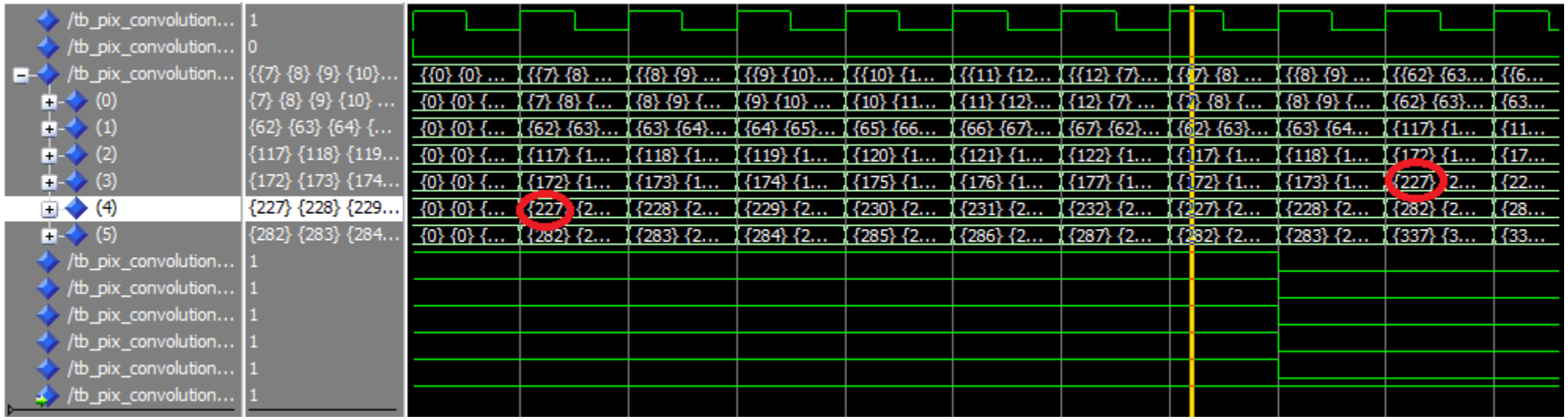
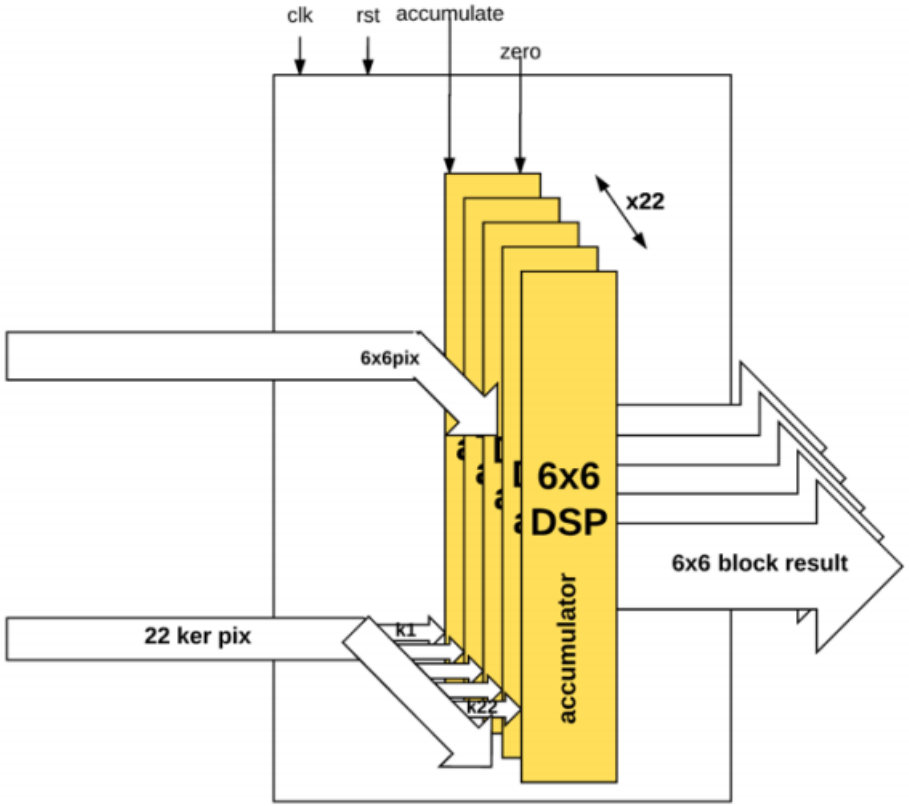


Figure [48] Test Bench showing pixel shifting logic.

Pixel Multiplication / addition



- The accumulator design handles maximum and minimum value saturation and after each cycle updates the result at the output. When accumulate enable signal is turned off, the output result is still available.

- ReLU Rectifier in Hardware
  - The function is simply a hardware realization of y = max(x,0) in signed 12 bits

Figure [49] Block diagram of the pixel accumulator block (792 DSPs).

Zero Padding

- the image size in the BRAM is 54 x 54 pixels.
- BRAM Image Structure
  - In first 8 cycles of the single block computation each pixel shifter can take the data only from single BRAM image (apart from the last one) and so that there is no repetitive data in BRAMs
  - ⇨ there are **6 BRAMS and each BRAM cell stores 6 pixels (72 bits)**.

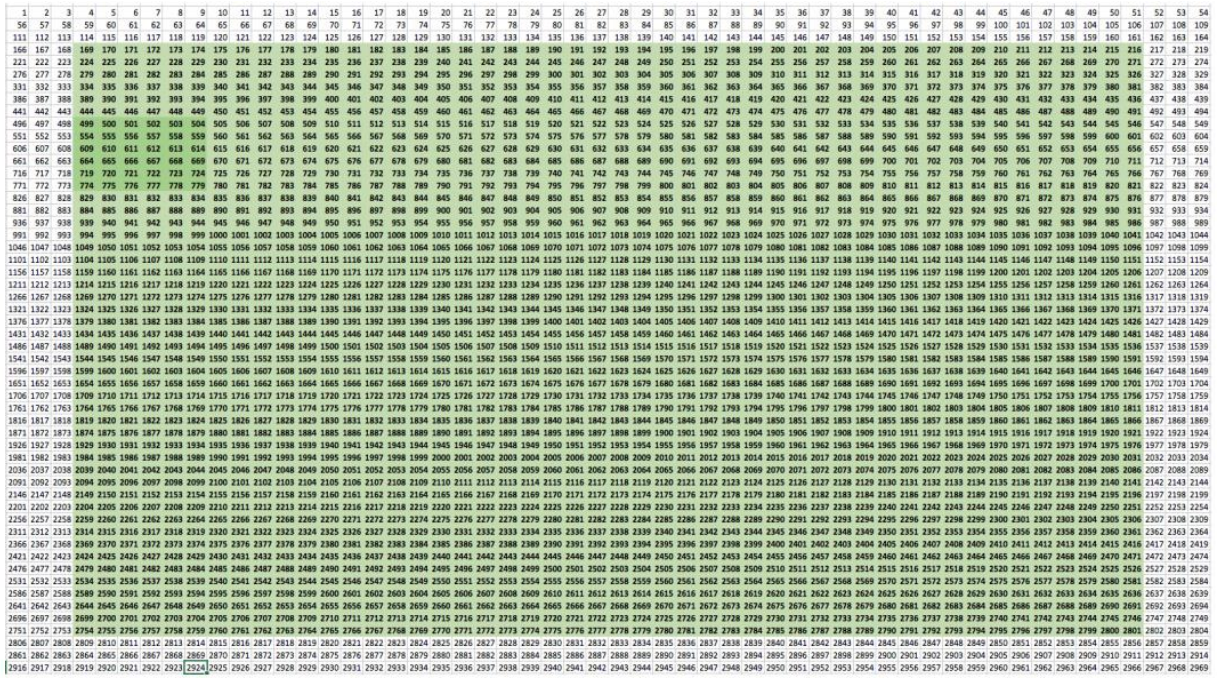


Figure [50] Zero padding mapping.

BRAM Image Structure

ADDR			BRAM1						BRAM2						BRAM3						BRAM4						BRAM5						BRAM6					
	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
0	1	2	3	4	5	6	56	57	58	59	60	61	111	112	113	114	115	116	166	167	168	169	170	171	221	222	223	224	225	226	276	277	278	279	280	281
1	7	8	9	10	11	12	62	63	64	65	66	67	117	118	119	120	121	122	172	173	174	**175**	**176**	**177**	227	228	229	**230**	**231**	**232**	282	283	284	**285**	**286**	**287**
2	13	14	15	16	17	18	68	69	70	71	72	73	123	124	125	126	127	128	**178**	**179**	**180**	181	182	183	**233**	**234**	**235**	236	237	238	**288**	**289**	**290**	291	292	293
3	19	20	21	22	23	24	74	75	76	77	78	79	129	130	131	132	133	134	184	185	186	187	188	189	239	240	241	242	243	244	294	295	296	297	298	299
4	25	26	27	28	29	30	80	81	82	83	84	85	135	136	137	138	139	140	190	191	192	193	194	195	245	246	247	248	249	250	300	301	302	303	304	305
5	31	32	33	34	35	36	86	87	88	89	90	91	141	142	143	144	145	146	196	197	198	199	200	201	251	252	253	254	255	256	306	307	308	309	310	311
6	37	38	39	40	41	42	92	93	94	95	96	97	147	148	149	150	151	152	202	203	204	205	206	207	257	258	259	260	261	262	312	313	314	315	316	317
7	43	44	45	46	47	48	98	99	100	101	102	103	153	154	155	156	157	158	208	209	210	211	212	213	263	264	265	266	267	268	318	319	320	321	322	323
8	49	50	51	52	53	54	104	105	106	107	108	109	159	160	161	162	163	164	214	215	216	217	218	219	269	270	271	272	273	274	324	325	326	327	328	329
9	331	332	333	334	335	336	386	387	388	389	390	391	441	442	443	444	445	446	496	497	498	499	500	501	551	552	553	554	555	556	606	607	608	609	610	611
10	337	338	339	**340**	**341**	**342**	392	393	394	**395**	**396**	**397**	447	448	449	**450**	**451**	**452**	502	503	504	505	506	507	557	558	559	560	561	562	612	613	614	615	616	617
11	**343**	**344**	**345**	346	347	348	**398**	**399**	**400**	401	402	403	**453**	**454**	**455**	456	457	458	508	509	510	511	512	513	563	564	565	566	567	568	618	619	620	621	622	623
12	349	350	351	352	353	354	404	405	406	407	408	409	459	460	461	462	463	464	514	515	516	517	518	519	569	570	571	572	573	574	624	625	626	627	628	629
13	355	356	357	358	359	360	410	411	412	413	414	415	465	466	467	468	469	470	520	521	522	523	524	525	575	576	577	578	579	580	630	631	632	633	634	635
14	361	362	363	364	365	366	416	417	418	419	420	421	471	472	473	474	475	476	526	527	528	529	530	531	581	582	583	584	585	586	636	637	638	639	640	641
15	367	368	369	370	371	372	422	423	424	425	426	427	477	478	479	480	481	482	532	533	534	535	536	537	587	588	589	590	591	592	642	643	644	645	646	647
16	373	374	375	376	377	378	428	429	430	431	432	433	483	484	485	486	487	488	538	539	540	541	542	543	593	594	595	596	597	598	648	649	650	651	652	653
17	379	380	381	382	383	384	434	435	436	437	438	439	489	490	491	492	493	494	544	545	546	547	548	549	599	600	601	602	603	604	654	655	656	657	658	659
18	661	662	663	664	665	666	716	717	718	719	720	721	771	772	773	774	775	776	826	827	828	829	830	831	881	882	883	884	885	886	936	937	938	939	940	941
19	667	668	669	670	671	672	722	723	724	725	726	727	777	778	779	780	781	782	832	833	834	835	836	837	887	888	889	890	891	892	942	943	944	945	946	947
20	673	674	675	676	677	678	728	729	730	731	732	733	783	784	785	786	787	788	838	839	840	841	842	843	893	894	895	896	897	898	948	949	950	951	952	953
21	679	680	681	682	683	684	734	735	736	737	738	739	789	790	791	792	793	794	844	845	846	847	848	849	899	900	901	902	903	904	954	955	956	957	958	959
22	685	686	687	688	689	690	740	741	742	743	744	745	795	796	797	798	799	800	850	851	852	853	854	855	905	906	907	908	909	910	960	961	962	963	964	965
23	691	692	693	694	695	696	746	747	748	749	750	751	801	802	803	804	805	806	856	857	858	859	860	861	911	912	913	914	915	916	966	967	968	969	970	971
24	697	698	699	700	701	702	752	753	754	755	756	757	807	808	809	810	811	812	862	863	864	865	866	867	917	918	919	920	921	922	972	973	974	975	976	977
25	703	704	705	706	707	708	758	759	760	761	762	763	813	814	815	816	817	818	868	869	870	871	872	873	923	924	925	926	927	928	978	979	980	981	982	983
26	709	710	711	712	713	714	764	765	766	767	768	769	819	820	821	822	823	824	874	875	876	877	878	879	929	930	931	932	933	934	984	985	986	987	988	989
27	991	992	993	994	995	996	1046	1047	1048	1049	1050	1051	1101	1102	1103	1104	1105	1106	1156	1157	1158	1159	1160	1161	1211	1212	1213	1214	1215	1216	1266	1267	1268	1269	1270	1271
28	997	998	999	1000	1001	1002	1052	1053	1054	1055	1056	1057	1107	1108	1109	1110	1111	1112	1162	1163	1164	1165	1166	1167	1217	1218	1219	1220	1221	1222	1272	1273	1274	1275	1276	1277
29	1003	1004	1005	1006	1007	1008	1058	1059	1060	1061	1062	1063	1113	1114	1115	1116	1117	1118	1168	1169	1170	1171	1172	1173	1223	1224	1225	1226	1227	1228	1278	1279	1280	1281	1282	1283
30	1009	1010	1011	1012	1013	1014	1064	1065	1066	1067	1068	1069	1119	1120	1121	1122	1123	1124	1174	1175	1176	1177	1178	1179	1229	1230	1231	1232	1233	1234	1284	1285	1286	1287	1288	1289
31	1015	1016	1017	1018	1019	1020	1070	1071	1072	1073	1074	1075	1125	1126	1127	1128	1129	1130	1180	1181	1182	1183	1184	1185	1235	1236	1237	1238	1239	1240	1290	1291	1292	1293	1294	1295
32	1021	1022	1023	1024	1025	1026	1076	1077	1078	1079	1080	1081	1131	1132	1133	1134	1135	1136	1186	1187	1188	1189	1190	1191	1241	1242	1243	1244	1245	1246	1296	1297	1298	1299	1300	1301
33	1027	1028	1029	1030	1031	1032	1082	1083	1084	1085	1086	1087	1137	1138	1139	1140	1141	1142	1192	1193	1194	1195	1196	1197	1247	1248	1249	1250	1251	1252	1302	1303	1304	1305	1306	1307
34	1033	1034	1035	1036	1037	1038	1088	1089	1090	1091	1092	1093	1143	1144	1145	1146	1147	1148	1198	1199	1200	1201	1202	1203	1253	1254	1255	1256	1257	1258	1308	1309	1310	1311	1312	1313
35	1039	1040	1041	1042	1043	1044	1094	1095	1096	1097	1098	1099	1149	1150	1151	1152	1153	1154	1204	1205	1206	1207	1208	1209	1259	1260	1261	1262	1263	1264	1314	1315	1316	1317	1318	1319
36	1321	1322	1323	1324	1325	1326	1376	1377	1378	1379	1380	1381	1431	1432	1433	1434	1435	1436	1486	1487	1488	1489	1490	1491	1541	1542	1543	1544	1545	1546	1596	1597	1598	1599	1600	1601
37	1327	1328	1329	1330	1331	1332	1382	1383	1384	1385	1386	1387	1437	1438	1439	1440	1441	1442	1492	1493	1494	1495	1496	1497	1547	1548	1549	1550	1551	1552	1602	1603	1604	1605	1606	1607
38	1333	1334	1335	1336	1337	1338	1388	1389	1390	1391	1392	1393	1443	1444	1445	1446	1447	1448	1498	1499	1500	1501	1502	1503	1553	1554	1555	1556	1557	1558	1608	1609	1610	1611	1612	1613
39	1339	1340	1341	1342	1343	1344	1394	1395	1396	1397	1398	1399	1449	1450	1451	1452	1453	1454	1504	1505	1506	1507	1508	1509	1559	1560	1561	1562	1563	1564	1614	1615	1616	1617	1618	1619
40	1345	1346	1347	1348	1349	1350	1400	1401	1402	1403	1404	1405	1455	1456	1457	1458	1459	1460	1510	1511	1512	1513	1514	1515	1565	1566	1567	1568	1569	1570	1620	1621	1622	1623	1624	1625

Figure [51] Single image BRAM structure (in 6 image BRAMS).

BRAM Image Structure

- BRAM Image Structure
  - It's needed to store 22 full images; therefore, it takes 81* 22 = 1782 addresses.
  - The BRAM is divided into 2 sectors of the 22-image size and for different layers images are read and write in the swapped sectors

Layer	Read Addresses	Write Addresses
L1	0-1781	1782-3563
L2	1782-3563	0-1781
L3	0-1781	1782-3563
L4	1782-3563	0-1781
L5	0-1781	1782-3563
L6	1782-3563	0-1781

Figure [52] Addressing of different image sectors.

BRAM Image Structure

- BRAM addressing
  - BRAM needs to store 72 bits per cell, its size been configured to 128 bits to be able to handle it from the processor side easily. The 72 image pixels are aligned to LSb and MSb's are padded with 0's.
  - Since the address of the BRAM is byte aligned ⇨ The real address that the image should be should be written or read is shifted left (padded 0's) 4 bits

  ⇨address 81 (0x51) will become 1296 (0x510)
- Kernel BRAMs
  - The kernel shifting logic had only been designed for 8x8 kernels, x

  ⇨The **4x4 kernels for layers L3-L6 has been padded by 0's and used as 8x8.**
  - Each kernel is **8x8 = 64** pixels and 22 kernel results are parallelized
  - The kernel pixels are also 12 bits, and it's need 22 of them in one cell to be read in parallel. ⇨**12 * 22 = 264**
  - The BRAM cell size needs to be the power of 2 ⇨ the BRAM cell size has been chosen as **512 bits**

  - 1st layer uses only 22 kernels but L2-L6 uses 484 kernels each

  ⇨ BRAM kernel cells is **64 * 1 + 5 * 64 * 22 = 7104**
  - 6 b constants for each layer, are stored at the end of the BRAM size

  ⇨ final number of cells is **7104 + 6 = 7110**

BRAM Image Structure

- Kernel BRAMs
  - It is possible to access each 32-bit part of the BRAM cell with the byte-aligned addressing.

The address space is shown below:

processing_system7_0

Data (32 address bits : 0x40000000 [1G])

axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	64K	0x4000_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0x4200_0000	64K	0x4200_FFFF
axi_bram_ctrl_2	S_AXI	Mem0	0x4400_0000	64K	0x4400_FFFF
axi_bram_ctrl_3	S_AXI	Mem0	0x4600_0000	64K	0x4600_FFFF
axi_bram_ctrl_4	S_AXI	Mem0	0x4800_0000	64K	0x4800_FFFF
axi_bram_ctrl_5	S_AXI	Mem0	0x4A00_0000	64K	0x4A00_FFFF
axi_bram_ctrl_6	S_AXI	Mem0	0x4C00_0000	512K	0x4C07_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
axi_gpio_2	S_AXI	Reg	0x4122_0000	64K	0x4122_FFFF

Figure [53] Kernel and BRAM address space in AXI bus.

State Machine

- The block diagram of the full accelerator system and state machine is shown below

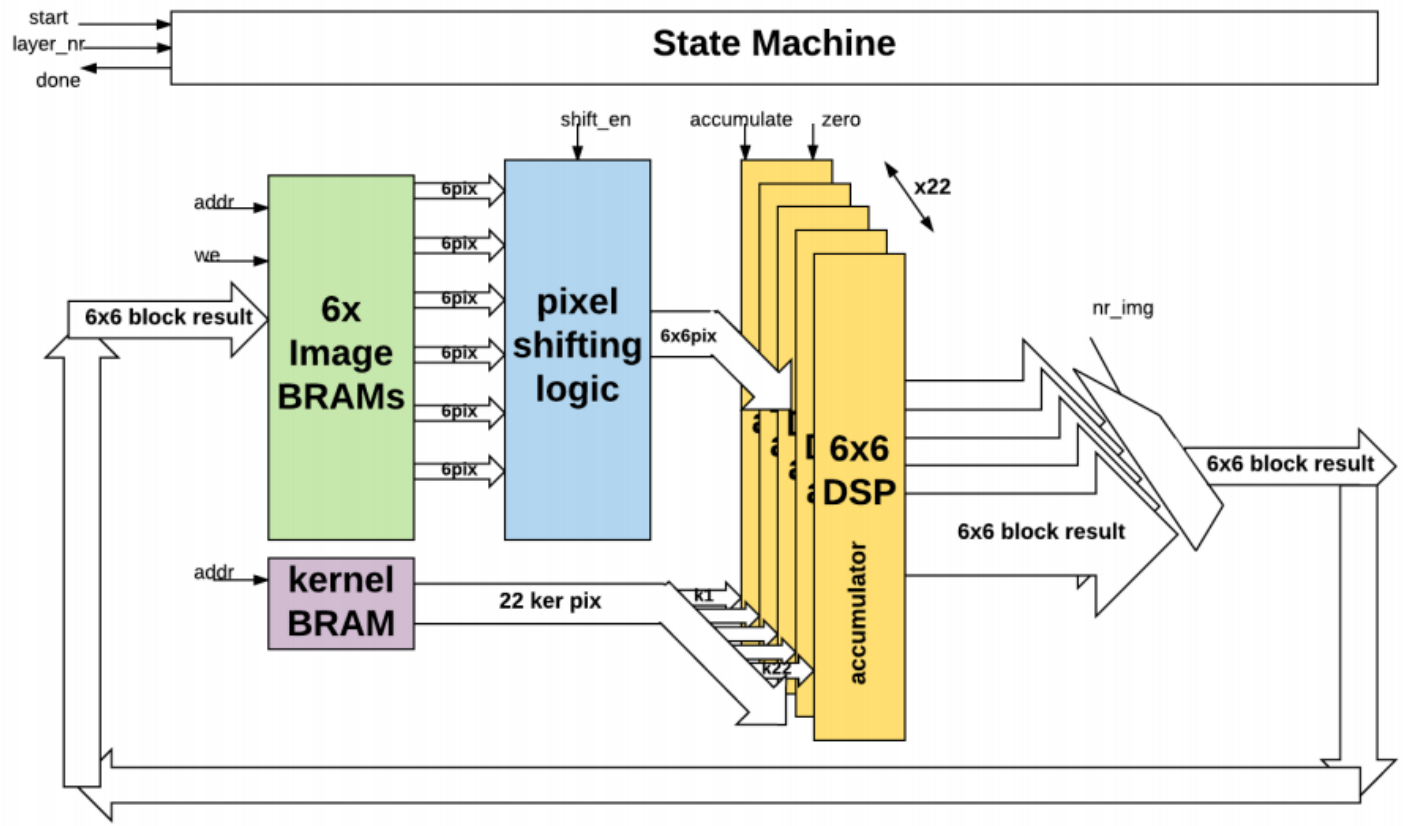
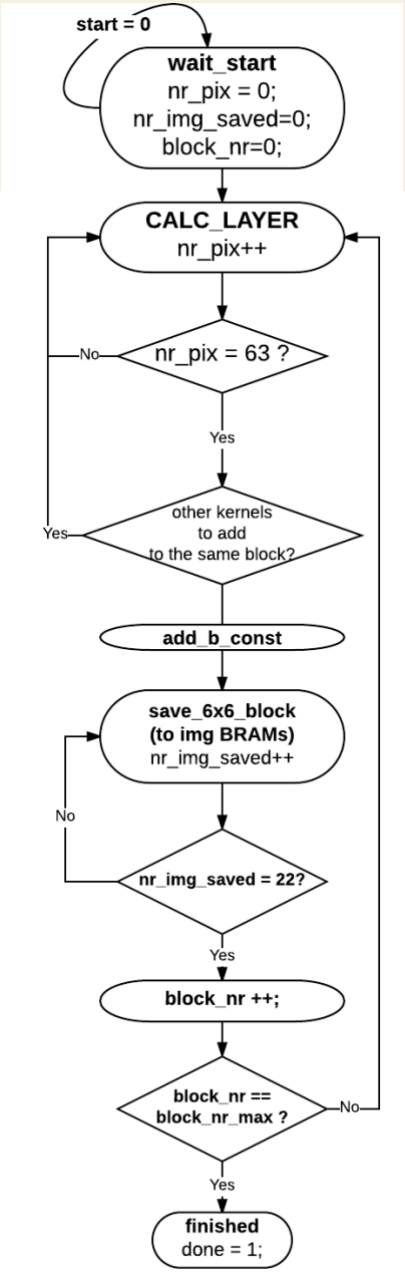


Figure [54] Simplified Block Diagram of Solution 2.



State Machine

- The main purpose of the state machine is to manage the addresses of the BRAMs, based on layer number (which determines image size, sector to R/W, number of input images etc.) and control the pixel shifting logic, DSPs and writing the result.
  - Nr_pix – counts from 0 to 63, when the DSPs are accumulating the kernel result.
  - Nr_img_saved – from 0 to 21, when saving the images needed to iterate through the DSP results.
  - Block_nr (block_row and block_col) – the results are computed in 6x6 blocks and each image has 8x8 blocks (for 48x48 image), 4x4 blocks (for 24x24 image) or 1x1 block(for 6x6 image).
- States:
  - Wait_start – waiting for start signal. Connected to GPIO.
  - CALC_LAYER – wait for the result, change the kernel pixels and shift the pixel values.
  - Add_b_const – puts the kernel address to point to the b constant for the current layer and make them accumulate it in the DSP (the pixels at this point are multiplexed to be 1.0, and therefore the DSP_value += pixel_value * b_const(from kernel BRAM), pixel value = 1.0 so DSP_value += b_const.

Figure [55] Simplified State machine Diagram of Solution 2.

State Machine

- The test bench for the whole accelerator design, as an example for L1, is shown below:

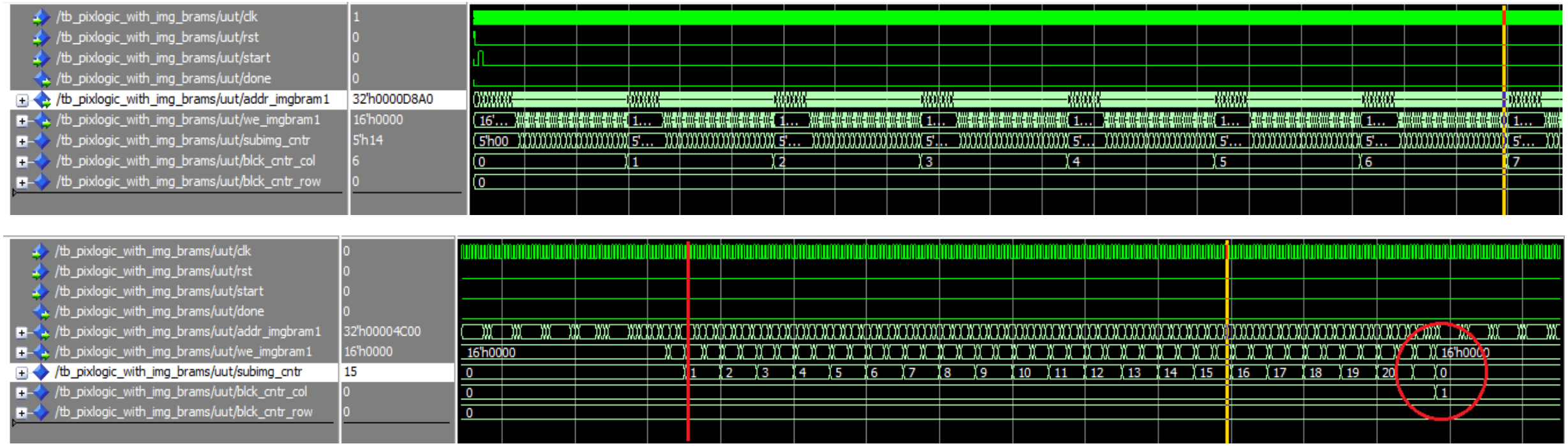


Figure [56] Test bench of the accelerator logic.

Block Design with ZynQ processor

- To be able to load the kernel coefficients and images, each BRAM ports 1 has been connected to the BRAM controller (connected to AXI bus and ZynQ) and BRAM ports 2 to the accelerator. The accelerator start, reset and layer_nr signals are managed by connecting it into a GPO, as well as the done signal to GPI.

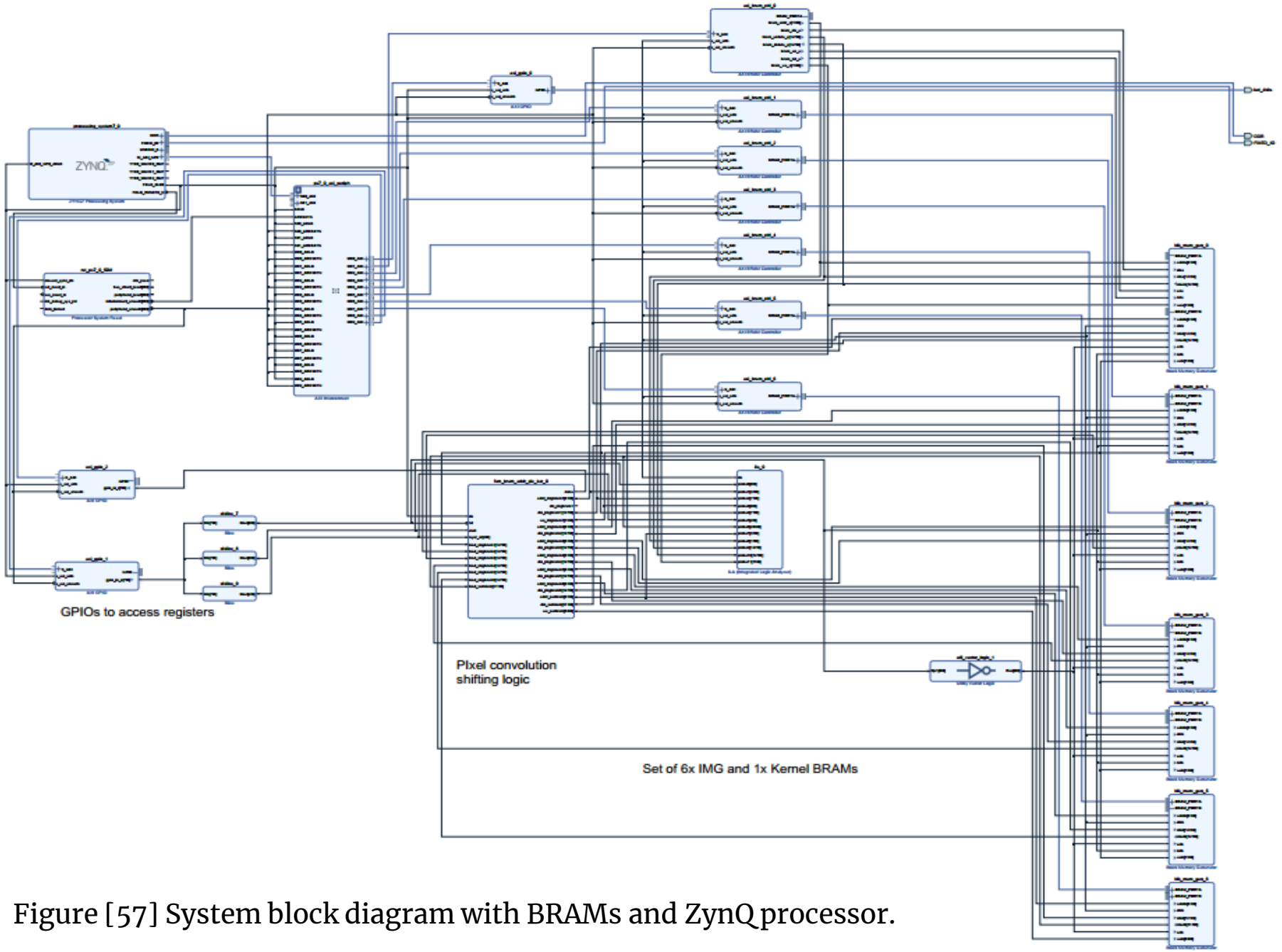
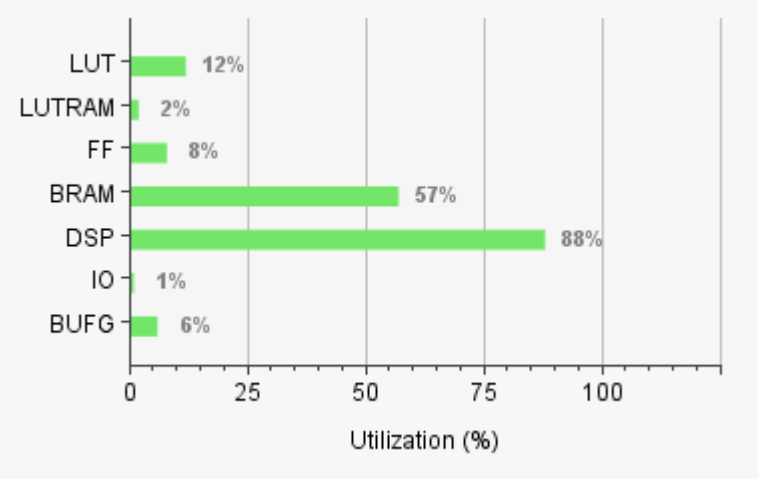


Figure [57] System block diagram with BRAMs and ZynQ processor.

Evaluation of Solution 2



Resource	Utilization	Available	Utilization %
LUT	26081	218600	11.93
LUTRAM	1533	70400	2.18
FF	35694	437200	8.16
BRAM	309.50	545	56.79
DSP	792	900	88.00
IO	4	362	1.10
BUFG	2	32	6.25

Figure [58] Resources Utilization of the ZynQ processor.

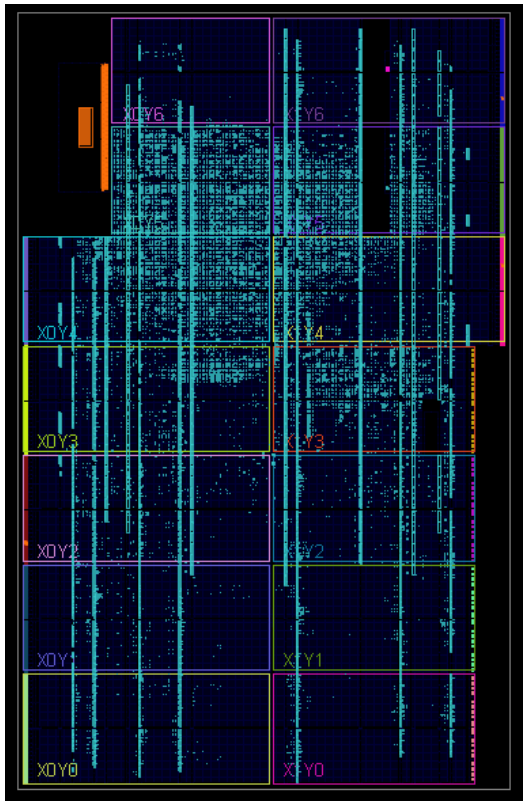


Figure [59] File Netlist of proposed Zynq.

- Observations:
  - FPGA Frequency : 110MHz
  - 792 DSPs are utilised.
  - 896kB of BRAM space is used.

Evaluation of Solution 2

- To check the timing of the solution, a ZYNQ software has been used to trigger the start of the accelerator for different layers. The timing results are shown below.

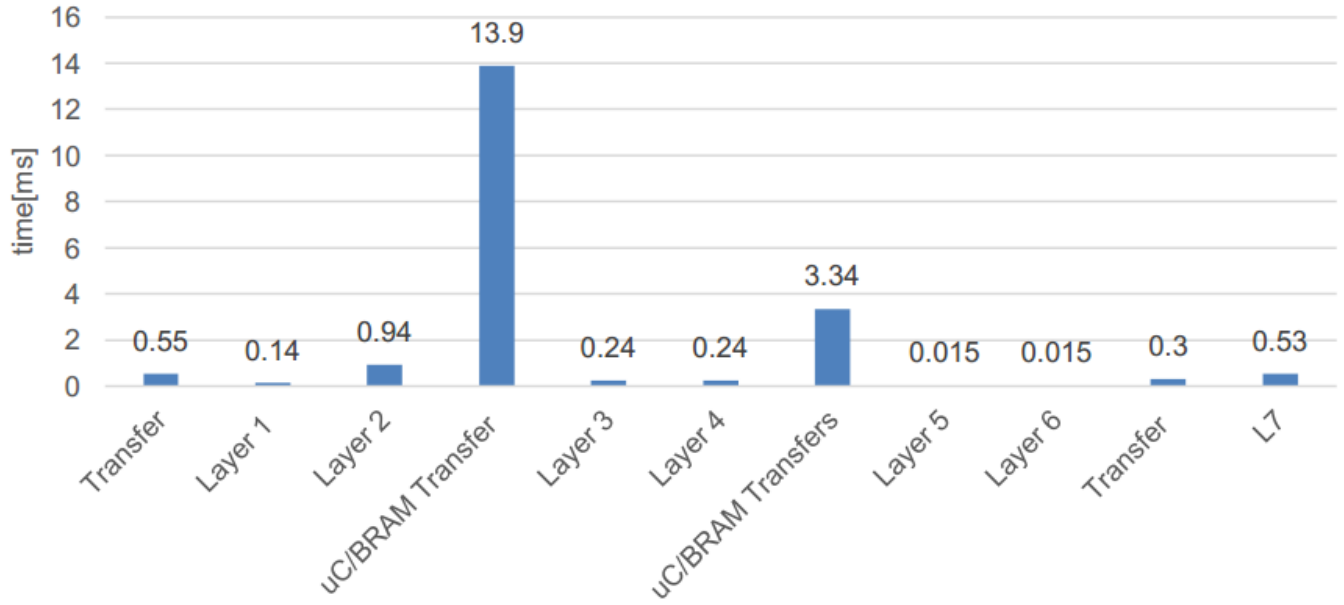


Figure [60] Final acceleration results of Sol 2- Timing.

Evaluation of Solution 2

	Software timings[s]	Accelerated timings [s]	FPGA accel. (xNR)	posiible when maxpool on FPGA (xNR)
python tensorflow	3		149	48051
Mac C/C++	0.6		30	1508
ZynQ Petalinux	11.5	0.02	571	28212

	before	now	future
Classifications per second	0.1 to 2	50	~500

Figure [61] Timing Results After Acceleration of Solution 2.

Results and Conclusions

- Acceleration ratios
  - The current acceleration results the 2D convolution is not a bottleneck any more. Thanks to multiple DSP utilization and smart pixel shifting logic the amount of time to compute convolutions and image accumulations results was reduced massively
- Analyzed the computational graph of the CNN.
  - The difficulty was mainly to understand the high-level functions used in python / tensorflow and to map it into the C code executable efficiently on the embedded platform.
- Profiled the CNN algorithm
  - After implementing fully functional classification algorithm in C, it was possible to precisely profile which operation was the bottlenech of the classification. That is why it was discovered that the 2D image convolution take more than 99% of the classification time.

Implement Result

Results

- Made the quantization analysis
  - Generated coefficients with different quantization levels and executed the algorithm with “flattening functions” to simulate the DSP behaviors. Then compared the generated classification errors in comparison to the “not quantized/flattened” coefficient solutions.
- Reviewed different accelerator architectures.
  - During the project, multiple designs of convolution acceleration has been reviewed and the solutions appropriate to the platform, the CNN and the level of complexity for the semester project has been chosen.
- Implemented and reviewed simple Solution 1
- Implemented and reviewed Solution 2
- Python Software to Generate the Constants in *.h file

Research paper

- **Hanh Phan-Xuan, Thuong Le-Tien, Sy Nguyen-Tan** (2019). *FPGA Platform applied for Facial Expression Recognition System using CNNs*.
- This work was accepted for oral presentation and published in **The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40 2019)** in Belgium.
- Beside Elsevier publisher also selected this work to publish on "*The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40 2019) / Aliated Workshops*", **Procedia Computer Science**, ISSN= 1877-0509, Vol. 151, pp. 651-658. Indexed by Scimago.

Conclusion

Challenges and future work

- **Implement Max Pooling in hardware**
  ⇨If I can reduce the bottleneck of MaxPooling (implement it in hardware), the acceleration results can be obtained much faster, i.e. at least every ~3ms.
- **Make 4x4 kernels pixel shifting logic**
  - Currently the pixel shjifting logic is working only for 8x8 kernels and 4x4 kernels are padded by 0's and treated as 8x8 kernels. The number of computations required for computing 4x4 kernels is much smaller (single block result in 16 not 64 cycles)

  ⇨the obtained acceleration could be up to 8 times faster for L4-L6
- **Improve the model accuracy**

References

- **D**. **Ververidis**, and **C. Kotropoulos**. (2004) "*Automatic speech classification to five emotional states based on gender information* " Proceedings of the EUSIPCO2004 Conference, Austria: 341-344.

- **Cowie, R., Douglas-Cowie, E., Tsapatsoulis, N., Votsis, G., Kollias, S., Fellenz, W., and Taylor**. (2001) "*Emotion recognition in human computer interaction* " IEEE Signal Processing magazine, Vol. 18, No. 1: 32-80.

- **Yann LeCun, Yoshua Bengio, and Georey Hinton**. (2015) "*Deep learning*." Nature. Vol.521: doi 10.1038/nature14539.

- **Amine Horseman**. (2007) "*SVM for Facial Expression Recognition.*" A demonstrate project using SVM.: https://github.com/amineHorseman/facial-expression-recognition-svm/commit/aecd525367f6d77e5d274de7c6f0166d5bfa4bb9.

- **Shima A., Azar F**. (2016) "*Convolutional Neural Networks for Facial Expression Recognition*." Proceedings of the Stanford University report.: 3-4. http://cs231n.stanford.edu/reports/2016/pdfs/005_Report.pdf.

Demo

Thank you for your attention!

Department of EEE, Ho Chi Minh City University of Technology, Vietnam.

You can find me at:

tansyab1@gmail.com

Tel.: +84-964-025-859